

EMBEDDED CONTROL HANDBOOK



Microchip





Microchip

Embedded Control Handbook

**SERVING A COMPLEX AND COMPETITIVE
WORLD WITH FIELD-PROGRAMMABLE
EMBEDDED CONTROL
SYSTEM SOLUTIONS**

"Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Inc. with respect to the accuracy or use of such information, or infringement of patents arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights."

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

The Microchip logo and name are registered trademarks of Microchip Technology Incorporated.

PICMASTER, PRO MASTER, PICSTART, PICSEE, PICPRO, TrueGauge, Total Endurance, SEEVAL and Smart Serial are trademarks of Microchip Technology Inc.

MACINTOSH is a trademark of Apple Computer, Inc.

IBM, IBM PC, PC/XT and AT are registered trademarks of IBM Corp.

SMC is a trademark of Standard Microsystems Corp.

DSPLAY is a trademark of Burr Brown Corp.

DFDP is a trademark of Atlanta Signal Processing Inc.

MathCad is a trademark of MathSoft Inc.

Microsoft, Mouse Systems, MS-DOS are registered trademarks of Microsoft Corp.

Windows is a trademark of Microsoft Corp.

I²C is a trademark of Philips Corporation.

MicrowireTM and tri-state are trademarks of National Semiconductor

Mouse Systems is a registered trademark of MSC Technologies, Inc.

Data I/O is a registered trademark of Data I/O Corporation.

Unisite, PROMLINK, PROCOMM, SITE, Site 48, ChipSite, PinSite, SetSite, HiTerm and HandlerSite are trademarks of Data I/O Corporation.

PROCOMM is a registered trademark of Datastorm Technologies, Inc.

ALLPRO is a trademark of Logical Devices, Inc.

All other trademarks mentioned herein are the property of their respective companies.

All rights reserved. Copyright © 1993, Microchip Technology Inc.



Table of Contents

	<u>Page</u>
SECTION 1 INTRODUCTION TO EMBEDDED SOLUTIONS FROM MICROCHIP	
Embedded Control Overview	1- 1
PIC16/17 Microcontroller Families Overview/Road Map	1- 1
PIC16/17 Naming Convention	1- 3
Low-Voltage Operation of PIC16/17 Microcontrollers	1- 4
Serial EEPROM Overview	1- 4
One-Time-Programmable EPROM Overview	1- 4
The Advantages of One-Time-Programmable	1- 4
Application-Specific Standard Product Family	1- 4
Ease of Production Utilizing Quick Turn Programming (QTP) and Serialized Quick Turn Programming (SQTP)	1- 6
SECTION 2 PIC16C5X APPLICATION NOTES	
Comparison of 8-Bit Microcontrollers - AN520	2- 1
Software Interrupt Techniques - AN514	2- 11
Software Stack Management - AN527	2- 15
Power-Up Considerations - AN522	2- 19
Implementation of an Asynchronous Serial I/O - AN510	2- 23
Using PIC16C5X as a Smart I ² C™ Peripheral - AN541	2- 41
PLD Replacement - AN511	2- 59
Analog to Digital Conversion - AN513	2- 79
Implementing Ohmmeter/Temperature Sensor - AN512	2- 85
Interfacing to AC Power Lines - AN521	2- 91
Implementing Wake-Up on Keystroke - AN528	2- 93
Multiplexing LED Drive and Keypad - AN529	2- 97
Implementing a Simple Serial Mouse Controller - AN519	2- 121
Intelligent Remote Positioner - AN531	2- 133
Programming PIC16C5X Devices on Data I/O Unisite - AN524	2- 149
Programming PIC16C5X Devices on Logical Devices ALLPRO - AN525	2- 153
Utility Math Routines - AN526	2- 155
Implementing an LCD Controller - AN563	2- 215
SECTION 3 PIC16CXX APPLICATION NOTES	
Using the Analog to Digital Converter - AN546	3- 1
Implementing Wake Up on Keystroke - AN552	3- 21
PortB as External Interrupt - AN566	3- 25
Table Read Using PIC16CXX - AN556	3- 29
Software Implementation of I ² C Bus Master - AN554	3- 33
Software Implementation of Asynchronous Serial I/O - AN555	3- 121
Four Channel Digital Volt Meter with Display and Keyboard - AN557	3- 157
SECTION 4 PIC17CXX APPLICATION NOTES	
Saving and Restoring Status on Interrupt - AN534	4- 1
Implementing Table Read and Write - AN548	4- 3
Frequency and Resolution Options for PWM Outputs - AN539	4- 7
Using PWM to Generate Analog Output - AN538	4- 15
Using the PWM - AN564	4- 17
Using the Capture Module - AN545	4- 29
Serial Port Utilities - AN547	4- 61
Utility Math Routines - AN544	4- 71
Implementing IIR Digital Filters - AN540	4- 121
Implementation of Fast Fourier Transforms - AN542	4- 139
Tone Generation - AN543	4- 161
Servo Control of a DC Brush Motor - AN532	4- 197







Table of Contents

	<u>Page</u>
SECTION 5 INTERFACING PIC16/17 WITH SERIAL EEPROMS	
Communicating with I ² C Bus Using PIC16C5X - AN515	5- 1
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X - AN530	5- 11
Logic Powered Serial EEPROMs - AN535	5- 29
SECTION 6 SERIAL EEPROMS TUTORIALS AND APPLICATION NOTES	
Basic Serial EEPROM Operation - AN536	6- 1
Everything a System Engineer Needs to Know About	
Serial EEPROM Endurance - AN537	6- 15
Using the Microchip Endurance Predictive Software - AN562	6- 23
Interfacing 24LCXX Serial EEPROMs to the PIC16C54 - AN567	6- 27
Using the 24C65 and 24C32 with Stand-alone PIC16/17 Code - AN558	6- 51
24C01A Compatibility Issues and Its Mobility for Memory Upgrade - AN517	6- 91
Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559	6- 93
Using the 93LC56 and 93LC66 - AN560	6- 97
ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues - AN516	6- 115
1.8 Volt Technology - Benefits	6- 117
Serial EEPROM Solutions vs. Parallel Solutions	6- 119
SECTION 7 DEVELOPMENT TOOLS	
Introduction to Microchip Development Tools	7- 1
Application-Specific Standard Products Division:	
PICSEE™ Tools Product Brief	7- 3
Logic Product Division:	
MPALC Cross Assembler Product Brief	7- 5
MPASM Universal Assembler/Linker Product Brief	7- 7
MPSIM Simulator Product Brief	7- 9
PICMASTER™-16X System Product Brief	7- 11
PICMASTER-17 System Product Brief	7- 15
PRO MASTER™ Product Brief	7- 19
PICSTART™-16B Product Brief	7- 21
Memory Products Division:	
Total Endurance™ Predictive Software Model	7- 23
Serial EEPROM Evaluation Board	7- 25
Microchip Bulletin Board Service (BBS)	7- 27
SECTION 8 ARTICLE REPRINTS FOR PIC16/17 MICROCONTROLLERS	
16C5X Lean and Mean PIC* Machines	8- 1
16C71 Enhanced PIC16Cxx Gets ADC and Interrupts	8- 7
16C84 Microchip PIC Adds EEPROM Data Memory	8- 8
17C42 Using the PIC Micro	8- 9
17C42 PIC17C42 Based DC Motor Control (Japanese)	8- 11
PICSTART-16B Take your Pick with Controllers	8- 17
Serial EEPROM Microchip Technology Rewrites the EEPROM	8- 18
Serial EEPROM Serial EEPROM for Embedded Applications	8- 19
Endurance A Tool for Calculating EEROM Endurance	8- 23
APPENDIX A OFFICE LOCATIONS	
Factory Representatives	A-1- 1
Distributors	A-2- 1
Field Sales Offices	A-3- 1

*PIC is registered trademark of Microchip Technology Inc. in the U.S.A. and a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.

5

6

7

8

CROSS REFERENCE GUIDE TO APPLICATION NOTES - ALPHABETICAL

	<u>Page</u>
24C01A Compatibility Issues and Its Mobility for Memory Upgrade	AN517 6- 69
Analog to Digital Conversion	AN513 2- 79
Basic Serial EEPROM Operation	AN536 6- 1
Communicating with I ² C Bus Using PIC16C5X	AN515 5- 1
Comparison of 8-Bit Microcontrollers	AN520 2- 1
ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues	AN516 6- 91
Everything a System Engineer Needs to Know About	
Four Channel Digital Volt Meter with Display and Keyboard	AN557 3- 157
Frequency and Resolution Options for PWM Outputs	AN539 4- 7
Implementing a Simple Serial Mouse Controller	AN519 2- 121
Implementing an LCD Controller	AN563 2- 215
Implementing IIR Digital Filters	AN540 4- 121
Implementing Ohmmeter/Temperature Sensor	AN512 2- 85
Implementing Table Read and Write	AN548 4- 3
Implementing Wake-Up on Keystroke	AN528 2- 93
Implementing Wake Up on Keystroke	AN552 3- 21
Implementation of an Asynchronous Serial I/O	AN510 2- 23
Implementation of Fast Fourier Transforms	AN542 4- 139
Intelligent Remote Positioner	AN531 2- 133
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X	AN530 5- 11
Interfacing 24LCXX Serial EEPROMs to the PIC16C54	AN567 6- 27
Interfacing to AC Power Lines	AN521 2- 9 i
Logic Powered Serial EEPROMs	AN535 5- 29
Multiplexing LED Drive and Keypad	AN529 2- 97
1.8 Volt Technology - Benefits	AN550 6- 117
Optimizing Serial Bus Operations with Proper Write Cycle Times	AN559 6- 71
PLD Replacement	AN511 2- 59
PortB as External Interrupt	AN566 3- 25
Power-Up Considerations	AN522 2- 19
Programming PIC16C5X Devices on Data I/O Unisite	AN524 2- 149
Programming PIC16C5X Devices on Logical Devices ALLPRO	AN525 2- 153
Saving and Restoring Status on Interrupt	AN534 4- 1
Serial EEPROM Endurance	AN537 6- 15
Serial EEPROM Solutions vs. Parallel Solutions	AN551 6- 119
Serial Port Utilities	AN547 4- 61
Servo Control of a DC Brush Motor	AN532 4- 197
Software Implementation of Asynchronous Serial I/O	AN555 3- 121
Software Implementation of I ² C Bus Master	AN554 3- 33
Software Interrupt Techniques	AN514 2- 11
Software Stack Management	AN527 2- 15
Table Read Using PIC16CXX	AN556 3- 29
Tone Generation	AN543 4- 161
Using PIC16C5X as a Smart I ² C Peripheral	AN541 2- 41
Using PWM to Generate Analog Output	AN538 4- 15
Using the Capture Module	AN545 4- 29
Using the PWM	AN564 4- 17
Using the Analog to Digital Converter	AN546 3- 1
Using the Microchip Endurance Predictive Software	AN562 6- 23
Using the 24C65 and 24C32 with Standalone PIC16/17 Code	AN558 6- 33
Using the 93LC56 and 93LC66	AN560 6- 75
Utility Math Routines (PIC16C5X)	AN526 2- 155
Utility Math Routines (PIC17CXX)	AN544 4- 71



CROSS REFERENCE GUIDE TO APPLICATION NOTES - NUMERICAL

	<u>Page</u>
AN510	Implementation of an Asynchronous Serial I/O2- 23
AN511	PLD Replacement2- 59
AN512	Implementing Ohmmeter/Temperature Sensor2- 85
AN513	Analog to Digital Conversion2- 79
AN514	Software Interrupt Techniques2- 11
AN515	Communicating with I ² C Bus Using PIC16C5X5- 1
AN516	ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues6- 91
AN517	24C01A Compatibility Issues and Its Mobility for Memory Upgrade6- 369
AN519	Implementing a Simple Serial Mouse Controller2- 121
AN520	Comparison of 8-Bit Microcontrollers2- 1
AN521	Interfacing to AC Power Lines2- 91
AN522	Power-Up Considerations2- 19
AN524	Programming PIC16C5X Devices on Data I/O Unisite2- 149
AN525	Programming PIC16C5X Devices on Logical Devices ALLPRO2- 153
AN526	Utility Math Routines (PIC16C5X)2- 155
AN527	Software Stack Management2- 15
AN528	Implementing Wake-Up on Keystroke2- 93
AN529	Multiplexing LED Drive and Keypad2- 97
AN530	Interfacing 93CX6 Serial EEPROMs to the PIC16C5X5- 11
AN531	Intelligent Remote Positioner2- 133
AN532	Servo Control of a DC Brush Motor4- 197
AN534	Saving and Restoring Status on Interrupt4- 1
AN535	Logic Powered Serial EEPROMs5- 29
AN536	Basic Serial EEPROM Operation6- 1
AN537	Everything a System Engineer Needs to Know About Serial EEPROM Endurance6- 15
AN538	Using PWM to Generate Analog Output4- 15
AN539	Frequency and Resolution Options for PWM Outputs4- 7
AN540	Implementing IIR Digital Filters4- 121
AN541	Using PIC16C5X as a Smart I ² C Peripheral2- 41
AN542	Implementation of Fast Fourier Transforms4- 139
AN543	Tone Generation4- 161
AN544	Utility Math Routines (PIC17CXX)4- 71
AN545	Using the Capture Module4- 29
AN546	Using the Analog to Digital Converter3- 1
AN547	Serial Port Utilities4- 61
AN548	Implementing Table Read and Write4- 3
AN550	1.8 Volt Technology - Benefits6- 117
AN551	Serial EEPROM Solutions vs. Parallel Solutions6- 119
AN552	Implementing Wake Up on Keystroke3- 21
AN554	Software Implementation of I ² C Bus Master3- 33
AN555	Software Implementation of Asynchronous Serial I/O3- 121
AN556	Table Read Using PIC16CXX3- 29
AN557	Four Channel Digital Volt Meter with Display and Keyboard3- 157
AN558	Using the 24XX65 and 24C32 with Standalone PIC16/17 Code6- 33
AN559	Optimizing Serial Bus Operations with Proper Write Cycle Times6- 71
AN560	Using the 93LC56 and 93LC666- 75
AN562	Using the Microchip Endurance Predictive Software6- 23
AN563	Implementing an LCD Controller2- 215
AN564	Using the PWM4- 17
AN566	PortB as External Interrupt3- 25
AN567	Interfacing 24LCXX Serial EEPROMs to the PIC16C546- 27



Microchip



CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT

Please note that application software written for one family is easily converted to fit another.

Subject	PIC16C5X	PIC16CXX	PIC17CXX	Serial EEPROM
24CXX serial EEPROM interface	AN515			
93CX6 serial EEPROM interface	AN530			
A/D conversion	AN513	AN546		
AC power line, interface to	AN521			
Addition, 16+16	AN526		AN544	
Addition, floating point	AN526		AN544	
Alarm clock implementation	AN529			
Analog to digital conversion	AN513			
Asynchronous serial port implementation in software	AN510			
BCD addition	AN526		AN544	
BCD conversion routines	AN526		AN544	
BCD subtraction	AN526			
BCD to binary	AN526		AN544	
Binary to BCD	AN526			
Brown-out circuits	AN522			
Capacitance measurement	AN512			
Capture routines			AN545	
Chip select with Vcc				AN535
Clock/calendar implementation	AN529			
Compatibility - 24C01A				AN517
Compatibility - 93C06				AN516
D/A using PWM			AN538	
Digital voltmeter		AN557		
Digital filters			AN540	
Division, 16/16, unsigned	AN526		AN544	
Division, 16/16, signed	AN526		AN544	
DTMF generation			AN543	
Endurance				AN537
Endurance predictive model				AN562
FFT			AN542	
Floating point addition	AN526		AN544	
Floating point multiplication	AN526		AN544	
Floating point routines	AN526		AN544	
Floating point subtraction	AN526		AN544	
Frequency measurement			AN545	
I/O expansion			AN547	
I ² C implementation (for serial EE interface only)	AN515			
I ² C implementaion	AN541	AN554		
IIR filter			AN540	

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	Serial EEPROM
Interfacing to 24C32/65				AN558
Interfacing to 24CXX				AN515
Interfacing to 24LCXX				AN567
Interfacing to 93CX6				AN530
Interfacing to 93LC56/66				AN560
Interrupt, PORTB		AN566		
Interrupt, software	AN514			
Keypad interface	AN528, AN529	AN557		
LCD controller	AN563			
LED display interfacing	AN529	AN557		
Math routines	AN526		AN544	
Measuring capacitance	AN512			
Measuring frequency			AN545	
Measuring period			AN545	
Measuring pulse-width			AN545	
Measuring resistance	AN512			
Measuring temperature	AN512			
Microwire interface	AN531			
Motor control	AN531		AN532	
Mouse, serial	AN519			
Multiplexing LED and keypad	AN529			
Multiplication, 8 x 8, unsigned	AN526		AN544	
Multiplication, 16 x 16, unsigned	AN526		AN544	
Multiplication, 16 x 16, signed	AN526		AN544	
Multiplication, floating point	AN526		AN544	
Parity generation			AN547	
Period measurement			AN545	
PID	AN531		AN532	
PLA implementation	AN511			
PLD replacement using PIC16CXX	AN511			
Position control	AN531		AN532	
Positioner	AN531			
Power on Reset	AN522			
Pseudo-random number generation			AN544	
Pulse width measurement			AN545	
PWM, using			AN564	
PWM, choosing frequency			AN539	
PWM, choosing resolution			AN539	
PWM generation in software	AN531			
PWM routines			AN539	



CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16C6X	PIC17CXX	Serial EEPROM
PWM to analog output			AN538	
Quadrature encoder interface			AN532	
Random number generation, Gaussian			AN544	
Random number generation			AN544	
Real time clock implementation (from AC power line)	AN521			
Resistance measurement with PIC16CXX	AN512			
RS232 interface			AN547	
SEEPROM operation				AN536
Sequencer implementation	AN511			
Serial EEPROM interfacing	AN515			
Serial port (USART) routines		AN555	AN547	
Servo control			AN532	
Sine wave generation			AN543	
Software stack	AN527		AN534	
Square root	AN526		AN544	
Stack management in software	AN527		AN534	
State machine implementation	AN511			
Status save and restore	AN534			
Subtraction, 16+16	AN526		AN544	
Subtraction, floating point	AN526		AN544	
Table Read		AN556	AN548	
Temperature measurement	AN512			
Thermostat implementation	AN512			
Tone generation			AN543	
Trajectory generation			AN532	
UART, software implementation	AN510			
USART routines			AN547	
Velocity control			AN532	
Voltmeter implementation		AN157		
Wake up on key-stroke	AN528	AN552		
Write cycle optimization				AN559
Zero crossing detect	AN521			



Microchip

SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS

**Motivated by customer
requirements....**

"Microchip Technology draws its impetus from the technology expectations of a large base of longstanding customers. Microchip is small enough to respond quickly with technology to serve our customers' needs. Moreover, as a fully integrated IC manufacturer, Microchip deploys its panoply of resources to act timely and efficiently, and on a worldwide scale: Technology Development, Design, Wafer Fabrication, Assembly and Test, Quality, Reliability and Customer Support.

**...and powered by continuous
improvement...**

"Worldwide competition leaves no room for divergence or mediocrity. Microchip Technology, committed to focus on and continuously improve all the aspects of its business, has a unique corporate culture. To improve performance, our employees are encouraged to analyze their methods continually. Personal empowerment expands the capability of personal responsibility to continually serve our customers better.

**...riding, leading and pushing
the wave of technological
change.**

"Our industry's life-line is innovation. The fast pace of technological change is inherent in our industry. Microchip Technology has accelerated the rate of change of its technology and products to leadership in providing user-programmable space-sensitive embedded control solutions.

"Change is our ally. Driving and managing customer-focused change is our winning strategy."

Steve Sanghi

Steve Sanghi
President & Chief Executive Officer



MICROCHIP TECHNOLOGY INCORPORATED

Company Profile

HIGHLIGHTS

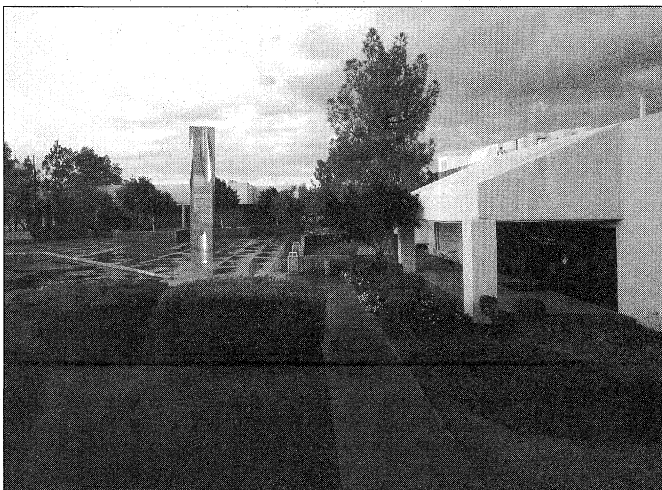
- Focused on providing field-programmable embedded control solutions
- Offers RISC 8-bit user-programmable microcontrollers and supporting logic products
- Offers Serial and Parallel EEPROMs and EPROMs
- A unique corporate culture dedicated to continuous improvement
- Research and development of high performance field-programmable products
- A history of innovation
- An experienced executive team focussed on innovation
- Fully integrated manufacturing
- A global network of manufacturing and customer support facilities

BUSINESS SCOPE

Microchip Technology Inc. manufactures and markets a variety of VLSI CMOS semiconductor components to support the field-programmable embedded control market. In particular, the company specializes in highly integrated, field-programmable RISC microcontrollers and related non-volatile memory products to meet growing market requirements for high performance, yet economical embedded control capability in an increasing number of price-sensitive products. Microchip's products feature the industry's most economical OTP (one-time programmable) capability, along with the compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.

MARKET FOCUS

Microchip targets selected markets where our advanced designs, progressive process technology and industry leading operating speeds enable us to deliver decidedly superior performance. The firm has positioned itself to maintain a dominant role as a supplier of high performance field-programmable microcontrollers and associated memory and logic products for embedded control applications.



Company headquarters near Phoenix, Arizona; executive offices, R & D and wafer fabrication occupy this 142,000-square-foot facility.



Microchip

GUIDING VALUES

Customers Are Our Focus

We establish successful customer partnerships by exceeding customer expectations for products, services and attitude. We earn our credibility through meeting commitments and producing quality products and services in a timely fashion. We believe each employee must effectively serve their internal customers in order for Microchip's external customers to be properly served.

Quality Comes First

We will perform correctly the first time, maintain customer satisfaction and measure our quality against requirements. We practice effective and standardized improvement methods, such as statistical process control to anticipate problems and implement root cause solutions. We believe that when quality comes first, reduced costs follow.

Continuous Improvement Is Essential

We utilize the concept of 'Vital Few' to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Employees Are Our Greatest Strength

We design jobs and provide opportunities in a fashion which clearly promotes pride in work, integrity, trust, teamwork, creativity, employee involvement and development, fairness, and productivity. We base recognition, advancement and compensation on an employee's achievement of excellence in team and individual performance. We provide for employee health and welfare by offering a competitive, comprehensive employee benefits program.

Products And Technology Are Our Foundation

We commit to ongoing investments and advancements in the design and development of our manufacturing process, device circuit and system technologies, which provide innovative, reliable and cost-effective products to support current and future market opportunities.

Total Cycle Times Are Competitive

We focus resources to optimize cycle times to our customers by empowering employees to achieve efficient cycle times in their area of responsibility. We believe that cycle time reduction is achieved by streamlining processes through the systematic removal of barriers to productivity.

Safety Is Never Compromised

We place our concern for safety of our employees and community at the forefront of our decisions, policies and actions. Each employee is responsible for safety.

Profits Provide For Everything We Do

We strive to maintain competitive profits as they allow continued investments and future growth, and indicate the overall success of Microchip.

Communication Is Vital

We encourage open, honest, constructive, and ongoing communication in all company and community relationships to resolve issues, exchange information and share knowledge.

Suppliers, Representatives, And Distributors Are Our Partners

We maintain mutually beneficial partnerships with suppliers, representatives, and distributors who are an integral link in the achievement of our mission and guiding values.

Professional Ethics Are Practiced

We manage our business and treat customers, employees, shareholders, investors, suppliers, distributors, representatives, community and government in a manner that exemplifies our honesty, ethics and integrity. We recognize our responsibility to the community and are proud to serve as an equal opportunity employer.

Microchip Technology Incorporated

FULLY INTEGRATED MANUFACTURING

Microchip delivers fast turnaround through total control over all phases of production. Research and development, design, mask making, wafer fabrication, assembly and quality assurance testing are conducted at facilities owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced statistical process control (SPC), and a continuous improvement culture has brought forth tight product consistency levels and high yields which enable Microchip to compete successfully in world markets. Microchip's unique approach to SPC provides customers with excellent costs, quality, reliability and on-time delivery.

A GLOBAL NETWORK OF PLANTS AND FACILITIES

Microchip is a global competitor providing local service to the world's technology centers. The Company's focal point is the design and technology advancement facility in Chandler, Arizona. Product and technology development is here, along with front-end wafer fabrication and electrical probing.

Microchip's assembly and test facility in Kaohsiung, Taiwan houses the technology and modern assembly methods necessary for plastic and ceramic packaging. Other quality-conscious firms which fabricate wafers in the Pacific Rim use Microchip's Kaohsiung plant for assembly.

Sales and application offices are located in key cities throughout the Western Hemisphere, Pacific Rim and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical support, purchasing information and failure analysis.

A PRODUCT FAMILY OF SHARED STRENGTHS

Microchip's product focus is CMOS field-programmable microcontrollers, non-volatile memories and peripherals, and application-specific standard products (ASSP). These product lines include PIC16/17 microcontrollers, EEPROMs, high-speed EPROMs, and peripherals in a broad range of product densities, speeds and packages.

MICROCONTROLLERS

PIC16/17 microcontrollers from Microchip combine high performance, low cost and small package size. They offer the best price/performance ratio in the industry. Large numbers of these devices are used in automotive and cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, security and telecommunication applications.

The widely-accepted CMOS PIC16CXX and PIC17CXX families are the industry's only 8-bit microcontrollers using a high-speed RISC architecture. Microchip pioneered the use of RISC architecture to obtain high speed and instruction efficiency. The CMOS PIC16CXX family is in high-volume production, with more than 40 million units shipped, and has achieved more than five thousand design wins worldwide.

The PIC17CXX family offers the world's fastest execution performance of any 8-bit microcontroller family. The PIC17CXX family extends the PIC16/17 microcontroller's high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set, and powerful vectored interrupt handling capabilities. The first member of the family, the PIC17C42, includes a powerful array of intelligent and precise on-chip peripheral features that are ideally suited for many demanding real-time embedded control applications including motor control, process control, security, automotive and medical applications. In addition, the PIC17C42 can function either as a stand-alone microcontroller or can execute instructions from up to 64K words of external program memory. The PIC17C42 features comprehensive timer/counter resources and I/O handling capabilities to address the requirements of complex embedded control applications.

	ROM	EPROM	EEPROM	
PIC17CXX	17CR4X	17C4X		High End 16-Bit Instruction
PIC16CXX	16CR6X 16CR7X	16C6X 16C7X	16C8X	Mid-Range 14-Bit Instruction
PIC16C5X	16CR5X 16CR5XA	16C5X 16C5XA	PICSEE™	Low-End 12-Bit Instruction

CMOS PIC16/17
Microcontroller Families

Microchip Technology Incorporated

Current CMOS PIC16/17 microcontroller product families include advanced features such as sophisticated timers, embedded A/D, extended instruction/data memory, inter-processor communication and ROM, EPROM and EEPROM memories.

Both PIC16CXX and PIC17CXX families are supported by user-friendly development systems including programmers and emulators.

DEVELOPMENT SYSTEMS

The PICMASTER™ is an advanced real-time in-circuit emulator system using the user-friendly Windows™ software environment. The PICMASTER is a Microchip-designed universal emulator for both PIC16CXX and PIC17CXX families. The PRO MASTER™ is an advanced full-featured programmer. PICSTART™ is a low-cost development kit which includes an assembler, simulator and programmer.

SOFTWARE SUPPORT

Both PIC16/17 microcontroller families are supported by assemblers, linker/loaders, libraries and a source-level debugger. The PIC16CXX family is also supported by a software simulator.

A full-featured C Compiler and Fuzzy Logic support are under development to support both families.

Customers can obtain on-line updates on Microchip Development Systems and Support Software via the Bulletin Board System (BBS). See page 7-27 for access information.

SERIAL EEPROMS

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages. Device densities range from 256K bits up to 64K bits. In addition to 5 volt-only operation, Microchip offers Serial EEPROMs that read and write down to 2.5, 2 or 1.8 volts. I²C™, Microwire™ and 4-wire bus interface protocols are standard. Devices come in three standard operating temperature ranges; commercial, industrial and automotive. Small footprint packages include: 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths and 14-lead SOIC. Other key features of the Serial EEPROM product line include: electrostatic discharge (ESD) protection greater than 4K volts and endurance of 100K cycles minimum and one million typical.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide, including consumer, automotive, industrial, computer and communications. To date, more than 80 million units have been produced. Microchip is continuing to develop additional unique Serial EEPROMs.

PARALLEL EEPROMS

The CMOS Parallel EEPROM devices from Microchip are available in 4K, 16K, and 64K densities. The manufacturing process used for these EEPROMs ensures 10,000 to 100,000 write and erase cycles typically. Data retention is more than 10 years. Fast write times are less than 200 μsec. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from -40°C to +125°C. Microchip's expertise in advanced SOIC, TSOP and VSOP surface mount packaging supports our customers' needs in space-sensitive applications.

Typical applications include computer peripherals, engine control, pattern recognition and telecommunications.

Microchip Technology Incorporated

EPROMS

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. High Speed EPROMs have access times as low as 55 nanoseconds. Typical applications include computer peripherals, military, instrumentation, and automotive devices. Microchip's expertise in Surface Mount Packaging on SOIC, TSOP and VSOP packages led to the development of the Surface Mount one-time-programmable (OTP) EPROM market where Microchip is the #1 supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

APPLICATION SPECIFIC STANDARD PRODUCTS (ASSP)

Microchip's new Application-Specific Standard Product (ASSP) Division has the mission of providing value added embedded control solutions through PIC16/17 microcontroller architecture products combined with innovative software, silicon and assembly technology. These products will incorporate technology that will offer a complete solution that is both unique to the customer and standard in manufacture to Microchip. The central theme of this family is to offer a complete solution which reduces or removes the barriers for customers to use Microchip solutions in their products through the use of software embedded in secure OTP- or ROM-based microcontrollers, packaged to provide the highest integration to the customer at the best overall system cost.

The MTA11XXX family is the most accurate and most integrated battery fuel gauge product available today. The family incorporates Microchip/SPAN patented *TrueGauge*[™] technology which digitally integrates battery charge and discharge current to provide an accurate (<3% typical) state of charge indication. The family operates with NiCd and NiMH battery packs from 3 Vdc to 30 Vdc. These products are ideal for portable PC, cellular phone and portable consumer product applications.

Ease of use, low voltage and low cost make the MTA41XXX mouse and trackball MCU firmware solutions ideal for implementing new designs for both PCs and Apple® computers. The products in the MTA41XXX family are 18-lead, low-power CMOS microcontroller ICs combined with application-specific software. By adding a few external components, the user can easily realize a complete mouse or trackball system.

The MTA810XX product line is the first in a series of Multi-Chip Module (MCM) products that integrate PIC16/17 microcontrollers with EEPROM technology. These PICSEE[™] devices are ideally suited for security, keyless entry, data logging and telecommunication applications. The combined product assembly techniques provide the user the highest performance solution in a compact and cost-effective package.

Future ASSP products will include advanced features such as mixed analog and digital capability as well as an ever broadening family of turnkey software solutions for the embedded control market.

OTHER MICROCHIP PRODUCTS

Other Microchip products, such as Liquid Crystal Display Drivers, are mature products with proven track record and a large, repeat customer base.

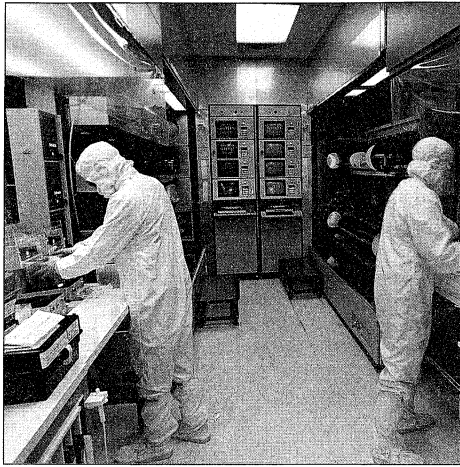
A HISTORY OF INNOVATION

Microchip has a long history of innovation in the semiconductor industry. For more than a quarter century, Microchip and its former parent company have been developers of leading-edge, cost-effective logic and memory products.

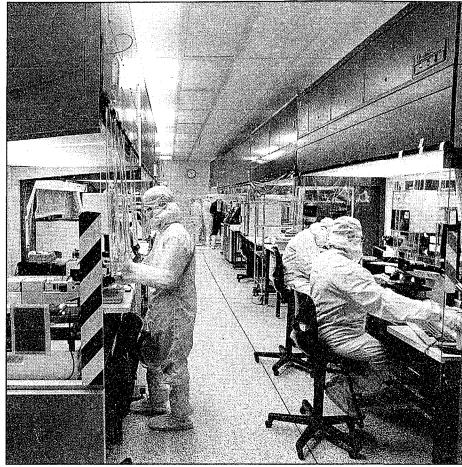
Microchip is credited with a number of firsts: The Metal-Oxide-Silicon (MOS) Integrated Circuit, DRAM, Serial EEPROM, Reduced Instruction Set Computer (RISC) microcontroller product family, UART, CMOS 64K EEPROM, and CMOS single chip DSP are all innovations that were originally developed and introduced by Microchip engineers.

Microchip Technology Incorporated

CHANDLER, AZ FACILITY



Chandler Wafer Fabrication: Diffusion Area



Chandler Wafer Fab: Sub-micron Alignment Area

TAIWAN FACILITY

Microchip's assembly and test operation in Kaohsiung, Taiwan received the prestigious Ishikawa Award for assembly and testing excellence.



The Microchip Kaohsiung plant's excellent track record and continuing efforts to achieve higher levels of quality and technological advancement has resulted in superior yields and fast turnaround.

Microchip Technology Incorporated

FUTURE PRODUCTS AND TECHNOLOGY

New process technology is constantly being developed for microcontroller, ASSP, EEPROM, and high-speed EPROM products. Advanced process technology modules are being developed that will be integrated into present product lines to continue to achieve a range of compatible processes. Current production technology utilizes dimensions down to 0.9 microns.

Microchip's research and development activities, include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools on line. Cycle times for new technology development are continuously reduced by using in-house mask making, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

More advanced technologies are under development, as well as advanced CMOS RISC-based microcontroller, ASSP and CMOS EEPROM and EPROM products. Objective specifications for new products are developed by listening to our customers and by close cooperation with our many customer-partners worldwide.

QUALITY WITHOUT COMPROMISE

Product reliability is designed into Microchip products at the outset. Wide design margins are established to guarantee that every product can be produced easily, error-free and within the tolerances of the manufacturing process.

All quality assurance tests are tighter than customer specifications. Products are tested at least two machine tolerances tighter than those specified by the customer.

Every new product is qualified under accelerated stress testing. Test samples encompass the full range of processed tolerances at each step. Data sheets detailing these processes enable customers to reach accurate decisions based on known quantitative values.

To determine whether a process is within normal manufacturing variation, industry-leading statistical control techniques are put to work at each process step. In-process controls are performed by operators in the wafer fabrication division and immediate corrective action is taken if they deem a process is out of tight control limits. Products are also sampled weekly through a variety of carefully monitored stress and accelerated life tests.

Microchip's documentation control program assures the correct document is always available at the point of use. Active documents are serialized and stamped to eliminate the possibility of performing a job from obsolete or incorrect instructions.

Individuals in all departments continuously analyze the methods employed at their positions and formulate plans to improve performance. In all areas of our business, everyone is expected to make continuous improvement.

A QUALITY AND RELIABILITY ALLIANCE WITH CUSTOMERS

Microchip works together with customers to establish mutual programs to improve the performance of our products in their systems. We go beyond the incoming inspection level and specification by extending our quality and reliability support to the point where the customer ships the system. Microchip's quality programs ensure that our products can be used with such impunity, a customer can implement improvement programs based on Microchip as your leading supplier.

SECTION 1

INTRODUCTION TO EMBEDDED SOLUTIONS

Embedded Control Overview	1- 1
PIC16/17 Microcontroller Families Overview/Road Map	1- 1
PIC16/17 Naming Convention	1- 3
Low-Voltage Operation of PIC16/17 Microcontrollers	1- 4
Serial EEPROM Overview	1- 4
One-Time-Programmable EPROM Overview	1- 4
The Advantages of One-Time-Programmable	1- 4
Application-Specific Standard Product Family	1- 4
Ease of Production Utilizing Quick Turn Programming (QTP) and Serialized Quick Turn Programming (SQTP)	1- 6

Introduction to Embedded Control Solutions

Microchip Technology's mission is to offer leadership semiconductor products for embedded control system applications. To do this we have focused our technology, engineering, manufacturing and marketing resources on two synergistic product lines: Serial EEPROMs and 8-bit PIC16/17 One Time Programmable (OTP) microcontrollers. These product lines provide the solutions to many of the problems facing designers of embedded control systems.

In keeping with one of our guiding values this Embedded Control Handbook has been produced to assist our customers, existing and new, in their efforts to design and produce a state-of-the-art embedded control system, whether it is intended to assist consumer, automotive, computer, communications or industrial embedded system design.

EMBEDDED CONTROL OVERVIEW

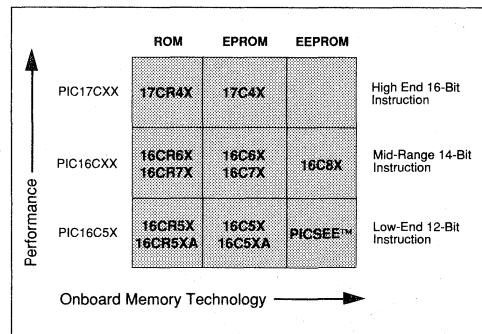
The requirement to manage information processing in today's systems applications necessitate the implementation of embedded control. As opposed to compute intensive environments like workstations and computers of all types, the embedded control system is intended not only to process specific inputs, but also to create the appropriate control response. The use of embedded control allows the system designer to develop a solution for the specific application with the least amount of overhead cost and space. Normally based around a microcontroller, where on-chip program memory and peripheral functions have been combined on a single chip, the needs of the embedded control system are specifically addressed with application specific software and very few, if any, peripheral devices. In the case of a system application where a larger memory may be required, the use of EPROM, EEPROM or Serial EEPROM, designed to communicate directly and efficiently with the microcontroller will provide this extra level of flexibility.

Microchip Technology has established itself as a world leader in providing user-programmable embedded control solutions. The combination of high performance microcontrollers from both the PIC17CXX and PIC16CXX Families, along with industry leading non-volatile memory products provides the basis for this leadership. Continuous innovation and improvement in both the design and manufacturing of embedded control solutions will allow Microchip to maintain and grow this leadership position.

PIC16/17 FAMILIES OVERVIEW AND ROADMAP

The key to providing a successful Embedded Control solution is to insure that the product available has been, and continues to be designed to meet the requirements of the system designer. In order to accomplish this, Microchip has set itself on a course to provide the most useful matrix of products, allowing the system designer to select the microcontroller which will best suit the needs of his particular application. This matrix of products, as shown below, provides the system designer with the ultimate in flexibility and versatility.

FIGURE 1 - MATRIX OF THE PIC16/17 FAMILY OF 8-BIT MICROCONTROLLERS



A strong foundation has been laid with the popular and versatile PIC16C5X Family of High Performance, OTP Microcontrollers. As the base family of PIC16/17s, the PIC16C5X Family provides low cost, small footprint solutions for most embedded control applications. With the low end microcontroller family, Microchip has become a significant factor in the worldwide microcontroller marketplace.

The addition of the PIC17C42, first member of the PIC17CXX Family of high end microcontrollers, demonstrated Microchips continuing commitment to meet the needs of the embedded control systems designer. Most recently, with the introduction of the PIC16C64 (40-pin), Microchip continues its penetration of the mid level 8-Bit microcontroller arena. The PIC16C71 with on-chip A/D converter and the PIC16C84 with EEPROM program and data memory are members of this "mid-range" family.

Introduction

TABLE 1 - THE PIC16/17 FAMILY OF 8-BIT MICROCONTROLLERS

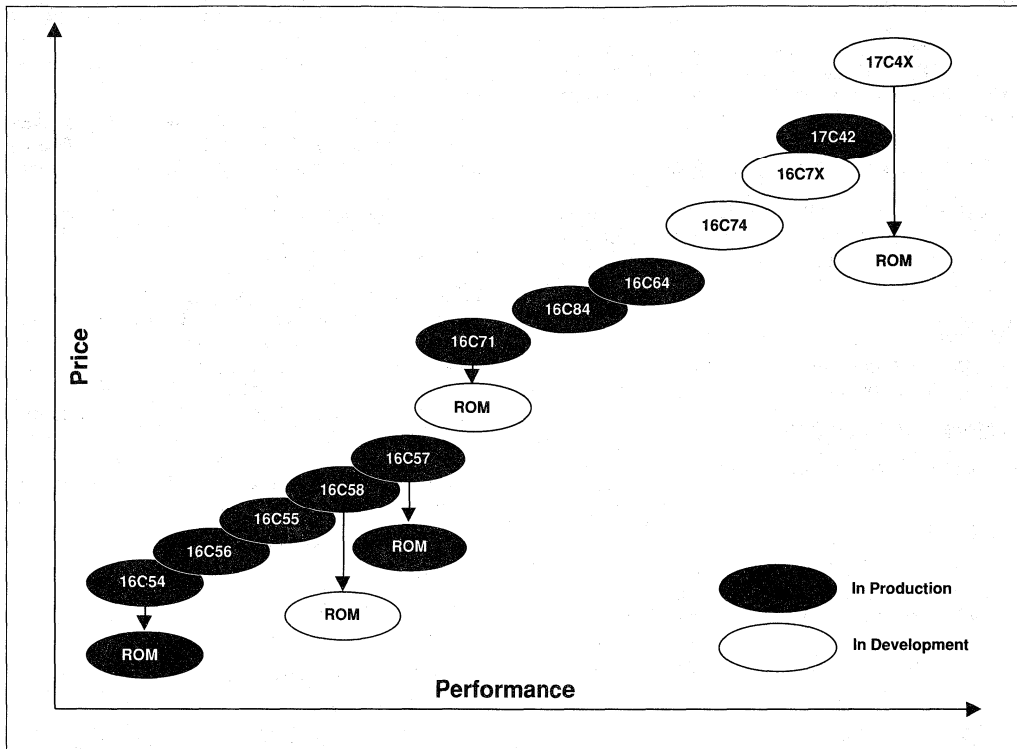
Status as of October 1993	Core Features				Program Memory	Data Memory	MHz	I/O and Peripheral Features				Packages					
	Number of Instructions	Number of Interrupt Sources	Levels of Hardware Stack	EPROM (Words)				RAM Data Memory (Bytes)	Number of I/O	UART	Number of 16-bit Timers		Number of 8-bit Timers	Capture Inputs/PMW/Compare	Analog to Digital Converter		
Family	Products**	Instruction Word Length (bits)	Number of Interrupt Sources	EPROM (Words)	RAM Data Memory (Bytes)	Maximum Oscillator Frequency	Number of I/O	UART	Number of 16-bit Timers	Number of 8-bit Timers	Capture Inputs/PMW/Compare	Analog to Digital Converter	Pin Count				
Low-end	PIC16C54	33	12	2	.5K	—	32	20	12	—	—	1	—	18	DIP, SOIC, SSOP		
	PIC16C54A	33	12	2	.5K	—	32	20	12	—	—	1	—	18	DIP, SOIC, SSOP		
	PIC16CR54	33	12	2	—	.5K	—	32	20	—	—	1	—	18	DIP, SOIC, SSOP		
	PIC16CR54A*	33	12	2	—	.5K	—	32	20	—	—	1	—	18	DIP, SOIC, SSOP		
	PIC16C55	33	12	2	.5K	—	32	20	20	—	—	1	—	28	DIP, SOIC, SSOP		
	PIC16C56	33	12	2	1K	—	32	20	12	—	—	1	—	18	DIP, SOIC, SSOP		
	PIC16C57	33	12	2	2K	—	80	20	20	—	—	1	—	28	DIP, SOIC, SSOP		
	PIC16CR57A	33	12	2	—	2K	—	80	20	—	—	1	—	28	DIP, SOIC, SSOP		
	PIC16C58A	33	12	2	2K	—	80	20	12	—	—	1	—	18	DIP, SOIC, SSOP		
Mid-range	PIC16C64	35	14	8	2K	—	128	20	33	Yes	—	1	2	1	—	DIP, PLCC, PQFP	
	PIC16C71	35	14	4	8	1K	—	36	13	—	—	—	1	—	8-bit 4 ch	DIP, SOIC	
	PIC16C84	35	14	4	8	1K	—	10	13	—	—	—	1	—	—	DIP, SOIC	
High-end	PIC17C42	55	16	11	16	2K	—	256	33	—	—	Yes	3	0	4	—	DIP, PLCC, PQFP

* In Development.

** All products have program memory code protect, watchdog timer, and power-on reset.

† The PIC17C42 can be user-configured for two 16-bit timers and two 8-bit timers.

FIGURE 2 - PIC16/17 MICROPROCESSOR MIGRATION PATH



PIC16/17 NAMING CONVENTION

The PIC16/17 family of microcontrollers covers the low-end, mid-range and high-end 8-bit controller applications.

The **PIC16C5X family** with its 12-bit wide instruction covers the low-end and offers the most cost advantage. In cost, this family competes well against the 4-bit microcontrollers, yet in performance it out-performs other low-end 8-bit controllers.

The **PIC16CXX family** (PIC16C6X, PIC16C7X, PIC16C8X) with its 14-bit wide instruction set and interrupt capability are most suitable for mid-range applications. This family is well-suited for integrating larger program memory, RAM and a wide variety of peripherals.

Finally, the **PIC17CXX family** with its 16-bit wide extended instruction set, external code execution capability and more sophisticated interrupt structure is a power high-end 8-bit controller family. (See Table 2.)

TABLE 2 - PIC16/17 NAMING CONVENTION

16C5X	Low End	PIC16C5X/CR5X	12-bit Architecture	Digital Only (OTP/ROM)
16CXX	Mid-Range	PIC16C6X/CR6X	14-bit Architecture	Digital Only (OTP/ROM)
		PIC16C7X/CR7X		With Analog Functions, e.g. A/D (OTP/ROM)
		PIC16C8X/CR8X		With EEPROM Memory (OTP/ROM)
17CXX	High End	PIC17C4X/CR4X	16-bit Architecture	Digital Only (OTP/ROM)

Introduction

PIC16/17 LOW VOLTAGE OPERATION

A wide operating range (as low as 2.0V for ROM and 2.5V for OTP) has made the PIC16C5X and PIC16CXX families the ideal solution for thousands of applications worldwide.

SERIAL EEPROM OVERVIEW

Serial EEPROMs from Microchip come in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges, and space saving packages.

Densities:

Currently range from 1K to 64K with higher density devices in development.

Bus Interface Protocols:

All major protocols are covered: 2-wire, 3-wire and 4-wire.

Operating Voltages :

In addition to standard 5V devices there are two low voltage families. The "LC" devices operate down to 2.5V, while the "AA" family operates, in both read and write mode, down to a breakthrough 1.8V, making these devices highly suitable for alkaline and NiCad battery powered applications.

Temperature Ranges:

Like all Microchip devices, Serial EEPROMs are offered in Commercial (0°C to 70°C), Industrial (-40°C to 85°C) and Automotive (-40°C to 125°C) operating temperature ranges.

Packages:

The focus here is on small. Most devices are available in 8 pin PDIP or 8 pin SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Endurance is specified at 1M Erase/Write typical and 100K cycles minimum with a data retention specification greater than 40 years. ESD protection is guaranteed up to 4K volts.

OTP EPROM OVERVIEW

Microchip also provides its customers with a number of CMOS OTP EPROMs. Densities offered include: 64K, 128K, 256K and 512K, all in a X8 organization. Our high speed 256K device also comes in a X16 organization. Access times range from a high performance 55ns to a practical 200ns or greater. Low voltage devices, capable

of operating at 3V are available at the 256K and 512K density levels. Surface mounted packages, such as TSOP, PLCC and SOIC as well as the more traditional DIP packages are offered. All EPROMs are available in Commercial, Industrial and Automotive temperature ranges.

A full listing of the Serial EEPROM and EPROM product offerings are shown in Table 3. See the individual Microchip data sheet /data book for detailed information.

THE ADVANTAGES OF ONE-TIME-PROGRAMMABLE

In keeping with Microchips goal of providing the embedded control system designer with the best tools available, Microchip has developed world-leading OTP technology. Microchip offers a wide variety of OTP EPROM products. Similarly, by basing the PIC16/17 Family of products around an EPROM program memory capability, all of the advantages of OTP, both in development and production, have been made economically available to the systems manufacturer. The benefits of OTP technology include:

- Lower costs and shorter lead times
- Reduced time to market
- In-circuit programming capability
- Code protection via security fuse
- Reduction of inventory requirements at system manufacturing site
- Quick correction of bugs detected in manufacturing
- Quick product feature changes in response to customer requests
- Reduces wasted inventory

APPLICATION SPECIFIC STANDARD PRODUCTS

The Application-Specific Standard Products (ASSP) Division complements and strengthens Microchip's leadership position in 8-bit microcontrollers and related specialty memory products for the embedded control market. The ASSP Division employs innovative multi-chip module packaging, applications expertise, firmware and new technology to create integrated, single-chip solutions for specific high-volume embedded control applications such as PC pointing devices and energy management. By offering more complete solutions, Microchip can provide its customers with higher value-added products, with the additional benefits of faster time-to-market and lower design overhead. A full ASSP product listing is shown in Table 4.

TABLE 3 - CMOS SERIAL EEPROMS PRODUCT SELECTION GUIDE

FAMILY	Device	Density Organization	Max. Clock Frequency	Endurance (cycles min.)	Temp. Range	# Pins	Package Types	Operating Voltage
2-Wire (I²C™) Bus Protocol								
STANDARD 2-WIRE FAMILY	24C01A	1K bits (128 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SN, SM	5.0V
	24C02A	2K bits (256 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SN, SM	5.0V
	24C04A	4K bits (512 X 8)	100 kHz	1M 100 K	C, I	8 14	P, J SL	5.0V
	85C72	1K bits (128 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SM	5.0V
	85C82	2K bits (256 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SM	5.0V
	85C92	4K bits (512 X 8)	100 kHz	1M 100 K	C, I E	8 14	P, J, SM	5.0V
LOW-VOLTAGE 2-WIRE FAMILY	24LC01B	1K bits (128 X 8)	400 kHz	1M	C, I	8	P, SN, SM	2.5V-5.5V
	24LC02B	2K bits (256 X 8)	400 kHz	1M	C, I	8	P, SN, SM	2.5V-5.5V
	24LC04B	4K bits (512 X 8)	400 kHz	1M	C, I	8 14	P, SN, SM SL	2.5V-5.5V
	24LC08B	8K bits (1K X 8)	400 kHz	1M	C, I	8 14	P, SN, SM SL	2.5V-5.5V
	24LC16B	16K bits (2K X 8)	400 kHz	1M	C, I	8 14	P, SN SL	2.5V-5.5V
AA LOW-VOLTAGE 2-WIRE FAMILY	24AA01	1K bits (128 X 8)	100 kHz	1M	C	8	P, SN, SM	1.8V-5.5V
	24AA02	2K bits (256 X 8)	100 kHz	1M	C	8	P, SN, SM	1.8V-5.5V
	24AA04	4K bits (512 X 8)	100 kHz	1M	C	8 14	P, SN, SM SL	1.8V-5.5V
	24AA08	8K bits (1K X 8)	100 kHz	1M	C	8 14	P, SN, SM SL	1.8V-5.5V
	24AA16	16K bits (2K X 8)	100 kHz	1M	C	8 14	P, SN SL	1.8V-5.5V
SMART SERIAL 2-WIRE FAMILY	24C32	32K bits (4K X 8)	400 kHz	1M	C, I	8	P, SM, SL	4.5V-5.5V
	24LC32	32K bits (4K X 8)	400 kHz	1M	C, I	8	P, SM, SL	2.5V-6.0V
	24C65	64K bits (8K X 8)	400 kHz	1M	C, I	8	P, SM, SL	4.5V-5.5V
	24LC65	64K bits (8K X 8)	400 kHz	1M	C, I	8	P, SM, SL	2.5V-6.0V
	24AA65	64K bits (8K X 8)	400 kHz	1M	C, I	8	P, SM, SL	1.8V-6.0V
3-Wire (Microwire™)/4-Wire Bus Protocol								
STANDARD 3-WIRE FAMILY	93C06	256 bits (16 X 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V
	93C46	1K bits (64 X 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V
	93C56	2K bits (256 X 8) or (128 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.0V-5.5V
	93C66	4K bits (512 X 8) or (256 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.0V-5.5V
LOW-VOLTAGE 3-WIRE FAMILY	93LC46	1K bits (128 X 8) or (64 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC56	2K bits (256 X 8) or (128 x 16)	2 MHz	1M	C, I	8 14	P, SN, SM SL	2.0V-6.0V
	93LC66	4K bits (512 X 8) or (256 x 16)	2 MHz	1M	C, I	8 14	P, SN, SM SL	2.0V-6.0V
	93LC46B	1K bits (64 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC56B	2K bits (128 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC66B	4K bits (256 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
AA LOW-VOLTAGE 3-WIRE FAMILY	93AA46	1K bits (128 X 8) or (64 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
	93AA56	2K bits (256 X 8) or (128 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
	93AA66	4K bits (512 X 8) or (256 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
4-WIRE	59C11	1K bits (128 X 8) or (64 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V

Product Selection as of October 1993

P = Plastic DIP
 J = Ceramic DIP
 SM = 150mil SOIC
 SN = 207mil SOIC
 SL = SOIC

 C = Commercial (0°C to 70°C)
 I = Industrial (-40°C to 85°C)
 E = Automotive (-40°C to 125°C)

Introduction

TABLE 4 - ASSP PRODUCT SELECTION GUIDE

Device	Interface	Features	Operating Voltage	Temp. Range	Number of Pins	Package Types	
FIRMWARE PRODUCTS							
POINTING DEVICES	MTA41300	Serial, PS/2	2 Button Mouse, Trackball	3.0V-6.25V	C, I	18	PDIP, SOIC, SSOP
	MTA41120	ADB	2 Button Mouse, Trackball	3.0V-6.25V	C, I	18	PDIP, SOIC, SSOP
	MTA41110	PS/2	2 Button Mouse, Trackball	3.0V-6.25V	C, I	18	PDIP, SOIC, SSOP
BATTERY DEVICES	MTA11200	RS232 or 1-wire	Fuel Gauge for NiCD, NiMH, Lead Acid	3.0V-6.25V	C, I	28	PDIP, SOIC, SSOP
PICSEE™ PRODUCTS							
STANDARD CMOS DEVICES	MTA81010-RC	DC to 4 MHz	512 x 12 EPROM, 1K EEPROM	3.0V-6.25V	C, I	28	PDIP, SOIC, JW
	MTA81010-XT	DC to 4 MHz	512 x 12 EPROM, 1K EEPROM	3.0V-6.25V	C, I	28	PDIP, SOIC, JW
	MTA8R1010-RC	DC to 4 MHz	512 x 12 ROM, 1K EEPROM	2.5V-6.25V	C, I	28	PDIP, SOIC
	MTA8R1010-XT	DC to 4 MHz	512 x 12 ROM, 1K EEPROM	2.5V-6.25V	C, I	28	PDIP, SOIC
LOW-POWER DEVICES	MTA81010-LP	DC to 40 KHz	512 x 12 EPROM, 1K EEPROM	2.5V-6.25V	C, I	28	PDIP, SOIC
	MTA8R1010-LP	DC to 40 KHz	512 x 12 ROM, 1K EEPROM	2.0V-6.25V	C, I	28	PDIP, SOIC

EASE OF PRODUCTION UTILIZING QUICK TURN PROGRAMMING (QTP) AND SERIALIZED QUICK TURN PROGRAMMING (SQTP)

Recognizing the needs of high volume manufacturing operations, Microchip has developed two programming methodologies which make the OTP products as easy to use in manufacturing as they are efficient in the system development stage.

Quick-Turn-Programming allows factory programming of OTP product prior to delivery to the system manufacturing operation. PIC16/17, EPROM and serial EEPROM products can be automatically programmed with the users program during the final stages of the test operation at Microchip's assembly and test operation in Kaohsiung, Taiwan. This low cost programming step allows the elimination of programming during system manufacturing and essentially allows the user to treat the PIC16/17 and memory products as custom ROM products. With 1 to 4 week lead times on QTP product, the user no longer needs to plan for the extended ROM masking lead-times and masking charges associated with custom ROM products. This capability, combined

with the off-the-shelf availability of standard OTP product, insures the user of product availability and the ability to reduce his time-to-market once product development has been completed.

Unique to the 8-Bit Microcontroller market is Microchip's ability to enhance the QTP capability with SQTP. Serialized Quick-Turn-Programming allows for the programming of devices with unique, random or serialized identification codes. As each PIC16/17 device is programmed with the customers program code, a portion of the program memory space can be programmed with a unique code, accessible from normal program memory, which will allow the user to provide each device with a unique identification. This capability is ideal for embedded systems applications where the transmission of key codes or identification of the device as a node within a network are essential. Taking advantage of this capability allows the system designer to eliminate the requirement for expensive off-chip code implementation using DIP switches or non-volatile memory components. The SQTP offering, available only from Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.

SECTION 2

PIC16C5X APPLICATION NOTES

Comparison of 8-Bit Microcontrollers - AN520	2- 1
Software Interrupt Techniques - AN514	2- 11
Software Stack Management - AN527	2- 15
Power-Up Considerations - AN522	2- 19
Implementation of an Asynchronous Serial I/O - AN510	2- 23
Using PIC16C5X as a Smart I ² C™ Peripheral - AN541	2- 41
PLD Replacement - AN511	2- 59
Analog to Digital Conversion - AN513	2- 79
Implementing Ohmmeter/Temperature Sensor - AN512	2- 85
Interfacing to AC Power Lines - AN521	2- 91
Implementing Wake-Up on Keystroke - AN528	2- 93
Multiplexing LED Drive and Keypad - AN529	2- 97
Implementing a Simple Serial Mouse Controller - AN519	2- 121
Intelligent Remote Positioner - AN531	2- 133
Programming PIC16C5X Devices on Data I/O Unisite - AN524	2- 149
Programming PIC16C5X Devices on Logical Devices ALLPRO - AN525	2- 153
Utility Math Routines - AN526	2- 155
Implementing an LCD Controller - AN563	2- 215

A Comparison of Low End 8-Bit Microcontrollers

INTRODUCTION

The PIC16C5X Family of microcontrollers from Microchip Technology, Inc. provides significant execution speed and code-compactness improvement over any other 8-bit microcontroller in its price range.

The superior performance of the PIC16C5X microcontrollers can be attributed primarily to its RISC-like architecture. The PIC16C5X employs Harvard architecture, i.e., has separate program memory space (12-bit wide instructions) and data memory space (8-bit wide data). It also uses a two stage pipelining instruction fetch and execution. All instructions are executed in a single cycle (200 ns @ 20 MHz clock) except for program branches which take two cycles, and there are only 33 instructions to remember.

Separation of program and data space allows the instruction word to be optimized to any size (12-bit wide in case of PIC16C5X). This makes it possible, for example, to load an 8-bit immediate value in one cycle. First, because there is no conflict between instruction fetch and data fetch (as opposed to von Neumann architecture) and secondary because the instruction word is wide enough to hold the 8-bit data.

In the following sections we will compare the PIC16C5X @ 20 MHz with:

- SGS-Thomson ST62 @ 8MHz
- Motorola MC68HC05 @ 4.2 MHz
- Intel 8048/8049 @ 11 MHz
- Zilog Z86CXX @ 12 MHz
- National COP800 @ 20 MHz

Several coding examples will be considered. While the comparisons are not entirely scientific, they will, nevertheless, demonstrate to the reader the relative superior performance of the PIC16C5X. The examples chosen here are used frequently in microcontroller applications.

PACKING BCD

This example will take two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.

<p>PIC16C5X</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>SWAPF</td> <td>REGHI,W</td> <td>1</td> <td>1</td> </tr> <tr> <td>IORWF</td> <td>REGLO</td> <td>1</td> <td>1</td> </tr> <tr> <td></td> <td></td> <td><u>2</u></td> <td><u>2</u></td> </tr> </tbody> </table> <p style="text-align: right;">0.4 μs</p>			Byte/Words	Cycles	SWAPF	REGHI,W	1	1	IORWF	REGLO	1	1			<u>2</u>	<u>2</u>	<p>COP800</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>X</td> <td>A,[B+]</td> <td>1</td> <td>2</td> </tr> <tr> <td>SWAP</td> <td>A</td> <td>1</td> <td>1</td> </tr> <tr> <td>OR</td> <td>A,[B]</td> <td>1</td> <td>1</td> </tr> <tr> <td>X</td> <td>A,[B]</td> <td><u>1</u></td> <td><u>1</u></td> </tr> <tr> <td></td> <td></td> <td>4</td> <td>5</td> </tr> </tbody> </table> <p style="text-align: right;">5 μs B is pointing to the higher BCD digit initially. After auto-increment, it points to the lower BCD digit.</p>			Byte/Words	Cycles	X	A,[B+]	1	2	SWAP	A	1	1	OR	A,[B]	1	1	X	A,[B]	<u>1</u>	<u>1</u>			4	5																																
		Byte/Words	Cycles																																																																						
SWAPF	REGHI,W	1	1																																																																						
IORWF	REGLO	1	1																																																																						
		<u>2</u>	<u>2</u>																																																																						
		Byte/Words	Cycles																																																																						
X	A,[B+]	1	2																																																																						
SWAP	A	1	1																																																																						
OR	A,[B]	1	1																																																																						
X	A,[B]	<u>1</u>	<u>1</u>																																																																						
		4	5																																																																						
<p>ST62</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>LD</td> <td>A, REGHI</td> <td>2</td> <td>4</td> </tr> <tr> <td>RLC</td> <td>A</td> <td>1</td> <td>4</td> </tr> <tr> <td>RLC</td> <td>A</td> <td>1</td> <td>4</td> </tr> <tr> <td>RLC</td> <td>A</td> <td>1</td> <td>4</td> </tr> <tr> <td>RLC</td> <td>A</td> <td>1</td> <td>4</td> </tr> <tr> <td>ADD</td> <td>A, REGLO</td> <td>1</td> <td>4</td> </tr> <tr> <td>LD</td> <td>REGLO, A</td> <td><u>2</u></td> <td><u>4</u></td> </tr> <tr> <td></td> <td></td> <td>10</td> <td>28</td> </tr> </tbody> </table> <p style="text-align: right;">45.5 μs REGHI & REGLO are registers addressable by short direct addressing mode.</p>			Byte/Words	Cycles	LD	A, REGHI	2	4	RLC	A	1	4	RLC	A	1	4	RLC	A	1	4	RLC	A	1	4	ADD	A, REGLO	1	4	LD	REGLO, A	<u>2</u>	<u>4</u>			10	28	<p>MC68HC05</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>LDA</td> <td>REGHI</td> <td>2</td> <td>3</td> </tr> <tr> <td>ROLA</td> <td></td> <td>1</td> <td>3</td> </tr> <tr> <td>ROLA</td> <td></td> <td>1</td> <td>3</td> </tr> <tr> <td>ROLA</td> <td></td> <td>1</td> <td>3</td> </tr> <tr> <td>ROLA</td> <td></td> <td>1</td> <td>3</td> </tr> <tr> <td>ADD</td> <td>REGLO</td> <td>2</td> <td>3</td> </tr> <tr> <td>STA</td> <td>REGLO</td> <td><u>2</u></td> <td><u>4</u></td> </tr> <tr> <td></td> <td></td> <td>10</td> <td>22</td> </tr> </tbody> </table> <p style="text-align: right;">10.5 μs</p>			Byte/Words	Cycles	LDA	REGHI	2	3	ROLA		1	3	ROLA		1	3	ROLA		1	3	ROLA		1	3	ADD	REGLO	2	3	STA	REGLO	<u>2</u>	<u>4</u>			10	22
		Byte/Words	Cycles																																																																						
LD	A, REGHI	2	4																																																																						
RLC	A	1	4																																																																						
RLC	A	1	4																																																																						
RLC	A	1	4																																																																						
RLC	A	1	4																																																																						
ADD	A, REGLO	1	4																																																																						
LD	REGLO, A	<u>2</u>	<u>4</u>																																																																						
		10	28																																																																						
		Byte/Words	Cycles																																																																						
LDA	REGHI	2	3																																																																						
ROLA		1	3																																																																						
ROLA		1	3																																																																						
ROLA		1	3																																																																						
ROLA		1	3																																																																						
ADD	REGLO	2	3																																																																						
STA	REGLO	<u>2</u>	<u>4</u>																																																																						
		10	22																																																																						
<p>Z86CXX</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>SWAP</td> <td>REGHI</td> <td>2</td> <td>8</td> </tr> <tr> <td>OR</td> <td>REGHI,REGLO</td> <td><u>2</u></td> <td><u>6</u></td> </tr> <tr> <td></td> <td></td> <td>4</td> <td>14</td> </tr> </tbody> </table> <p style="text-align: right;">5.33 μs REGHI and REGLO are addressable via the working register addressing mode.</p>			Byte/Words	Cycles	SWAP	REGHI	2	8	OR	REGHI,REGLO	<u>2</u>	<u>6</u>			4	14	<p>8048/8049</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>MOV</td> <td>A,Rx</td> <td>1</td> <td>1</td> </tr> <tr> <td>SWAP</td> <td>A</td> <td>1</td> <td>1</td> </tr> <tr> <td>ORL</td> <td>A,Ry</td> <td>1</td> <td>1</td> </tr> <tr> <td>MOV</td> <td>Ry,A</td> <td><u>1</u></td> <td><u>1</u></td> </tr> <tr> <td></td> <td></td> <td>4</td> <td>4</td> </tr> </tbody> </table> <p style="text-align: right;">5.45 μs Register Rx contains higher BCD digit, Ry holds lower BCD digit.</p>			Byte/Words	Cycles	MOV	A,Rx	1	1	SWAP	A	1	1	ORL	A,Ry	1	1	MOV	Ry,A	<u>1</u>	<u>1</u>			4	4																																
		Byte/Words	Cycles																																																																						
SWAP	REGHI	2	8																																																																						
OR	REGHI,REGLO	<u>2</u>	<u>6</u>																																																																						
		4	14																																																																						
		Byte/Words	Cycles																																																																						
MOV	A,Rx	1	1																																																																						
SWAP	A	1	1																																																																						
ORL	A,Ry	1	1																																																																						
MOV	Ry,A	<u>1</u>	<u>1</u>																																																																						
		4	4																																																																						

Microcontroller Comparison

LOOP CONTROL

This example is one of simple loop control where a register containing loop count is decremented, tested for

zero and if not branched back to the beginning of the loop.

<p>PIC16C5X</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>DECFSZ</td> <td>COUNT</td> <td>1</td> <td>1/2</td> </tr> <tr> <td>GOTO</td> <td>BEG_LOOP</td> <td>$\frac{1}{2}$</td> <td>$\frac{2/-}{3/2}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">0.6µs/0.4 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	DECFSZ	COUNT	1	1/2	GOTO	BEG_LOOP	$\frac{1}{2}$	$\frac{2/-}{3/2}$	0.6µs/0.4 µs				<p>COP800</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>DRSZ</td> <td>COUNT</td> <td>1</td> <td>3</td> </tr> <tr> <td>JP</td> <td>BEG_LOOP</td> <td>$\frac{1}{2}$</td> <td>$\frac{3}{6}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">COUNT is Register (RAM F0h-FFh). 6 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	DRSZ	COUNT	1	3	JP	BEG_LOOP	$\frac{1}{2}$	$\frac{3}{6}$	COUNT is Register (RAM F0h-FFh). 6 µs			
		Byte/Words	Cycles																														
DECFSZ	COUNT	1	1/2																														
GOTO	BEG_LOOP	$\frac{1}{2}$	$\frac{2/-}{3/2}$																														
0.6µs/0.4 µs																																	
		Byte/Words	Cycles																														
DRSZ	COUNT	1	3																														
JP	BEG_LOOP	$\frac{1}{2}$	$\frac{3}{6}$																														
COUNT is Register (RAM F0h-FFh). 6 µs																																	
<p>ST62</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>DEC</td> <td>X</td> <td>1</td> <td>4</td> </tr> <tr> <td>JRZ</td> <td>BEG_LOOP</td> <td>$\frac{1}{2}$</td> <td>$\frac{2}{6}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">9.75 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	DEC	X	1	4	JRZ	BEG_LOOP	$\frac{1}{2}$	$\frac{2}{6}$	9.75 µs				<p>MC68HC05</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>DECX</td> <td></td> <td>1</td> <td>3</td> </tr> <tr> <td>BEQ</td> <td></td> <td>$\frac{2}{3}$</td> <td>$\frac{3}{6}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">2.86 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	DECX		1	3	BEQ		$\frac{2}{3}$	$\frac{3}{6}$	2.86 µs			
		Byte/Words	Cycles																														
DEC	X	1	4																														
JRZ	BEG_LOOP	$\frac{1}{2}$	$\frac{2}{6}$																														
9.75 µs																																	
		Byte/Words	Cycles																														
DECX		1	3																														
BEQ		$\frac{2}{3}$	$\frac{3}{6}$																														
2.86 µs																																	
<p>Z86CXX</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>DJNZ</td> <td>COUNT, BEG_LOOP</td> <td>2</td> <td>10/12</td> </tr> <tr> <td colspan="4" style="text-align: center;">1.67 µs/2 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	DJNZ	COUNT, BEG_LOOP	2	10/12	1.67 µs/2 µs				<p>8048/8069</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>DJNZ</td> <td>Rx, BEG_LOOP</td> <td>2</td> <td>2</td> </tr> <tr> <td colspan="4" style="text-align: center;">2.73 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	DJNZ	Rx, BEG_LOOP	2	2	2.73 µs											
		Byte/Words	Cycles																														
DJNZ	COUNT, BEG_LOOP	2	10/12																														
1.67 µs/2 µs																																	
		Byte/Words	Cycles																														
DJNZ	Rx, BEG_LOOP	2	2																														
2.73 µs																																	

Bit Test & Branch

This example tests a single bit in a register or a RAM location and makes a conditional branch. We assume

that MSB is tested and a branch is to be taken if the bit is set.

<p>PIC16C5X</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>BTFSZ</td> <td>REG,7</td> <td>1</td> <td>1/2</td> </tr> <tr> <td>GOTO</td> <td>NEWADD</td> <td>$\frac{1}{2}$</td> <td>$\frac{2/-}{3/2}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">0.6 µs/0.4 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	BTFSZ	REG,7	1	1/2	GOTO	NEWADD	$\frac{1}{2}$	$\frac{2/-}{3/2}$	0.6 µs/0.4 µs				<p>COP800</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>IFBIT</td> <td>7,[B]</td> <td>1</td> <td>1</td> </tr> <tr> <td>JP</td> <td>NEWADD</td> <td>$\frac{1}{2}$</td> <td>$\frac{3}{4}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">B points to the memory location under test. 4 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	IFBIT	7,[B]	1	1	JP	NEWADD	$\frac{1}{2}$	$\frac{3}{4}$	B points to the memory location under test. 4 µs			
		Byte/Words	Cycles																														
BTFSZ	REG,7	1	1/2																														
GOTO	NEWADD	$\frac{1}{2}$	$\frac{2/-}{3/2}$																														
0.6 µs/0.4 µs																																	
		Byte/Words	Cycles																														
IFBIT	7,[B]	1	1																														
JP	NEWADD	$\frac{1}{2}$	$\frac{3}{4}$																														
B points to the memory location under test. 4 µs																																	
<p>ST62</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>JRR</td> <td>7, NEWADD</td> <td>3</td> <td>5</td> </tr> <tr> <td colspan="4" style="text-align: center;">8.125 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	JRR	7, NEWADD	3	5	8.125 µs				<p>MC68HC05</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>BRCLR,7</td> <td>NEWADD</td> <td>3</td> <td>5</td> </tr> <tr> <td colspan="4" style="text-align: center;">2.38 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	BRCLR,7	NEWADD	3	5	2.38 µs											
		Byte/Words	Cycles																														
JRR	7, NEWADD	3	5																														
8.125 µs																																	
		Byte/Words	Cycles																														
BRCLR,7	NEWADD	3	5																														
2.38 µs																																	
<p>Z86CXX</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>BTJRT</td> <td>NEWADD, REG,7</td> <td>3</td> <td>16/18</td> </tr> <tr> <td colspan="4" style="text-align: center;">2.67 µs/3.0 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	BTJRT	NEWADD, REG,7	3	16/18	2.67 µs/3.0 µs				<p>8048/8049</p> <table> <thead> <tr> <th></th> <th></th> <th>Byte/Words</th> <th>Cycles</th> </tr> </thead> <tbody> <tr> <td>MOV</td> <td>A,@Rx</td> <td>1</td> <td>1</td> </tr> <tr> <td>ANL</td> <td>A,#80H</td> <td>2</td> <td>2</td> </tr> <tr> <td>JNZ</td> <td>NEWADD</td> <td>$\frac{2}{5}$</td> <td>$\frac{2}{5}$</td> </tr> <tr> <td colspan="4" style="text-align: center;">Registers R1 is assumed to be pointing to the memory location under test. 6.82 µs</td> </tr> </tbody> </table>			Byte/Words	Cycles	MOV	A,@Rx	1	1	ANL	A,#80H	2	2	JNZ	NEWADD	$\frac{2}{5}$	$\frac{2}{5}$	Registers R1 is assumed to be pointing to the memory location under test. 6.82 µs			
		Byte/Words	Cycles																														
BTJRT	NEWADD, REG,7	3	16/18																														
2.67 µs/3.0 µs																																	
		Byte/Words	Cycles																														
MOV	A,@Rx	1	1																														
ANL	A,#80H	2	2																														
JNZ	NEWADD	$\frac{2}{5}$	$\frac{2}{5}$																														
Registers R1 is assumed to be pointing to the memory location under test. 6.82 µs																																	

Microcontroller Comparison

Shifting Out 8-Bit Data & Clock

We will now consider the task of serially shifting out an 8-bit data and clock. Data and clock outputs are generated under program control by toggling two output pins.

Data is transmitted on the rising edge of the clock. No attempt is made to make the clock output symmetrical in order to make the code efficient. Data out is guaranteed on the falling edge of the clock. These conditions are satisfactory for most applications.

PIC16C5X				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh
XMIT	MOVLW	08H	; Bit Count	1	1	1
	MOVWF	BITCNT	;	1	1	1
XM1	BCF	PORTB,0	; 0 → Data Out Pin	1	1	1
	BCF	PORTB,1	; 0 → Clock Out Pin	1	1	1
	RRF	XDATA	; Rotate Right thru Carry	1	1	1
	BTFSC	STATUS,CARRY	; Test Carry Bit	1	2	1
	BSF	PORTB,0	; 1 → Data Out Pin	1	—	1
	BSF	PORTB,1	; 1 → Clock Out Pin	1	1	1
	DECFSZ	BITCNT	; Decrement Count	1	1	1
			; Skip if Zero			
GOTO	XM1	;	1	2	2	
BCF	PORTC,1	; 0 → Clock	1	1	1	
			11	74	74	

Transmit time is the same for 00h or FFh: 74 Tcyc = 14.8 μs. Note that there was no need to load the data in the Accumulator (W) since the PIC can operate directing on file registers.

COP800				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh
XMIT	LD	A,XDATA	; Load Data in Acc.	2	3	3
	LD	BITCNT #08H	; Load Bit Count	2	3	3
	LD	B,#D0H	; B Points to PORTL	2	3	3
XM1	RBIT	0,[B]	; 0 → Clock	1	1	1
	RBIT	1,[B]	; 0 → Data	1	1	1
	RRCA	;	; Rotate A Right thru Carry	1	1	1
	IFC	;	; 1	1	1	
	SBIT	1,[B]	; 1 → Data	1	—	1
	SBIT	0,[B]	; 0 → Clock	1	1	1
	DRSZ	BITCNT	; Decrement Bit Count	1	3	3
	JP	XM1	; and Go Back if ≠ 0	1	3	3
RBIT	0,[B]	;	1	3	3	
			16	100	108	

Accumulator A is first loaded with the data word. Transmit time is maximum for data = FFh: 105 Tcyc = 105 μs.

ST62				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh
	LDI	A, #08	;	2	4	4
	LD	X, A	; Bit Count	1	4	4
	LD	A, W	; Xmit Data	1	4	4
XM1	RES	0, DRB	; 0 → Clock	2	4	4
	RES	1, DRB	; 0 → Data	2	4	4
	SLA	A	;	2	4	4
	JRNC	XM2	;	1	2	2
XM2	SET	1, DRB	; 1 → Data	2	—	4
	SET	0, DRB	; 1 → CLK	2	4	4
	DEC	X	;	1	4	4
	JRNZ	XM1	;	1	2	2
RES	0, DRB	; 0 → Data	2	4	4	
			19	208	240	

Register W contains the Data word.
Transmit time for FFh = 240 cycles = 390 μs.



Microcontroller Comparison

SHIFTING OUT 8-BIT DATA AND CLOCK (CONT.)

MC68HC05				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh
XMIT	LDA LDX	XDATA #\$08	; Load Xmit Data ; Load Bit Count	2 2	3 2	3 2
XM1	BCLR,	0,PORTB	: 0 → Clock	2	5	5
	BCLR,	1,PORTB	: 0 → Data	2	5	5
	ROLA			1	3	3
	BCC	XM2		2	3	3
XM2	BSET	1,PORTB	: 1 → Data	2	—	5
	BSET	0,PORTB	: 1 → Clock	2	—	5
	DECX			1	5	5
	BNE	XM1		1	3	3
	BCLR	0,PORTB	: 0 → Data	2	3	3
				2	5	5
				<u>20</u>	<u>226</u>	<u>266</u>
Transmit time is maximum for transmitting FFh = 266 cycles = 126.7 μs.						
Z86CXX				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh
XMIT	LD AND	COUNT,#8 P2,#%FC	; Load Bit Count ; 0 → Data, Clock	3 2	10 6	10 6
XM1	RRC	XDATA		2	6	6
	JR	NC,XM2		2	12	10
	OR	P2, #01	: 1 → Data	3	—	10
XM2	OR	P2, #02	: 1 → Clock	3	—	10
	DJNZ	COUNT,XM1		2	10	10
	AND	P2,#%FC	: 0 → Clock, Data	2	12	12
				3	10	10
				<u>21</u>	<u>348</u>	<u>412</u>
Transmit time is maximum for transmitting FFh = 412 cycles = 68.67 μs.						
8048/8049				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh
XMIT	MOV	A,@R0	; R0 Points to Data Word	1	1	1
	MOV	R1,#08H	; Load Bit Count	2	2	2
XM1	ANL	PORT1,#0FCH	: 0 → Data, Clock	2	2	2
	RRC	A	; Rotate Right A thru Carry	1	1	1
	JC	XM2		2	2	2
	ORL	PORT1,#01H	: 1 → Data	2	—	2
XM2	ORL	PORT1,#02H	: 1 → Clock	2	2	2
	DJNZ	R1,XM1	; Decrement Count	2	2	2
				<u>14</u>	<u>75</u>	<u>91</u>
Transmit time is maximum for transmitting FFh = 91 cycles = 124.1 μs.						

Microcontroller Comparison

Software Timer

Microcontrollers quite often need to implement time delays. Debouncing key input, pulse width modulation, and phase angle control are just a few examples. Imple-

menting a 10 ms time delay loop subroutine will be considered in this section.

				Byte/Words	Cycles
PIC16C5X					
DELAY	MOVLW	41H	;10 ms Delay Loop	1	1
	MOVWF	COUNT2	;	1	1
	CLRF	COUNT1	;	1	1
LOOP	INCFSZ	COUNT1	;This Inner Loop will be	1	2/1
	GOTO	LOOP	; Executed 256 Times	1	2
	DECFSZ	COUNT2	;	1	2/1
	GOTO	LOOP	;	1	2
	RET		;	<u>1</u>	
				8	2
Execution time for the routine = $5 + (255 \times 3 + 5) \cdot 65 = 20025 T_{cyc} = 10.011$ ms. The PIC16C5X can implement delay times very precisely (when necessary) because of its fine instruction cycle resolution.					
COP800					
DELAY	LD	COUNT1,#0BH	;10 ms Delay Loop	2	3
	LD	B,#0EH	;	1	1
LOOP	DRSZ	B	;	1	1
	JP	LOOP	;	1	1
	DRSZ	COUNT1	;	1	1
	JP	LOOP	;	1	1
	RET		;	<u>1</u>	5
				8	
Execution time for the routine = $(6N2 + 6) N1 + 9$ cycles. Here $N1 = 0BH$ and $N2 = 0EH$, which gives us: $999 T_{cyc} = 9.99$ ms.					
ST62					
	LDI	A, #FF		2	4
	LD	X, A	; LOOP1 Count	1	4
	LDI	A, #04		2	4
	LD	Y, A	; LOOP2 Count	1	4
LOOP	DEC	X	; 0 CLK	1	4
	JRNZ	LOOP	;	1	2
	DEC	Y	; 0 CLK	1	4
	JRNZ	LOOP	;	<u>1</u>	2
				10	
Execution time for the subroutine = $(6N1 + 6) N2 + 16$ cycles, where $N1 = FFh$, $N2 = 04$ gives us 10.01 ms.					
MC68HC05					
DELAY	LDX	\$2D	;10 ms Delay Loop	2	2
	LDX	\$5C	;	2	2
LOOP	DECA		;	1	3
	BNE		;	2	2
	DECX	LOOP	;	1	3
	BNE		;	2	2
	RTS	LOOP	;	<u>1</u>	6
				11	
Execution time for the subroutine = $(5 \times N1 + 5) N2 + 10$, with $N1 = 2DH$, $N2 = 5CH$, time delay = 10.081 ms.					



Microcontroller Comparison

SOFTWARE TIMER (CONT.)

Z86CXX				Byte/Words	Cycles
DELAY	LD	COUNT1,#%61	;10 ms Delay Loop	2	6
	LD	COUNT2,#%33	;	2	6
LOOP	DJNZ	COUNT1,LOOP	;	2	10/12
	DJNZ	COUNT2,LOOP	;	2	10/12
	RET		;	1	14
				9	
Total execution time = (12N1 + 10) N2, with N1 = 61H, N2 = 33H, time delay = 59976 cycles = 9.979 ms.					
8048/8049				Byte/Words	Cycles
DELAY	MOV	COUNT1,#13H	;10 ms Delay Loop	2	2
LOOP1	MOV	COUNT2,#AFH	;	2	2
LOOP2	DJNZ	COUNT2,LOOP2	;	2	2
	DJNZ	COUNT1,LOOP1	;	2	2
	RET		;	1	2
				9	
Execution time for the subroutine = (2N1 + 4) N2 + 4 cycles. Here N1 = 13H, N2 = AFH, which gives us: 7354 cycles = 10.028 ms.					

Microcontroller Comparison

SUMMARY

Table 1 summarizes code sizes for different microcontrollers. The overall relative code size number is an average of the individual relative code sizes. Given that the PIC16C5X's program word size is 12-bit, whereas all the other microcontrollers have 8-bit program memory, a compaction of 1.5 μ s is expected. Clearly, the PIC16C5X meets this compaction (except for the COP800) and exceeds in most comparisons.

Table 2 summarizes relative execution speed. The overall speed is an average of relative speed numbers. For example, the COP800 will, on an average, exhibit 27% of the code execution speed of a PIC16C5X. In other words, the PIC16C5X will be $1/0.27 = 3.7$ times faster than a COP800 on an average.

TABLE 1 - COMPARISON OF CODE EFFICIENCY*

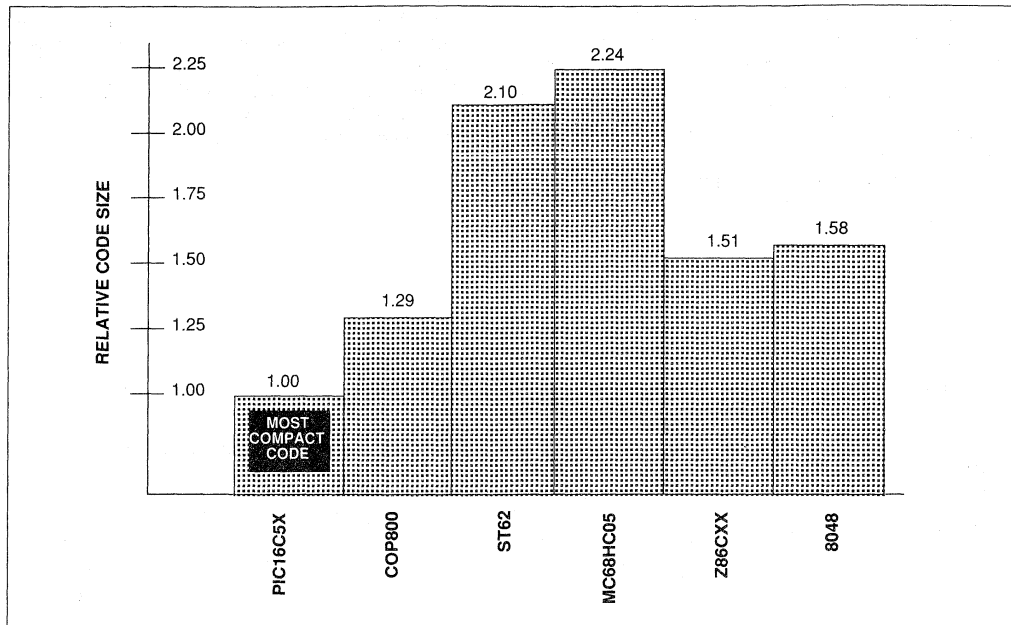
Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Soft- ware Timer	Overall
COP800	4 2.00	2 1.00	2 1.00	16 1.46	8 1.00	1.29
ST62	10 5.00	2 1.00	3 1.50	19 1.73	10 1.25	2.10
MC68HC05	10 5.00	3 1.50	3 1.50	20 1.82	11 1.38	2.24
Z86CXX	4 2.00	2 1.00	3 1.50	21 1.91	9 1.125	1.51
8048/8049	4 2.00	2 1.00	5 2.51	14 1.28	9 1.13	1.58
PIC16C5X @ 8 MHz	2	2	2	11	8	1.00

* In each box, the top number is the number of program memory locations required to code the application. The bottom number is relative code size compared to the PIC16C5X:

$$\frac{\text{\# program memory locations for other microcontroller}}{\text{\# program memory locations for the PIC16C5X}}$$

Microcontroller Comparison

FIGURE 1 - CODE SIZE COMPARISON



Microcontroller Comparison

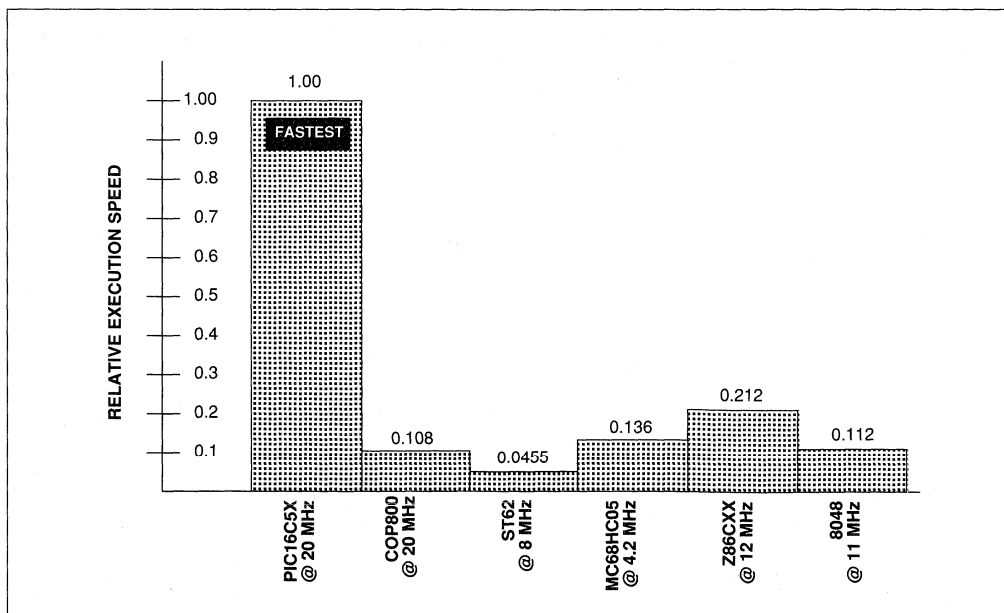
TABLE 2 - COMPARISON OF EXECUTION SPEED*

Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800 @ 20 MHz	5 μ s 0.08	6 μ s 0.0832	4 μ s 0.1252	105 μ s 0.1408	-	0.108
ST62 @ 8 MHz	45.5 μ s 0.0088	9.75 μ s 0.0615	8.125 μ s 0.0738	390 μ s 0.0329	-	0.0455
MC68HC05 @ 4.2 MHz	10.05 μ s 0.038	2.86 μ s 0.1748	2.38 μ s 0.21	126.7 μ s 0.1168	-	0.136
Z86CXX @ 12 MHz	2.33 μ s 0.172	1.835 μ s 0.272	2.835 μ s 0.176	68.67 μ s 0.224	-	0.212
8048/8049 @ 11 MHz	5.45 μ s 0.0732	2.73 μ s 0.1824	6.82 μ s 0.0732	124.1 μ s 0.1196	-	0.112
PIC16C5X @ 20 MHz	0.4 μ s	0.6/0.4 μ s	0.6/0.4 μ s	14.8 μ s	-	1.00

* In each box, the top number is the time required to execute the example code, while the bottom number is a measure of relative performance compared to the PIC16C5X:

$$\frac{\text{time required to execute code by the PIC16C5X}}{\text{time required to execute code by other microcontroller}}$$

FIGURE 2 - EXECUTION SPEED COMPARISON



Microcontroller Comparison

NOTES:

Software Interrupt Techniques

INTRODUCTION

This application note describes a unique method for implementing interrupts in software on the PIC16C5X series of microcontrollers. The method takes advantage of the PIC16C5X's architecture which allows changing the program counter under software control. Up to 8 interrupt lines are possible, but the practical limit for simple code generation is 6 interrupts, or 64 possible input conditions. The interrupt detection time is under software control and standard I/O pins are used as the interrupt lines.

THEORY OF OPERATION

SOFTWARE POLLING OF I/O LINES REPLACES HARDWARE INTERRUPT

The interrupt conditions are determined by detecting changes on the I/O lines that have been selected to be the interrupt lines. These changes are used to create a jump table that allows a different program response to each interrupt condition. The interrupt response time is under software control and can be as short as ten to twenty microseconds, depending on main program and interrupt subroutine program length.

CREATING THE INTERRUPT SUBROUTINE JUMP TABLE

Each I/O condition may have its own unique subroutine to respond to changes on the interrupt lines. Direct access to these routines is achieved by using the PIC16C5X's ability to change the program counter under software control. Here is an example of how two I/O lines may be polled:

```

MOVF  CONDTN,W  ;LOAD I/O CONDITION INTO W
                ;REGISTER

ANDLW  3        ;MASK OFF TOP 6 BITS

ADDWF  2,1      ;ADD INPUT TO PROGRAM COUNTER
                ;TO CREATE JUMP TABLE

GOTO   MAIN     ;FOR NO CHANGE GO TO MAIN
                ;PROGRAM

GOTO   INT1     ;FOR CHANGE IN BIT 0 GOTO INT1
GOTO   INT2     ;FOR CHANGE IN BIT 1 GOTO INT2
GOTO   INT3     ;FOR BOTH CHANGE GOTO INT3

```

The changes to the I/O lines have been used to create a two bit number that is added to the program counter. The GOTO that is executed depends on the new program counter address.

CREATING CONSTANT TIME POLLING

In most applications requiring interrupts, it is important to poll the interrupt lines at fixed time intervals, usually only a few microseconds in length. Two techniques may be used on the PIC16C5X to achieve this. They are dividing the main program into multiple sections and implementing an elapsed time counter (see flow chart). Both of these techniques use the same program jump table concept that was described above. First, the main program is divided into several sections based on the desired I/O polling time. When MAIN is called a branch register is added to the program counter. This determines which section of MAIN code should be executed next. At the end of execution the branch register is decremented so the next section of code will be executed after the next polling. If the branch register is zero then the number of sections of main code is added into it to start the main program over again.

An elapsed time counter can be implemented using the RTCC counter. At the beginning of I/O polling the RTCC register is cleared. It then starts counting the instruction cycles. Then after the main program subsection has been executed, the RTCC register is subtracted from the desired polling time. This determines how many instructions need to be executed before the next polling. A jump table is then created to execute these instructions before the next polling. An example is shown below. This example assumes from zero to 15 additional instruction cycles are needed. Actual numbers need to be computed for each individual application.

```

MOVLW  POLL    ;POLL=DESIRED POLL CYCLES - 15

SUBWF  RTCC,W  ;DETERMINE HOW MUCH TIME TO WAIT

ADDWF  2,1     ;ADD WAIT TIME TO PROGRAM
                COUNTER

NOP                    ;15 ADDITIONAL INSTRUCTION CYCLES
:
:                    ;TOTAL OF 15 NOP'S

NOP                    ;1 ADDITIONAL INSTRUCTION CYCLES

GOTO   START    ;0 ADDITIONAL INSTRUCTION CYCLES

```

Software Interrupt Techniques

For example, if the desired instruction time is 50 cycles and the subsection we just executed has a consumed a total of 40 instruction cycles (including all overhead cycles) the value of

$$RTCC(40) - POLL(50-15(35)) = 5$$

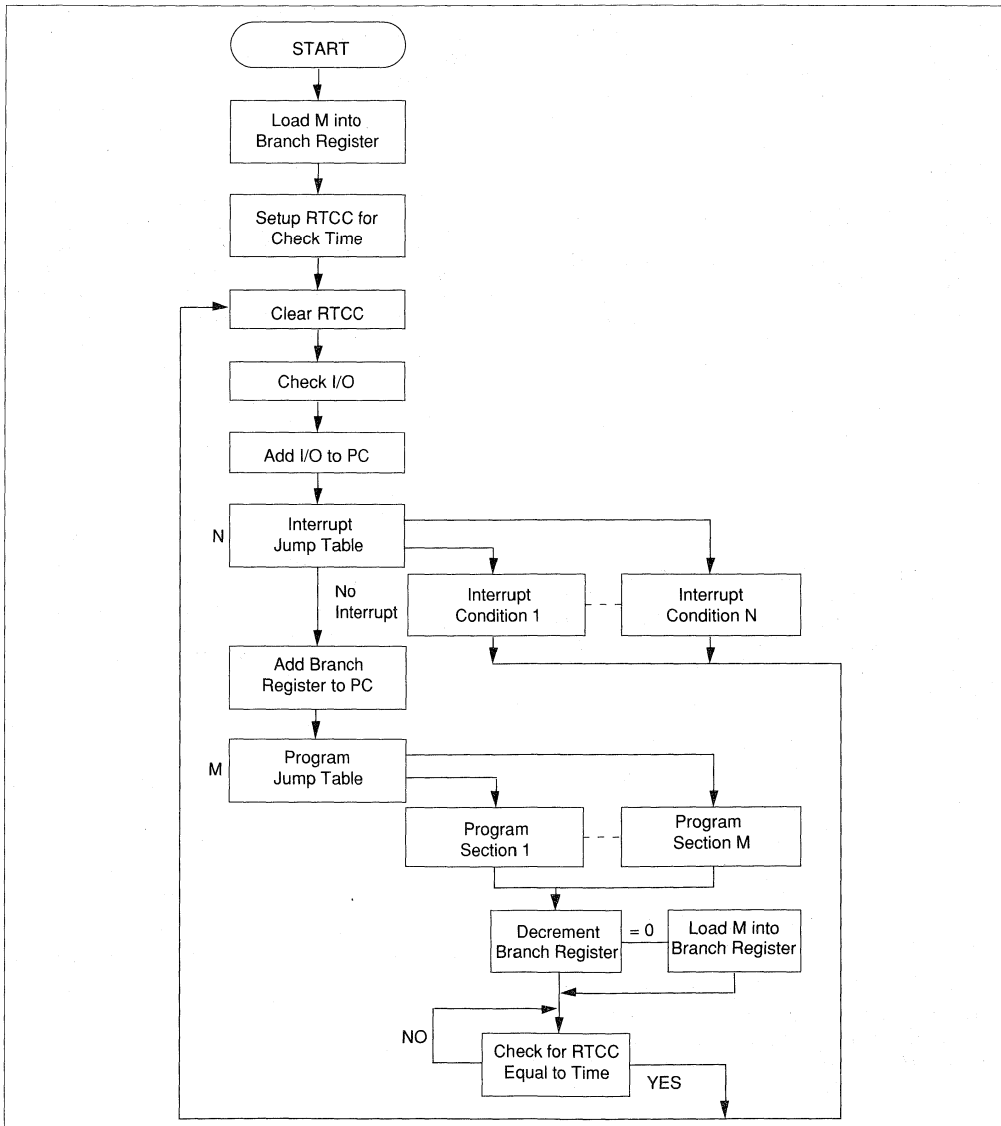
will be added to the program counter. The program will then jump to the sixth NOP. That NOP plus the 9 following it will be executed for a total of ten more instruction cycles. Note that the final GOTO has two

instruction cycles and these must be included in the program overhead.

Example

The following example (see flow chart and code) is the core program for the software interrupt technique described above. This program assumes four interrupt conditions, four main program sections and an eight additional elapsed time instructions.

FIGURE 1 - SOFTWARE INTERRUPT FLOW CHART



Software Interrupt Techniques

APPENDIX A:

MPASM B0.54

PAGE 1

```

LIST      P=16C54

;SOFTWARE INTERRUPT APPLICATIONS
;BRANCH IS MAIN PROGRAM REGISTER

0008      BRANCH EQU      8
0009      CNDTN EQU      9
000A      IO EQU         0A
000B      TEMP EQU       0B

0000 0069      SETUP CLRF  CNDTN
0001 0C04      MOVLW  4
0002 0028      MOVWF  BRANCH ;FOUR MAIN PROGRAM SECTIONS
0003 0C08      MOVLW  8
0004 0002      OPTION   ;SET RTCC TO ONE COUNT PER INSTRUCTION CYCLE

0005 0061      START  CLRF  1 ;CLEAR RTCC REGISTER
0006 0206      MOVF   6,W ;READ I/O
0007 002A      MOVWF  IO
0008 0109      IORWF  CNDTN,W ;THIS SECTION OF CODE CALCULATES THE
0009 002B      MOVWF  TEMP ;JUMP TABLE. ANY INPUT THAT CHANGES FROM
000A 0209      MOVF   CNDTN,W ;A ZERO TO A ONE IS CONSIDERED AN INTERRUPT.
000B 00AB      SUBWF  TEMP,1 ;THE EQUATION IS:
000C 020A      MOVF   IO,W ; (IO + CNDTN) - CNDTN = INTERRUPT
000D 0029      MOVWF  CNDTN ;WHERE IO IS CURRENT INPUT AND
000E 020B      MOVF   TEMP,W ;CNDTN IS PREVIOUS INPUT.
000F 0E03      ANDLW  3 ;MASK OFF TOP 6 BITS
0010 01E2      ADDWF  2,1 ;ADD INPUT TO PC TO CREATE JUMP TABLE
0011 0A1B      GOTO  MAIN ;FOR INPUT=00
0012 0A15      GOTO  INT1 ;FOR INPUT=01
0013 0A17      GOTO  INT2 ;FOR INPUT=10
0014 0A19      GOTO  INT3 ;FOR INPUT=11

0015 0000      INT1  NOP ;INTERRUPT LINE 1 CODE
0016 0A05      GOTO  START
0017 0000      INT2  NOP ;INTERRUPT LINE 2 CODE
0018 0A05      GOTO  START
0019 0000      INT3  NOP ;INTERRUPT LINES 1 AND 2 CODE
001A 0A05      GOTO  START

001B 0208      MAIN  MOVF  BRANCH,W
001C 01E2      ADDWF  2,1 ;ADD BRANCH TO PC TO CREATE JUMP TABLE
001D 0000      NOP
001E 0A28      GOTO  MAIN4 ;JUMP TABLE, LAST FIRST ON DECREMENT TABLE
001F 0A26      GOTO  MAIN3
0020 0A24      GOTO  MAIN2
0021 0A22      GOTO  MAIN1

0022 0000      MAIN1 NOP ;MAIN PROGRAM CODE BANK ONE
0023 0A2A      GOTO  BRNCHK
0024 0000      MAIN2 NOP ;MAIN PROGRAM CODE SECTION TWO
0025 0A2A      GOTO  BRNCHK
0026 0000      MAIN3 NOP ;MAIN PROGRAM CODE SECTION THREE
0027 0A2A      GOTO  BRNCHK
0028 0000      MAIN4 NOP ;MAIN PROGRAM CODE SECTION FOUR
0029 0A2A      GOTO  BRNCHK

002A 02E8      BRNCHK DECFSZ BRANCH,1 ;DECREMENT BRANCH REGISTER AND CHECK FOR ZERO
002B 0A2E      GOTO  TIMCHK
002C 0C04      MOVLW  4
002D 0028      MOVWF  BRANCH ;RELOAD BRANCH WITH 4 AT END OF MAIN

002E 0C29      TIMCHK MOVLW D'41' ;CHECK TO SEE IF RTCC HAS REACHED 50 (50-7)

```

Software Interrupt Techniques

```
002F 0081          SUBWF 1,W      ;DETERMINE WAIT TIME
0030 01E2          ADDWF 2,1     ;ADD WAIT TIME TO PC
0031 0000          NOP
0032 0000          NOP
0033 0000          NOP
0034 0000          NOP
0035 0000          NOP
0036 0000          NOP
0037 0000          NOP
0038 0A05          GOTO  START
                   END
```

```
Errors   :    0
Warnings :    0
```



Software Stack Management

INTRODUCTION

The PIC16C5X has a stack which is only 2 deep, as a result of which only two nested calls can be made (i.e. only one call within a call routine). If more than two levels of subroutine nesting is required, the following Ap Note can be used to implement a stack manager to handle the flow of the calls.

Note: Since the amount of RAM on the PIC16CXX is limited, it would be prudent to determine the maximum number of nested calls which have to be made in a program and define the stack length appropriately.

IMPLEMENTATION

This Application Brief implements a 5 deep stack, so 5 nested calls can be made without overflowing the stack. NCALL is defined as a MACRO which will be used instead of the mnemonic CALL, when a subroutine call is made. The NCALL routine, "pushes" the return PC value on the "stack" and then executes the called subroutine. At the end of the subroutine, instead of using the RETLW k instruction, a GOTO RETURN is executed, where RETURN is a routine which "pops" the return PC value from the "stack" and resumes the normal flow of the program.

Notes:

Since Software Stack Management utilizes the FSR register, and indirect addressing, the user should restore the "original" values to the FSR register if it is utilized elsewhere in the program.

*Author: Stanley D'Souza
Logic Products Division*

Software Stack Management

MPASM B0.54

PAGE 1

```

;*****
;
;   sm.asm:
;   Routine, demonstrating how to implement a stack
;   manager capable of handling more than 2
;   subsequent subroutine calls.
;   Note: Since this is a demo, NOP has been used
;   where normally the body of the subroutine would
;   reside.
;*****
;
;       LIST    P=16C54
0002      PC      EQU    2
0004      FSR     EQU    4
;
;       ORG     8
0008 0005  STACK  RES    5      ;define stack size = 5.
;
;       ORG     01FF
01FF 0A07  GOTO   START
;
;       ORG     0
;
;   TNIT  MOVLW  STACK  ;load "stack" as indirect pointer
0000 0C08      MOVWF  FSR  ;      /
0001 0024      GOTO   START ;      /
0002 0A07
;
;*****
;define NCALL as a MACRO used instead of the
;mnemonic CALL.
;
NCALL  MACRO  LABEL
        MOVF  PC,W      ;save PC on "stack"
        MOVWF 0         ;      /
        INCF  FSR       ;Inc. "stack" pointer.
        GOTO  LABEL     ;jump to routine
        ENDM
;
;return from subroutine NCALL
;
0003 00E4  RET     DECF   FSR      ;point to last "stack" location
0004 0C03      MOVLW  3         ;add 3 and output value from FSR
0005 01C0      ADDWF  0,W       ;      /
0006 0022      MOVWF  PC        ;load in PC as next executable
;                                     instruction
;
;*****
;
;   START  NOP
;          NCALL  TOM
0007 0000
;          MOVF  PC,W      ;save PC on "stack"
0008 0202      MOVWF  0         ;      /
0009 0020      INCF  FSR       ;Inc. "stack" pointer.
000A 02A4      GOTO  TOM        ;jump to routine
000B 0A0F
;
;          NOP            ;body of main routine
000C 0000
;          NOP            ;      /
000D 0000
000E 0003  SLEEP
;
;   TOM    NOP
;          NCALL  DICK
000F 0000
;          MOVF  PC,W      ;save PC on "stack"
0010 0202      MOVWF  0         ;      /
0011 0020      INCF  FSR       ;Inc. "stack" pointer.
0012 02A4      GOTO  DICK        ;jump to routine
0013 0A16

```

Software Stack Management

```
0014 0000      NOP      ;body of routine TOM
0015 0A03      GOTO     RET

;
0016 0000      DICK     NOP
                   NCALL   HARRY
0017 0202      MOVF    PC,W   ;save PC on "stack"
0018 0020      MOVWF   0      ; /
0019 02A4      INCF    FSR   ;Inc. "stack" pointer.
001A 0A1D      GOTO     HARRY ;jump to routine

001B 0000      NOP      ;body of routine DICK
001C 0A03      GOTO     RET

;
001D 0000      HARRY    NOP      ;body of routine HARRY
001E 0000      NOP      ; /
001F 0A03      GOTO     RET

;
;
END
```

```
Errors : 0
Warnings : 0
```

Software Stack Management

NOTES:



Power-Up Considerations

INTRODUCTION

When powering up all microcontrollers it is necessary for the power supply voltage to traverse voltage ranges where the device is not guaranteed to operate before the power supply voltage reaches its final state. Since some circuits on the device (logic) will start operating at voltage levels lower than other circuits on the chip (memory), the device may power-up in an unknown state. To guarantee that the device starts up in a known state, it is necessary that it contain a power-up reset circuit. PIC16C5X microcontrollers are equipped with on-chip power-on reset circuitry, which eliminates the need for external reset logic. This circuit will function in most power-up situations where V_{CC} rise time is fast enough (50 ms or less). This application note describes the typical power-up sequence for PIC16C5X microcontrollers. Methods of assuring reset on power-up and after a brownout are discussed and simple, low cost external solutions are discussed for power-up situations where the PIC16C5X's internal circuitry cannot provide the reset.

POWER-UP SEQUENCE

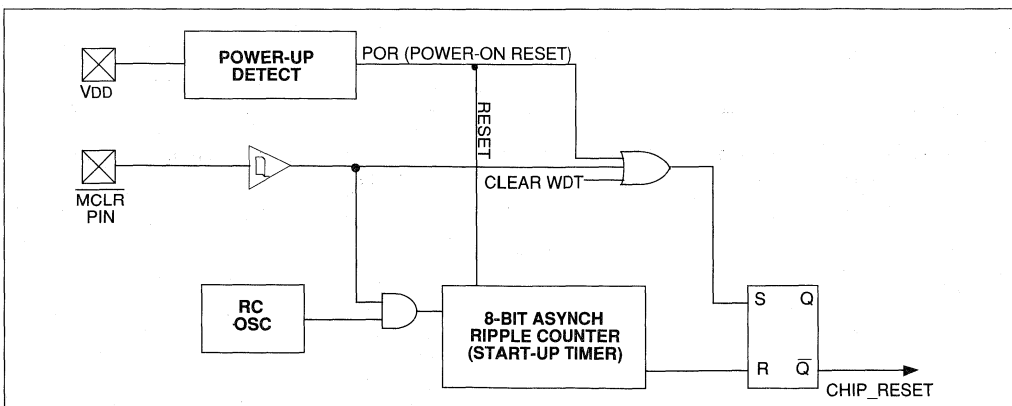
The PIC16C5X incorporates complex power-on reset (POR) circuitry on-chip which provides solid, reliable internal chip reset for most power-up situations. To use this feature, the user merely needs to tie MCLR to V_{DD} . A simplified block diagram of the on-chip reset circuitry is shown in Figure 1. On power-up, the reset latch and

the start-up timer are reset to appropriate states by the power-on reset (POR). The start-up timer will begin counting once it detects MCLR to be high (i.e., external chip reset goes inactive). After the time-out period, which is typically 18 ms long, the timer will reset the reset latch and thus end the on-chip reset signal.

Figures 2 and 3 are two power-up situations with relative fast rise time on V_{DD} . In Figure 1, V_{DD} is stable when MCLR is brought high (i.e., reset pulse is being provided by external source). The chip actually comes out reset about t_{OST} ms after that, where t_{OST} = oscillator start-up timer. (The timer is called oscillator start-up timer because the time-out was incorporated primarily to allow the crystal oscillator to stabilize on power-up.) In Figure 3, the MCLR and V_{DD} are tied together and clearly the on-chip reset mechanism is being utilized. The V_{DD} is stable before the start-up timer expires and there is no problem with proper reset.

Figure 4, where V_{DD} rise time is much greater than t_{OST} (typically 18 ms) clearly is the potentially problematic situation. The POR (power-on reset) pulse comes when V_{DD} is about 1.5V. Most CMOS logic, including the start-up timer starts functioning between 1.5V to 2.0V. When the start-up timer starts times out, the chip reset is ended and the chip attempts to execute. If by this time the V_{DD} has reached $V_{DD\ MIN}$ value, then all circuits are guaranteed to function correctly and power-up reset is successful. If, however, the V_{DD} slope was too slow and had not reached $V_{DD\ MIN}$, then the chip may or may not function properly.

FIGURE 1 - PIC16C5X INTERNAL RESET CIRCUIT



Power-Up Considerations

FIGURE 2 - EXTERNAL RESET PULSE

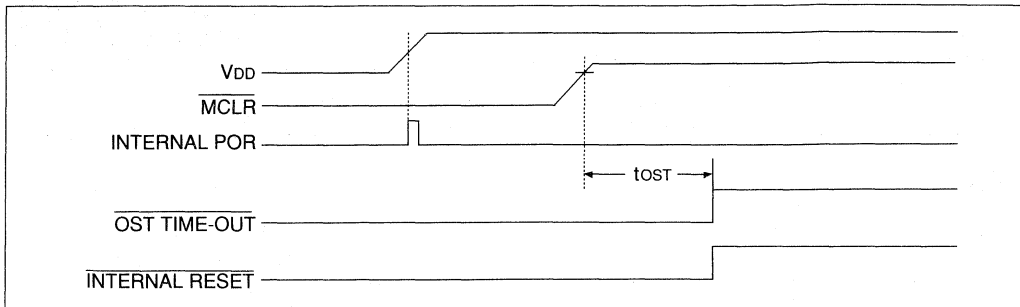


FIGURE 3 - INTERNAL RESET (V_{DD} AND MCLR TIED TOGETHER)

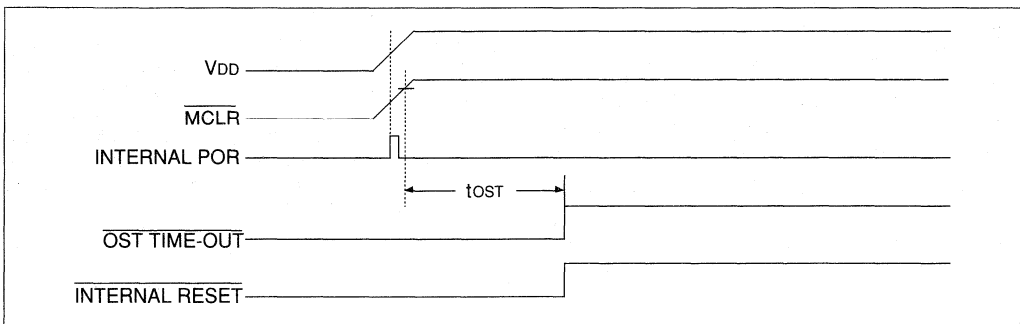
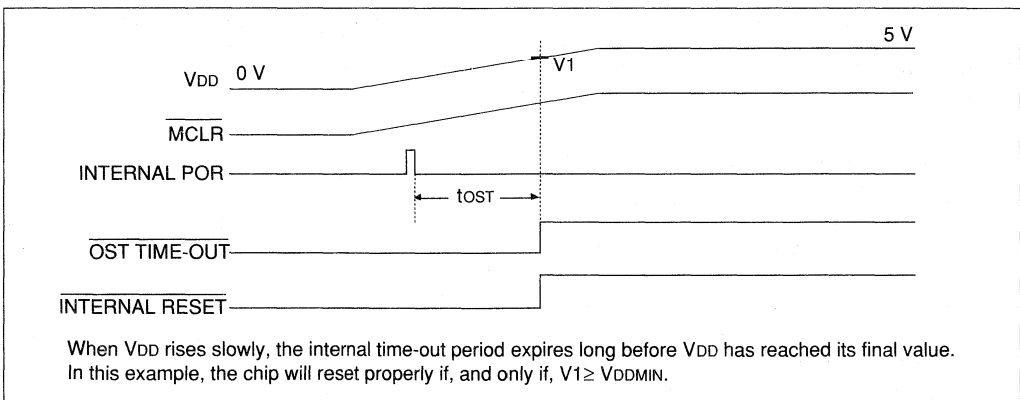


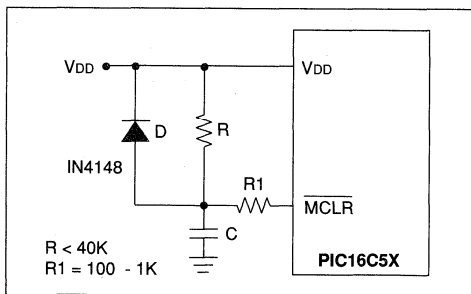
FIGURE 4 - INTERNAL RESET (V_{DD} AND MCLR TIED TOGETHER): SLOW V_{DD} RISE TIME



EXTERNAL POWER-ON RESET CIRCUIT

To use power supplies with slow rise times it is necessary to use an external power-on reset circuit such as the one shown in Figure 5. This circuit uses an external RC to generate the reset pulse. The time constant of the RC should be long enough to guarantee that the reset pulse is still present until V_{DD} has reached V_{DD} min. R should be 40K or less to guarantee that the MCLR will pull to within 0.2 volts of V_{DD}. (since the leakage spec on MCLR is ±5 μA, a resistor larger than 40K may cause input high voltage on this pin to be less than V_{DD} - 0.2V, the required spec). The diode D is used to rapidly discharge the capacitor on power-down. This is very important as a power-up reset pulse is needed after a short power-down (less than the time constant of RC) or after a power spike. The resistor R1 protects against high current flowing into MCLR pin from fully charged capacitor C in the event MCLR pin breakdown is induced through ESD or EOS. The circuit, however, does not protect against brown-out situations where the power does not drop to zero, but merely dips below V_{DD} MIN. In such a situation, voltage at the MCLR pin will not go low enough (i.e., below V_{IL}) to guarantee a reset pulse. The following section presents an example circuit to protect against such brown-outs.

FIGURE 5 - EXTERNAL POWER-ON RESET CIRCUIT



BROWNOUT PROTECTION

In many applications it is necessary to guarantee a reset pulse whenever V_{DD} is less than V_{DD} min. This can be accomplished using a brownout protection circuit such as the one shown in Figure 6. This is a simple circuit that causes a reset pulse whenever V_{DD} drops below the zener diode voltage plus the V_{be} of Q1. A 3.3 volt zener will produce a reset pulse whenever V_{DD} drops below about 4 volts. This circuit has a typical accuracy of about ±100 mV. A less expensive, albeit less precise, brown-out circuit is shown in Figure 7. Transistor Q1 turns off when V_{be} = V_{DD}·R1/(R1+R2) falls below 0.7 V allowing R3 to pull down MCLR input.

FIGURE 6 - BROWNOUT PROTECTION CIRCUIT

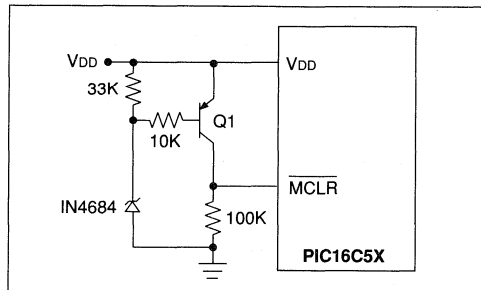
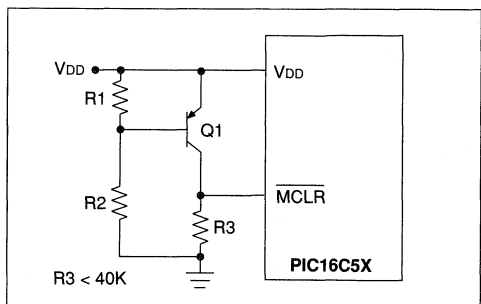


FIGURE 7 - BROWNOUT PROTECTION CIRCUIT



Author: Sumit Mitra
 Logic Products Division

Power-Up Considerations

NOTES:



Implementation of an Asynchronous Serial I/O

INTRODUCTION

The PIC16C5X series from Microchip Technology, Inc., are 8 bit, high speed EPROM based microcontrollers. This application note describes the implementation of an Asynchronous serial I/O using Microchip's PIC16C5X series of high speed 8 bit microcontrollers. These EPROM based microcontrollers can operate at very high speeds with a minimum of 200 ns cycle time @ 20 MHz input clock. Many microcontroller applications require chip to chip serial data communications. Since the PIC16C5X series have no on chip serial ports, serial communication has to be performed in software. For many cost-sensitive high volume applications, implementation of a serial I/O through software provides a more cost effective solution than dedicated logic. This application note provides code for PIC16C5X to simulate a serial port using 2 I/O Pins (one as input for reception and the other as output for transmission).

IMPLEMENTATION

Two programs are provided in this application note. One program simulates a full duplex RS-232 communication and the other provides implementation of half duplex communication. Using Half-Duplex, rates up to 19200 baud can be implemented using an 8 MHz input clock. In case of Full-Duplex, the software can handle up to 9600 baud at 8 MHz and 19200 baud at 20 MHz, one or two stop bits, 8 or 7 data bits, No Parity and can transmit

or receive with either LSB first (normal mode) or MSB first (CODEC like mode). It should be noted that the higher the input clock the better the resolution. These options should be set up during assembly time and not during run time. The user simply has to change the header file for the required communication options. The software does not provide any handshaking protocols. With minor modifications, the user may incorporate software handshaking using XON/XOFF. To implement hardware handshaking, an additional 2 digital I/O Pins may be used as RTS (ready to send) and CTS (clear to send) lines.

Figure 1 shows a flow chart for serial transmission and Figure 2 shows a flow chart for reception. The flowcharts show cases for transmission/reception with LSB first and 8 data bits. For reception, the data receive pin, DR, is polled approximately every $B/2$ seconds ($52 \mu\text{s}$ in case of 9600 baud) to detect the start bit, where B is the time duration of one bit ($B = 1/\text{Baud}$). If a start bit is found, then the first data bit is checked for after $1.25B$ seconds. From then on, the other data bits are checked every B seconds ($104 \mu\text{s}$ in case of 9600 baud).

In the case of transmission, first a start bit is sent by setting the transmit data pin, DX to zero for B seconds, and from then on the DX pin is set/cleared corresponding to the data bit every B seconds. Assembly language code corresponding to the following flowcharts are given in Figures 3 and 4.

Asynchronous Serial I/O

FIGURE 1 - TRANSMISSION FLOW CHART

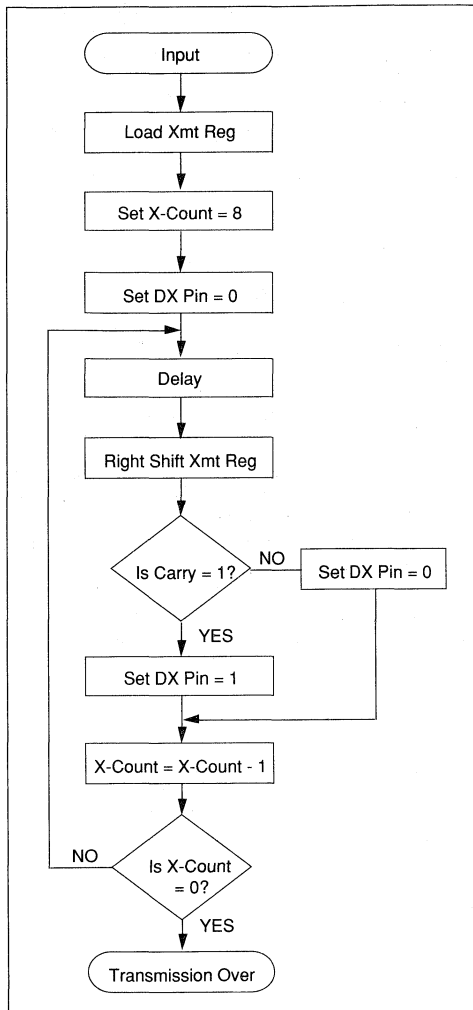


FIGURE 2 - RECEPTION FLOW CHART

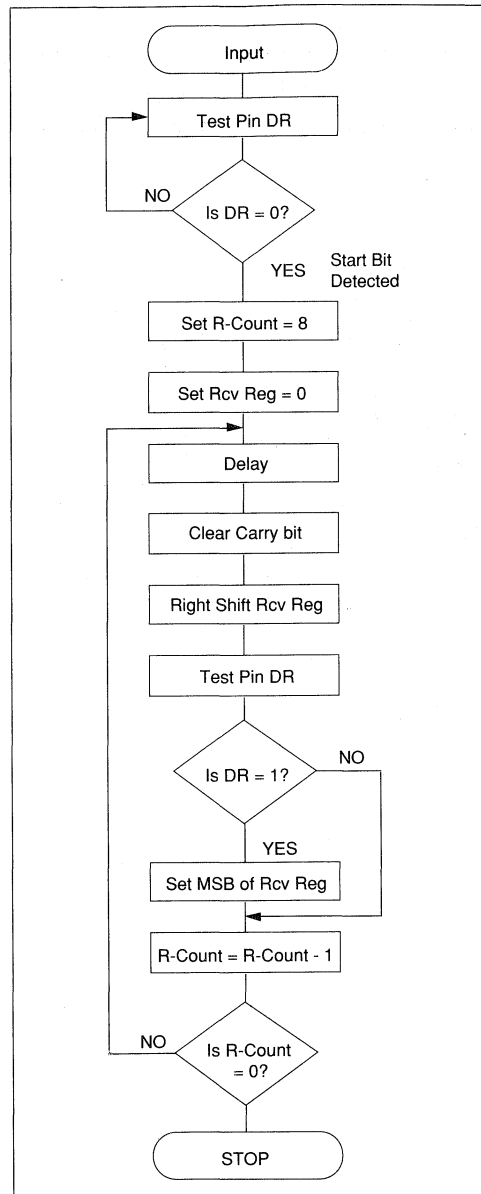


FIGURE 3 - TRANSMIT ASSEMBLY CODE (CORRESPONDING TO FIGURE 1)

```

;***** Transmitter*****
Xmtr   movlw   8           ; Assume XmtReg contains data to be Xmtd
        movwf  XCount      ; 8 data bits
        bcf   Port_A,DX    ; Send Start Bit
X_next call    Delay       ; Delay for B/2 Seconds
        rrf   XmtReg
        btfsc STATUS,CARRY ; Test the bit to be transmitted
        bsf   Port_A,DX    ; Bit is a one
        btfss STATUS,CARRY
        bcf   Port_A,DX    ; Bit is zero
        decfsz Count       ; If count = 0, then transmit a stop bit
        goto  X_next      ; transmit next bit
;
X_Stop call    Delay
        bsf   Port_A,DX    ; Send Stop Bit
X_Over goto    X_Over
    
```



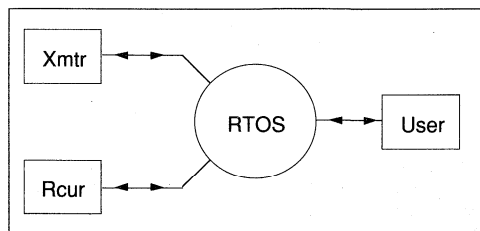
FIGURE 4 - RECEIVE ASSEMBLY CODE (CORRESPONDING TO FIGURE 2)

```

;***** Receiver *****
;
Rcvr   btfsc  Port_A,DR    ; Test for Start Bit
        goto  Rcvr        ; Start Bit not found
        movlw 8           ; Start Bit Detected
        movwf RCount      ; 8 Data Bits
        clrf  RcvReg      ; Receive Data Register
R_next call    Delay       ; Delay for B/2 Seconds, B=Time duration of 1 Bit
        bcf   STATUS,CARRY ; Clear CARRY bit
        rrf   RcvReg      ; to set if MSB first or LSB first
        btfsc Port_A,DR    ; Is the bit a zero or one ?
        bsf   RcvReg,MSB   ; Bit is a one
        call  Delay
        decfsz RCount
        goto  R_next
R_Over goto    R_Over     ; Reception done
    
```

The software is organized such that the communication software acts as a Real Time Operating System (RTOS) which gives control to the User routine for a certain time interval. After this predetermined time slot, the user must give back the control to the Operating System. This is true only in the case of full-duplex implementation. Timing considerations are such that the user gets control for approximately half the time of the bit rate and the rest of the half time is used up by the Operating System (and software delays). Please refer to Table 1 for the delay constants and the time the User gets at 8 MHz input clock. Delay constants and the time that the User gets at 20 MHz and 4 MHz input clock speeds are given in the source code listing of the full duplex routine. At frequencies other than 4, 8, or 20 MHz, the delay constants and the time the User gets can be computed from the equations given in Figure 6.

FIGURE 5 - FULL DUPLEX BLOCK DIAGRAM



Asynchronous Serial I/O

FIGURE 6 - EQUATIONS FOR DELAY CONSTANTS

```
Baud_Cycles = Clkout/Baud ;
User_time = Baud_Cycles * (float) 0.5 ;
K0 = (1.25*Baud_Cycles - 2.0*User_time - 89)/3.0 ; IF (K0 < 0)
{
K0 = 0.0 ;
User_time = 0.50* (1.25*Baud_Cycles - 89.0) ;
}
K1 = (1.25*Baud_Cycles-18 - User_time - 59.0 - 3.K0)/3.0 ;
K2 = (Baud_Cycles - User_time - 41.0 - 3.K0)/3.0 ;
K3 = (Baud_Cycles - User_time - 61.0 - 3.K0)/3.0 ;
K4 = (Baud_Cycles - User_time - 55.0 - 3.K0)/3.0 ;
K5 = (Baud_Cycles - User_time - 55.0 - 3.K0)/3.0 +1.0 ;
K6 = 0.0 ;
K7 = (1.25*Baud_Cycles - User_time - 39.0 - 3.K0)/3.0 ;
```

TABLE 1 - DELAY CONSTANTS AT 8 MHZ INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	-	0	5	39	109
K1	-	39	80	150	288
K2	-	27	51	86	155
K3	-	21	44	80	148
K4	-	23	46	82	150
K5	-	24	47	83	151
K6	-	0	0	0	0
K7	-	45	86	156	295
User Cycles	-	86	208	416	832

TABLE 2 - DELAY CONSTANTS AT 20 MHZ INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	0	13	57	143	317
K1	49	98	184	358	705
K2	34	60	103	191	364
K3	27	53	96	184	357
K4	29	55	98	186	359
K5	30	56	99	187	360
K6	0	0	0	0	0
K7	56	104	190	365	712
User Cycles	118	260	521	1042	2083

For example, if the baud rate selected is 9600 bps (@ 8 MHz), then the total time frame for one bit is approximately 104 μs. Out of this 104 μs, 61 μs is used by the Operating System and the other 43 μs is available to the User. It is the User's responsibility to return control to the Operating System exactly after the time specified in Table 1. For very accurate timing (with resolution up to one clock cycle) the User may set up the RTCC timer with the Prescaler option for calculating the real time. With RTCC set to increment on internal CLKOUT

(500 ns @ 8 MHz CLKIN) and the prescaler assigned to it, very accurate and long timing delay loops may be assigned. This method of attaining accurate delay loops is not used in the RS232 code (RTOS), so that the RTCC is available to the User for other important functions. If the RTCC is not used for other functions, the User may modify the code to replace the software delay loops, by counting the RTCC. For an example of using this method of counting exact timing delays, refer to the "User" routine in Full-Duplex code (Appendix B).

The software uses the minimal processor resources. Only 6 data RAM locations (File Registers) are used. The RTOS uses one level of stack, but it is freed once the control is given back to the user. The watchdog timer and RTCC are not used. The user should clear the watchdog timer at regular intervals, if the WDT is enabled.

The usage of the program is described below. The user should branch to location "Op_Sys" exactly after time as specified in Table 1 or as computed from Equations in Fig. 6. Whereas, the transmission is totally under User control, the Reception is under the control of the Operating System. As long as the user does not set the X_flag, no transmission occurs. On the other hand the Operating System is constantly looking for a start bit and the user should not modify either R_done flag or RcvReg.

TRANSMISSION

Transmit Data is output on DX pin (Bit 0 of Port_A). In the user routine, the user should load the data to be transmitted in the XmtReg and Set the X_flag (bsf FlagRX,X_flag). This flag gets cleared after the transmission. The user should check this flag (X_flag) to see if transmission is in progress. Modifying XmtReg when X_flag is set will transmit erroneous data.

RECEPTION

Data is received on pin DR (Bit 1 of Port_A). The User should constantly check the "R_done" flag to see if reception is over. If the reception is in progress, R_flag is set. If the reception is over, "R_done" flag is set to 1. The "R_done" flag gets reset to zero when a next start bit is detected. The user should constantly check the R_done flag, and if SET, then the received word is in Register "RcvReg". This register gets cleared when a new start bit is detected. It is recommended that the receive register RcvReg be copied to another register after R_done flag is set. The R_done flag also gets cleared when the next start bit is detected.

The User may modify the code to implement an N deep buffer (limited to the number of Data RAM locations available) for receive. Also, if receiving at high speeds, and if the N deep buffer is full, an XOFF signal (HEX 13) may be transmitted. When ready for receiving more data, an XON signal (HEX 11) should be transmitted.

SUMMARY

PIC16C5X family of microcontrollers allow users to implement half or full duplex RS232 communication.

*Author: Amar Palacherla
Logic Products Division*

Asynchronous Serial I/O

APPENDIX A: ASSEMBLY LANGUAGE FOR HALF DUPLEX

MPASM B0.54

PAGE 1

```
; RS-232 Communication With PIC16C54
;
; Half Duplex Asynchronous Communication
;
; This program has been tested at Bauds from 1200 to 19200 Baud
; ( @ 8,16,20 Mhz CLKIN )
;
; As a test, this program will echo back the data that has been
; received.
;
;
; LIST P=16C54, C=80, T=ON
; INCLUDE "PICREG.H"
;***** PIC16C5X Header *****

01FF PIC54 equ 1FFh ; Define Reset Vectors
01FF PIC55 equ 1FFh
03FF PIC56 equ 3FFh
07FF PIC57 equ 7FFh
;
0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h
;
0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h
;
;*****

;
; STATUS REG. Bits
; Carry Bit is Bit.0 of F3
0000 CARRY equ 0h
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h
;
0001 Same equ 1h
;
0000 LSB equ 0h
0007 MSB equ 7h
;
0001 TRUE equ 1h
0001 YES equ 1h
0000 FALSE equ 0h
0000 NO equ 0h
;
;*****
```

Asynchronous Serial I/O

```

;***** Communication Parameters *****
;
0001      X_MODE equ 1      ; If ( X_MODE==1) Then transmit LSB
;
;          if ( X_MODE==0) Then transmit MSB
0001      R_MODE equ 1      ; If ( R_MODE==1) Then receive LSB
;
;          if ( X_MODE==0) Then receive MSB
0001      X_Nbit equ 1      ; if (X_Nbit==1) # of data bits ( T
0001      R_Nbit equ 1      ; if (R_Nbit==1) # of data bits ( R
;
0000      Sbit2 equ 0      ; if Sbit2 = 0 then 1 Stop Bit else
;
;*****
0005      X_flag equ PA0    ; Bit 5 of F3 ( PA0 )
0006      R_flag equ PA1    ; Bit 6 of F3 ( PA1 )
;
0000      DX equ 0         ; Transmit Pin ( Bit 0 of Port A )
0001      DR equ 1         ; Receive Pin ( Bit 1 of Port A )
;
;
0044      BAUD_1 equ .68    ; 3+3X = CLKOUT/Baud
0043      BAUD_2 equ .67    ; 6+3X = CLKOUT/Baud
0022      BAUD_3 equ .34    ; 3+3X = 0.5*CLKOUT/Baud
0056      BAUD_4 equ .86    ; 3+3X = 1.25*CLKOUT/Baud
0042      BAUD_X equ .66    ; 11+3X = CLKOUT/Baud
0042      BAUD_Y equ .66    ; 9 +3X = CLKOUT/Baud
;
;***** Data RAM Assignments *****
;
ORG 08H      ; Dummy Origin
;
0008 0001   RcvReg RES 1      ; Data received
0009 0001   XmtReg RES 1      ; Data to be transmitted
000A 0001   Count RES 1      ; Counter for #of Bits Transmitted
000B 0001   DlyCnt RES 1
;*****
;
;          ORG 0
;
0000 0068   Talk clrf RcvReg      ; Clear all bits of RcvReg
;
0001 0625           btfsc Port_A,DR      ; check for a Start Bit
0002 0A30           goto User          ; delay for 104/2 uS
0003 0923           call Delay4         ; delay for 104+104/4
;*****
;
;          Receiver
;
Rcvr
0004 0C08           IF R_Nbit
;          movlw 8          ; 8 Data bits
;          ELSE
;          movlw 7          ; 7 data bits
;          ENDF
;
0005 002A           movwf Count
0006 0403           R_next bcf STATUS,CARRY

```

Asynchronous Serial I/O

```
0007 0328          IF      R_MODE
                   rrf      RcvReg,Same      ; to set if MSB first or LS
                   ELSE
                   rlf      RcvReg,Same
0008 0625          ENDF
                   btfsz   Port_A,DR
                   ;
                   IF      R_MODE
                   IF      R_Nbit
0009 05E8          bsf      RcvReg,MSB      ; Conditional Assembly
                   ELSE
                   bsf      RcvReg,MSB-1
                   ENDF
                   ELSE
                   bsf      RcvReg,LSB
                   ENDF
                   ;
000A 091F          call     DelayY
000B 02EA          decfsz  Count,Same
000C 0A06          goto     R_next
                   ;*****
000D 0208          R_over  movf   RcvReg,0      ; Send back What is Just Re
000E 0029          movwf   XmtReg
                   ;*****
                   ;      Transmitter
                   ;
Xmtr
000F 0C08          IF      X_Nbit
                   movlw   8
                   ELSE
                   movlw   7
                   ENDF
0010 002A          movwf   Count
                   ;
                   IF      X_MODE
                   ELSE
                   IF      X_Nbit
                   ELSE
                   rlf      XmtReg,Same
                   ENDF
                   ENDF
                   ;
0011 0405          bcf      Port_A,DX      ; Send Start Bit
0012 0925          call     Delay1
0013 0403          X_next  bcf      STATUS,CARRY
                   ;
                   IF      X_MODE
0014 0329          rrf      XmtReg,Same      ; Conditional Assembly
                   ELSE
                   ; to set if MSB first or LS
                   rlf      XmtReg,Same
                   ENDF
                   ;
0015 0603          btfsz   STATUS,CARRY
0016 0505          bsf      Port_A,DX
0017 0703          btfsz   STATUS,CARRY
0018 0405          bcf      Port_A,DX
0019 0921          call     DelayX
001A 02EA          decfsz  Count,Same
001B 0A13          goto     X_next
001C 0505          bsf      Port_A,DX      ; Send Stop Bit
001D 0925          call     Delay1
                   ;
                   IF      Sbit2
                   bsf      Port_A,DX
                   call     Delay1
                   ENDF
```

Asynchronous Serial I/O

```
001E 0A00      ;          goto    Talk          ; Back To Reception & Trans
;
;          End of Transmission
;
001F 0C42      DelayY  movlw   BAUD_Y
0020 0A28          goto    save
0021 0C42      DelayX  movlw   BAUD_X
0022 0A28          goto    save
0023 0C56      Delay4  movlw   BAUD_4
0024 0A28          goto    save
0025 0C44      Delay1  movlw   BAUD_1          ; 104 uS for 9600 baud
0026 0A28          goto    save
0027 0C43      Delay2  movlw   BAUD_2
0028 002B      save    movwf   DlyCnt
0029 02EB      redo_1  decfsz  DlyCnt, Same
002A 0A29          goto    redo_1
002B 0800          retlw   0
;
002C 0C0E      main    movlw   0EH          ; Bit 0 of Port A is Output
;
002D 0005          tris   Port_A          ; Set Port_A.0 as output (
;
002E 0525          bsf   Port_A, DR
;
002F 0A00      goto    Talk
;
;
0030 0C22      User    movlw   BAUD_3
0031 002B          movwf  DlyCnt
0032 02EB      redo_2  decfsz  DlyCnt, Same
0033 0A32          goto    redo_2
0034 0A00      goto    Talk          ; Loop Until Start Bit Foun
;
;
;          ORG    PIC54
01FF 0A2C      goto    main
;
END

Errors : 0
Warnings : 0
```

Asynchronous Serial I/O

APPENDIX B: ASSEMBLY LANGUAGE LISTING FOR FULL DUPLEX

MPASM B0.54 PAGE 1
RS232 Communication Using PIC16C54

```
*****  
; TITLE "RS232 Communication Using PIC16C54"  
;  
; Comments :  
; (1) Full Duplex  
; (2) Tested from 1200 to 9600 Baud( @ 8 Mhz )  
; (3) Tested from 1200 to 19200 Baud(@ 16 & 20 Mhz)  
;  
; The User gets a total time as specified by the User Cycles  
; in the table ( or from equations ). The user routine has to  
; exactly use up this amount of time. After this time the User  
; routine has to give up the control to the Operating System.  
; If less than 52 uS is used, then the user should wait in a  
; delay loop, until exactly 52 uS.  
;  
; Transmission :  
; Transmit Data is output on DX pin (Bit DX of Port_A).  
; In the user routine, the user should load the  
; data to be transmitted in the XmtReg and Set the  
; X_flag ( bsf FlagRX,X_flag ). This flag gets cleared  
; after the transmission.  
;  
; Reception :  
; Data is received on pin DR ( bit DR of Port_A ).  
; The User should constantly check the "R_done" flag  
; to see if reception is over. If the reception is  
; in progress, R_flag is set to 1.  
; If the reception is over, "R_done" flag is set to 1.  
; The "R_done" flag gets reset to zero when a next start  
; bit is detected. So, the user should constantly check  
; the R_done flag, and if SET, then the received word  
; is in Register "RcvReg". This register gets cleared  
; when a new start bit is detected.  
;  
; Program Memory :  
; Total Program Memory Locations Used ( except initialization  
; in "main" & User routine ) = 132 locations.  
;  
; Data Memory :  
; Total Data memory locations (file registers used) = 6  
; 2 File registers to hold Xmt Data & Rcv Data  
; 1 File registers for Xmt/Rcv flag test bits  
; 3 File registers for delay count & scratch pad  
;  
; Stack :  
; Only one level of stack is used in the Operating System/RS232  
; routine. But this is freed as soon as the program returns to the  
; user routine.  
;  
; RTCC : Not Used  
; WDT : Not Used  
;  
; LIST P=16C54  
; INCLUDE "MPREG.H"  
;***** PIC16C5X Header *****  
01FF PIC54 equ 1FFH ; Define Reset Vectors  
01FF PIC55 equ 1FFH  
03FF PIC56 equ 3FFH  
07FF PIC57 equ 7FFH  
;  
0001 RTCC equ 1h  
0002 PC equ 2h  
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
```

Asynchronous Serial I/O

```

0004      FSR      equ      4h
;
0005      Port_A   equ      5h
0006      Port_B   equ      6h      ; I/O Port Assignments
0007      Port_C   equ      7h
;
;*****
;
;          ; STATUS REG. Bits
0000      CARRY   equ      0h      ; Carry Bit is Bit.0 of F3
0000      C        equ      0h
0001      DCARRY  equ      1h
0001      DC       equ      1h
0002      Z_bit   equ      2h      ; Bit 2 of F3 is Zero Bit
0002      Z        equ      2h
0003      P_DOWN  equ      3h
0003      PD       equ      3h
0004      T_OUT   equ      4h
0004      TO       equ      4h
0005      PA0     equ      5h
0006      PA1     equ      6h
0007      PA2     equ      7h
;
0001      Same    equ      1h
;
0000      LSB     equ      0h
0007      MSB     equ      7h
;
0001      TRUE    equ      1h
0001      YES     equ      1h
0000      FALSE  equ      0h
0000      NO      equ      0h
;
;*****

        INCLUDE    "RS232.H"
;*****
; RS232 Communication Parameters
;
;
0001      X_MODE  equ      1      ; If ( X_MODE==1) Then transmit LSB first
; if ( X_MODE==0) Then transmit MSB first ( CODEC like )
0001      R_MODE  equ      1      ; If ( R_MODE==1) Then receive LSB first
; if ( R_MODE==0) Then receive MSB first ( CODEC like )
0001      X_Nbit  equ      1      ; if ( X_Nbit==1) # of data bits ( Transmission ) is
8 else 7
0001      R_Nbit  equ      1      ; if ( R_Nbit==1) # of data bits ( Reception ) is 8
else 7
;
0000      SB2     equ      0      ; if SB2 = 0 then 1 Stop Bit
;          ; if SB2 = 1 then 2 Stop Bit
;*****
;          Transmit & Receive Test Bit Assignments
;
0000      X_flag  equ      0      ; Bit 0 of FlagRX
0002      R_flag  equ      2      ; Bit 1 of FlagRX
0003      S_flag  equ      3      ; Bit 2 of FlagRX
0004      BitXsb  equ      4      ; Bit 3 of FlagRX
0005      A_flag  equ      5
0006      S_bit   equ      6      ; Xmt Stop Bit Flag( for 2/1 Stop bits )
;
0001      R_done  equ      1      ; When Reception complete, this bit is SET
0000      X_done  equ      X_flag ; When Xmission complete, this bit is Cleared
;
0000      DX      equ      0      ; Transmit Pin ( Bit 0 of Port A )
0001      DR      equ      1      ; Recieve Pin ( Bit 1 of Port A )

```

Asynchronous Serial I/O

```

;
;***** Data RAM Assignments *****
;
;          ORG      08H      ; Dummy Origin
;
0008 0001      RcvReg RES    1      ; Data received
0009 0001      XmtReg RES    1      ; Data to be transmitted
000A 0001      Xcount RES    1      ; Counter for #of Bits Transmitted
000B 0001      Rcount RES    1      ; Counter for #of Bits to be Received
000C 0001      DlyCnt RES    1      ; Counter for Delay constant
000D 0001      FlagRX RES    1      ; Transmit & Receive test flag hold register
;
;          Constants      19200  9600   4800   2400   1200
;          ( @ 20 Mhz )
;
;          K0      0      13      57      143      317*
;          K1      49     98      184     358*     705*
;          K2      34     60      103     191     364*
;          K3      27     53       96     184     357*
;          K4      29     55       98     186     359*
;          K5      30     56       99     187     360*
;          K6      0      0        0      0        0
;          K7      56     104     190     365*     712*
;
;          User Cycles   118     260     521     1042    2083
; *****
;
;          Constants      19200  9600   4800   2400   1200
;          ( @ 8 Mhz )
;
;          K0      -      0        5      39      109
;          K1      -     39       80     150     288*
;          K2      -     27       51     86      155
;          K3      -     21       44     80      148
;          K4      -     23       46     82      150
;          K5      -     24       47     83      151
;          K6      -      0        0      0        0
;          K7      -     45       86     156     295*
;
;          User_Cycles   -      86     208     416     832
; *****
;
;          Constants      19200  9600   4800   2400   1200
;          ( @ 4 Mhz )
;
;          K0      -      -        0      5      39
;          K1      -      -        39     80     150
;          K2      -      -        27     51     86
;          K3      -      -        21     44     80
;          K4      -      -        23     46     82
;          K5      -      -        24     47     83
;          K6      -      -         0      0      0
;          K7      -      -        45     86     156
;
;          User_Cycles   -      -      86     208     416
; *****
;
; The constants marked " * " are >255. To implement these constants
; in delay loops, the delay loop should be broken into 2 or more loops.
; For example, 357 = 255+102. So 2 delay loops, one with 255 and
; the other with 102 may be used.

```


Asynchronous Serial I/O

```
*****
;          Set Delay Constants for 9600 Baud @ CLKIN = 8 Mhz
;
0000      K0      EQU      .0
0002      K1      EQU      .39
001B      K2      EQU      .27
0015      K3      EQU      .21
0017      K4      EQU      .23
0018      K5      EQU      .24
0000      K6      EQU      .0
002D      K7      EQU      .45
;
*****

;
          ORG      0
*****
;
0000 0C01      Delay      movlw      K0+1
0001 002C      movwf      DlyCnt          ; Total Delay = 3K+6
0002 02EC      redo      decfsz      DlyCnt, Same
0003 0A02      goto      redo
0004 0800      retlw      0
;
0005 002C      Delay1     movwf      DlyCnt
0006 02EC      redo_1     decfsz      DlyCnt, Same      ;
0007 0A06      goto      redo_1
0008 0A8D      goto      User
;
0009 002C      Delay2     movwf      DlyCnt
000A 02EC      redo_2     decfsz      DlyCnt, Same      ; Delay = = 260 Cycles
000B 0A0A      goto      redo_2
000C 0A67      goto      User_1
;
000D 0625      R_strt     btfsc      Port_A, DR          ; check for a Start Bit
000E 0A17      goto      ShellY          ; delay for 104/2 us
000F 042D      bcf        FlagRX, R_done      ; Reset Receive done flag
0010 054D      bsf        FlagRX, R_flag      ; Set flag for Reception in Progress
0011 078D      btfss      FlagRX, BitXsb
0012 05AD      bsf        FlagRX, A_flag      ; A_flag is for start bit detected in R_strt
0013 0068      clr        RcvReg          ; Clear all bits of RcvReg
          IF      R_Nbit
0014 0C08      movlw      8          ; 8 Data bits
          ELSE
          movlw      7          ; 7 data bits
          ENDIF
0015 002B      movwf      Rcount
0016 0A78      goto      Shell          ; delay for 104+104/4
;
0017 078D      ShellY     btfss      FlagRX, BitXsb
0018 0A78      goto      Shell
0019 054D      bsf        FlagRX, R_flag
001A 0A78      goto      Shell
;
001B 0403      R_next     bcf        STATUS, CARRY
          IF      R_MODE
001C 0328      rrf        RcvReg, Same      ; to set if MSB first or LSB first
          ELSE
          rlf        RcvReg, Same
          ENDIF
001D 0625      btfsc      Port_A, DR
          IF      R_MODE
          IF      R_Nbit
```

Asynchronous Serial I/O

```
001E 05E8      bsf      RcvReg,MSB          ; Conditional Assembly
                ELSE
                bsf      RcvReg,MSB-1
                ENDIF
                ELSE
                bsf      RcvReg,LSB
                ENDIF

001F 02EB      decfsz  Rcount,Same
0020 0A78      goto    Shell
0021 044D      bcf     FlagRX,R_flag
0022 056D      bsf     FlagRX,S_flag
0023 052D      bsf     FlagRX,R_done
0024 0A78      goto    Shell
                ;
                ;           Reception Done
                ;

0025 0405      X_strt  bcf     Port_A,DX          ; Send Start Bit
                IF      X_Nbit
0026 0C08      movlw  8
                ELSE
                movlw  7
                ENDIF

0027 002A      movwf  Xcount
                IF      X_MODE
                ELSE
                IF      X_Nbit
                ELSE
                rlf     XmtReg,Same
                ENDIF
                ENDIF

0028 0A50      goto    X_SB
                ;

0029 0403      X_next  bcf     STATUS,CARRY
                IF      X_MODE

002A 0329      rrf     XmtReg,Same          ; Conditional Assembly
                ELSE          ; to set if MSB first or LSB first
                rlf     XmtReg,Same
                ENDIF

002B 0603      btfsz  STATUS,CARRY
002C 0505      bsf     Port_A,DX
002D 0703      btfss  STATUS,CARRY
002E 0405      bcf     Port_A,DX
002F 00EA      decf   Xcount,Same
0030 0A52      goto    X_Data
                ;

0031 040D      X_SB_1  bcf     FlagRX,X_flag      ; Xmt flag = 0 - transmission over
0032 0C09      movlw  9
0033 002A      movwf  Xcount
0034 0505      bsf     Port_A,DX          ; Send Stop Bit
0035 0A60      goto    X_Stop
                ;

0036 0505      X_SB_2  bsf     Port_A,DX
0037 04CD      bcf     FlagRX,S_bit
0038 0A60      goto    X_Stop
                ;
                ;           End of Transmission
                ;

0039 076D      R0_X0  btfss  FlagRX,S_flag
003A 0A8D      goto    User
003B 046D      bcf     FlagRX,S_flag
003C 0900      call   Delay
003D 0C2E      movlw  K7+1
003E 0A05      goto    Delay1
                ;

003F 0900      R1_X0  call   Delay
0040 0C28      movlw  K1+1          ; delay for 1st bit is 104+104/4
0041 002C      movwf  DlyCnt
                IF      R_Nbit
```

Asynchronous Serial I/O

```

0042 0C08      movlw    8                ; 8 Data bits
                ELSE
                movlw    7                ; 7 data bits
                ENDIF

0043 018B      xorwfc   Rcount,W
0044 0643      btfsc   STATUS,Z_bit
0045 0A06      goto    redo_1
0046 0C1C      movlw   K2+1
0047 0A05      goto    Delay1
                ;

                R1_X1                ; same as R0_X1
0048 0C09      R0_X1  movlw   9
0049 008A      subwfc  Xcount,W
004A 0643      btfsc   STATUS,Z_bit
004B 0A25      goto    X_strt
004C 022A      movf    Xcount,Same        ; to check if All data bits Xmted
004D 0743      btfss  STATUS,Z_bit
004E 0A29      goto    X_next

                IF      SB2
                btfsc  FlagRX,S_bit
                goto   X_SB_2
                bsf   FlagRX,S_bit
                goto   X_SB_1
                ELSE
004F 0A31      goto    X_SB_1
                ENDIF
                ;
                ;
0050 0A51      X_SB   goto    cycle4
0051 0A52      cycle4 goto    X_Data
                ;
0052 06AD      X_Data btfsc   FlagRX,A_flag
0053 0A59      goto    Sbdly
0054 068D      btfsc  FlagRX,BitXsb
0055 0A5D      goto    ABC
0056 0900      call   Delay
0057 0C16      movlw  K3+1
0058 0A09      goto   Delay2
                ;
0059 04AD      Sbdly  bcf     FlagRX,A_flag
005A 0900      call   Delay
005B 0C18      movlw  K4+1
005C 0A09      goto   Delay2
                ;
005D 048D      ABC    bcf     FlagRX,BitXsb
005E 0900      call   Delay
005F 0A67      goto   User_1
                ;

                X_Stop
0060 06AD      btfsc  FlagRX,A_flag
0061 0A59      goto   Sbdly
0062 068D      btfsc  FlagRX,BitXsb
0063 0A5D      goto   ABC
0064 0900      call   Delay
0065 0C19      movlw  K5+1
0066 0A09      goto   Delay2
                ;
0067 064D      User_1 btfsc  FlagRX,R_flag
0068 0A77      goto   Sync_1                ; Reception already in progress
0069 066D      btfsc  FlagRX,S_flag
006A 0A74      goto   Sync_3
006B 0625      btfsc  Port_A,DR            ; check for a Start Bit
006C 0A77      goto   Sync_2                ; No Start Bit - goto User routine
006D 042D      bcf    FlagRX,R_done        ; Reset Receive done flag
006E 044D      bcf    FlagRX,R_flag
006F 058D      bsf    FlagRX,BitXsb        ; Set flag for Reception in Progress
0070 0068      clrf   RcvReg              ; Clear all bits of RcvReg
                IF      R_Nbit

```

Asynchronous Serial I/O

```

0071 0C08      movlw    8                      ; 8 Data bits
                ELSE
                movlw    7                      ; 7 data bits
                ENDIF

0072 002B      movwf    Rcount
0073 0A8D      goto     User
                ;

0074 046D      Sync_3  bcf     FlagRX,S_flag
0075 0C01      movlw    K6+1
0076 0A05      goto     Delay1
                ;
                Sync_1
0077 0A8D      Sync_2  goto     User
                ;
                ;*****
                ;
0078 064D      Shell   btfsc   FlagRX,R_flag
0079 0A7D      goto     Chek_X
007A 060D      btfsc   FlagRX,X_flag
007B 0A48      goto     R0_X1
007C 0A39      goto     R0_X0                      ; Case for R0_X0
007D 060D      Chek_X  btfsc   FlagRX,X_flag
007E 0A48      goto     R1_X1
007F 0A3F      goto     R1_X0
                ;
                ;
                ;*****
                ;       Operating System
                ; The User routine after time = B/2, should branch Here
                ;
0080 074D      Op_Sys  btfss   FlagRX,R_flag
0081 0A0D      goto     R_strt
0082 0A1B      goto     R_next
                ;
                ;*****
                ;
0083 0C0E      main    movlw    0EH                      ; Bit 0 of Port A is Output
0084 0005      tris    Port_A                      ; Set Port_A.0 as output ( DX )
                ; & Port_A.1 is input ( DR )
0085 0505      bsf     Port_A,DX
0086 0C09      movlw    9
0087 002A      movwf    Xcount                      ; If Xcount == 9, Then send start bit
0088 006D      clrf    FlagRX                      ; Clear All flag bits.
                IF      SBZ
                bsf     FlagRX,S_bit; Set Xmt Stop bit flag(2 Stop Bits)
                ELSE
0089 04CD      bcf     FlagRX,S_bit                      ; Clear Xmt Stop bit flag
                ENDIF

008A 0C1F      movlw    1FH                      ; Prescaler = 4
008B 0002      OPTION  ; Set RTCC increment on internal Clock
008C 0A80      goto     Op_Sys
                ;
                ;*****
                ;
                ;       ***** User Routine *****
                ; The User routine should use up time exactly = User time as given
                ; in the Constants Table ( or by Equations for constants ).
                ; At 9600, this 86 Clock Cycles. RTCC timer is used here to count
                ; upto 86 cycles ( From 128-86 To 0 ) by examining Bit 7 of RTCC.
                ;

```

Asynchronous Serial I/O

```
0030          K_user equ    .128+.6-.86
;
008D 0C30      User      movlw  K_user
008E 0021      movwf    RTCC
008F 062D      btfsc   FlagRX,R_done
0090 0A97      goto     ErrChk
0091 060D      SetXmt   btfsc   FlagRX,X_flag
0092 0A9C      goto     Op
0093 0C41      movlw   41H
0094 0029      movwf   XmtReg
0095 050D      bsf     FlagRX,X_flag      ; Enable Xmission
0096 0A9C      goto     Op
;
;          ErrChk
0097 0C5A      movlw   "Z"
0098 0188      xorwf   RcvReg,W
0099 0643      btfsc   STATUS,Z_bit
009A 0A91      goto     SetXmt
009B 0A9B      error   goto     rror      ; Received word is not "Z"
;
009C 07E1      Op       btfss   RTCC,MSB      ; Test for RTCC bit 7
009D 0A9C      goto     Op      ; If Set, Then RTCC has incremented
009E 0A80      Oflow   goto     Op_Sys      ; to 128.
;
; *****
;
;          ORG     PIC54
01FF 0A83      goto     main

;          END

Errors   :    0
Warnings :    0
```

Asynchronous Serial I/O

NOTES:

Using a PIC16C5X as a Smart I²C™ Peripheral

INTRODUCTION

The PIC16C5X microcontrollers from Microchip are ideally suited for use as smart peripheral devices under the control of the main processors in systems due to their low cost and high speed. They are capable of performing tasks which would simply overload a conventional microprocessor, or require considerable logic circuitry, at a cost competitive with lower mid-range PLDs. To minimize the engineering overhead of adding multiple controllers to a product, it is convenient for the auxiliary controllers to emulate standard I/O peripherals.

A common interface found in existing products is the I²C bus. This efficient, 2-wire bi-directional interface allows the designer to connect multiple devices together, with the microprocessor able to send data to and receive data from any device on the bus. This interface is found on a variety of components, such as PLLs, DACs, video controllers, and EEPROMs. If a product already contains one or more I²C devices, it is simple to add a PIC16C5X emulating a compatible component.

This application note describes the implementation of a standard slave device with multiple, bi-directional registers. A subset of the full I²C specification is supported, which can be controlled by the same software which would talk to a Microchip 24LCXX series EEPROM.

THE I²C BUS

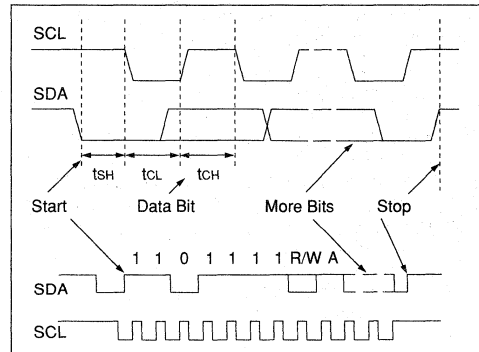
The I²C bus is a master-slave 2-wire interface, consisting of a clock line (SCL) and a data line (SDA). Bi-directional communication (and in a full, multi-master system, collision detection and clock synchronization) is facilitated through the use of a wire-and (ie. active-low, passive high) connection.

The standard-mode I²C bus supports SCL clock frequency up to 100 KHz. The newly released fast-mode I²C bus supports clock rate up to 400 KHz. This application note will support 100 KHz (standard-mode) clock rate.

Each device has a unique seven bit address, which the master uses to access each individual slave device.

During normal communication, SDA is only permitted to change while SCL is low, thus providing two violation conditions (see Figure 1) which are used to signal a start condition (SDA drops while SCL is high) and a stop condition (SDA rises while SCL is high), which frame a message.

FIGURE 1 - I²C TIMING



Each byte of a transfer is 9 bits long (see timing chart in the program listing). The talker sends 8 data bits followed by a "1" bit. The listener acknowledges the receipt of the byte and permission to send the next byte by inserting a "0" bit over the trailing "1". The listener may indicate "not ready for data" by leaving the acknowledge bit as a "1".

The clock is generated by the master only. The slave device must respond to the master within the timing specifications of the I²C definition otherwise the master would be required to operate in slow mode, which most software implementations of I²C masters do not actually support. The specified (standard-mode) tCL is 4.7 μs, and tCH is only 4 μs, so it would be extremely difficult to achieve the timing of a hardware slave device with a conventional microcontroller.

MESSAGE FORMAT

A message is always initiated by the master, and begins with a start condition, followed by a slave address (7 MSBs) and direction bit (LSB = 1 for READ, 0 for WRITE). The addressed slave must acknowledge this byte if it is ready to communicate any data. If the slave fails to respond, the master should send a stop and retry.

If the direction bit is "0" the next byte is considered the sub-address (this is an extension to I²C used by most multi-register devices). The sub-address selects which "register" or function subsequent read or write operations will affect. Any additional bytes will be received and stored in consecutive locations until a stop is sent. If the slave is unable to process more data, it could terminate transfer by not acknowledging the last byte.

A Smart I²C Peripheral

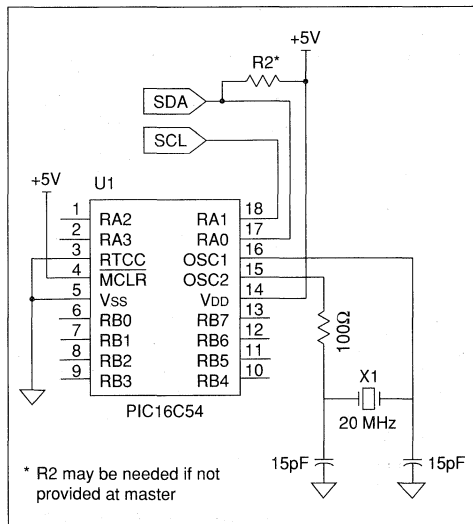
If the direction bit is "1", the slave will transfer successive bytes to the master (while the master holds the line at '1'), while the master acknowledges each byte with a "0" in the ninth bit. The master can terminate the transfer by not acknowledging the last byte, while the slave can stop the transfer by generating a stop condition.

The start address of a read operation is set by sending a write request with a sub-address only (no data bytes). For a detailed set of timing diagrams and different communication modes, consult any of the Microchip 24LCXX EEPROM specifications. This program communicates using the same formats.

IMPLEMENTATION

The chip will respond to slave address "DEVICE_ADDRESS", which by default is D6₁₆ (D7₁₆ for read). This address was chosen because it is the fourth optional address of a Phillips PCF8573 clock/calender or a TDA8443 tiple video switch (unlikely that a product would contain four of those).

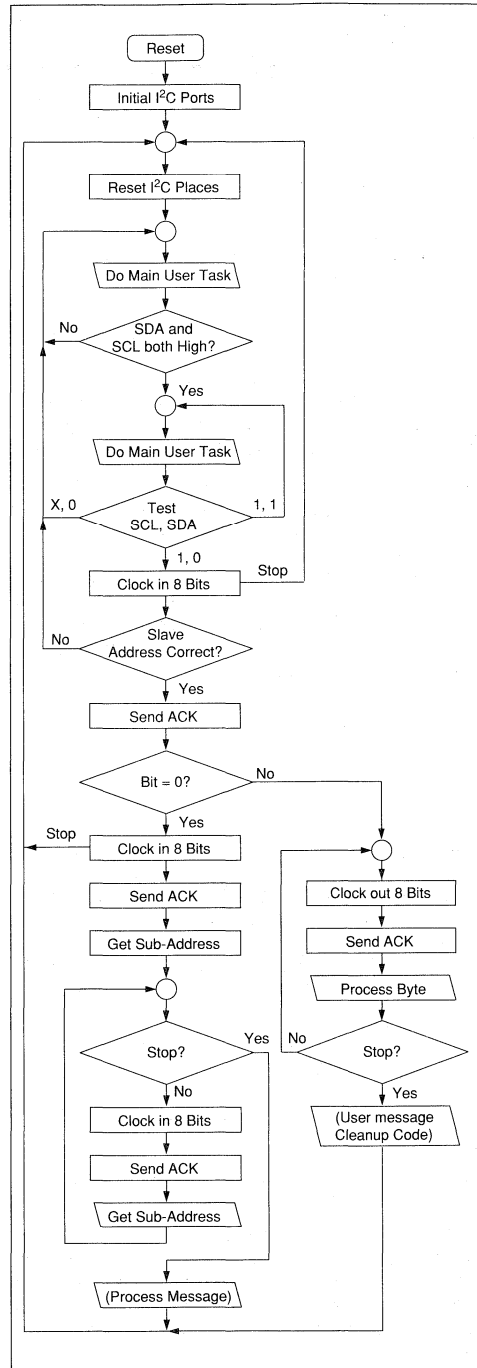
FIGURE 2 - SCHEMATIC OF I²C CONNECTIONS



The connections to the device are shown in Figure 2. The use of RA0 for data in is required. Data is shifted directly out of the port. The code could be modified to make it port independent, but the loss of efficiency may hinder some real-time applications.

This application emulates an I²C device with 8 registers, accessed as sub-addresses 1 through 8 (modulo 7), plus a data channel (0). The example code returns an ID string when the data channel is accessed. When bytes are written to sub-addresses other than 0, they are stored in I2CR0-I2CR7 (I2CR0 gets data written to sub-address 8).

FIGURE 3 - I²C DEVICE FLOWCHART



When the initial sub-address is 0, the flag B:ID is set. This is used to indicate access to a special channel. In this case, the data channel is used to return an ID message, or output data to port B, however the natural extension would be to use this as a data I/O channel.

To make the basic device routines easily adaptable to a variety of uses, macros are used to implement the application specific code. This allows the developer the option of using subroutine calls, or in-line code to avoid the 4 clock cycle overhead and use of the precious stack.

Macro

User code function

USER_MAIN	Code to execute in the main loop while not in a message. If this code takes too long, tSH of 4 μ s will be violated (see Fig. 1). The slave will simply miss the address, not acknowledge, and the master will retry.
USER_Q	This would be quick user code to implement real-time processes. In most applications, this macro would be empty. If used, this routine should be kept under 4 μ s if possible.
USER_MSG	This would be user code to process a message. It is inserted after a message is successfully received.
USER_RECV	This would be user code to process a received byte. It allows the user to add extra code to implement special purpose sub-addresses such as FIFOs.
USER_XMIT	This would be user code to prepare an output byte. In the default routine, it traps sub-address 0 and calls the ID string function.

References:

I²C Bus Specification, Phillips Components, December 1988.

The I²C bus and how to use it (including specification), Signetics/Phillips Semiconductors, January 1992.

Fenger, Carl, "The Inter-Integrated Circuit (I²C) Serial Bus: Theory and Practical Consideration", *Application Note 168*, Phillips Components, December 1988.

"24C16 16K CMOS Serial Electrically Erasable PROM", *Microchip Data Book (1992)*.

About the Author:

Don Lekei has been designing microprocessor based products over 14 years. He has developed many software and hardware products for a wide variety of applications. Mr. Lekei is Manager of Advanced Technologies at NII Norsat International Inc. at their Canadian headquarters in Surrey, British Columbia. Norsat designs and manufactures products to receive broadcast communications from satellites, terrestrial broadcasting systems and optical fibre. Norsat develops technologies and products for satellite entertainment television, broadcast music and data networks.

A Smart I²C Peripheral

APPENDIX A:

MPASM B0.54

PAGE 1

```

                                LIST    P=16C54, C=80, N=0, R=DEC

0676      CPU    EQU    1654
0000      SIM    EQU    0          ;Change timing constants fo

                                IF      (CPU==1654) || (CPU==1655)
01FF      _RESVEC EQU    01FFH      ;16c54 start address
                                ENDIF

                                IF      CPU==1656
                                _RESVEC EQU    03FFH      ;16C56 start address
                                ENDIF

                                IF      CPU==1657
                                _RESVEC EQU    07FFH      ;16C57 start address
                                ENDIF

;*** Reset Vector *****

                                ORG     _RESVEC          ;
RESVEC    GOTO    INIT          ;

01FF 0A0B

;*****

;*****

;* Macros to set/clear/branch/skip on bits
;* These macros define and use synthetic "bit labels"
;* Bit labels contain the address and bit of a location
;*
;*****

;*      Usage              Description
;*      -----
;*
;*      BIT    label,bit,file ;Define a bit label
;*      SEB    label          ;set bit using bit label
;*      CLB    label          ;clear bit using bit label
;*
;*      SKBS   label         ;SKIP on bit set
;*      SKBC   label         ;SKIP on bit clear
;*      BBS    label,address ;BRANCH on bit set
;*      BBC    label,address ;BRANCH on bit clear
;*      CBS    label,address ;CALL on bit set
;*      CBC    label,address ;CALL on bit clear
;*
;*****

BIT    MACRO  label,bit,file ;Define a bit label
label  EQU    file<<8|bit

ENDM

SEB    MACRO  label          ;Set bit
BSF    label>>8,label&7    ;(macro)
ENDM

```



```

CLB      MACRO  label                ;Clear bit
          BCF   label>>8,label&7     ; (macro)
          ENDM                          ;

SKBS     MACRO  label                ;Skip on bit set
          BTFSS label>>8,label&7     ; (macro)
          ENDM

SKBC     MACRO  label                ;Skip on bit clear
          BTFSC label>>8,label&7     ; (macro)
          ENDM

BBS      MACRO  label,address        ;Branch on bit set
          BTFSC label>>8,label&7     ; (macro)
          GOTO  address              ; (macro)
          ENDM                          ;

BBC      MACRO  label,address        ;Branch on bit clear
          BTFSS label>>8,label&7     ; (macro)
          GOTO  address              ; (macro)
          ENDM

CBS      MACRO  label,address        ;Call on bit set
          CALL  label>>8,label&7     ; (macro)
          ENDM                          ;

CBC      MACRO  label,address        ;Call on bit clear
          CALL  label>>8,label&7     ; (macro)
          ENDM

;For Assembler portability

0000     W      EQU  0                ;For file,W
0000     w      EQU  0                ;For file,W
0001     F      EQU  1                ;For file,F
0001     f      EQU  1                ;For file,F

;*****
;* REGISTER DECLARATIONS
;*****

          ORG   0                    ;ORG for register declarati

0000 0001 ind   RES  1                ;0=pseudo-reg 0 for indirec
0001 0001 RTCC  RES  1                ;1=real time counter
0002 0001 PC    RES  1                ;2=PC
0003 0001 STATUS RES 1                ;3=status reg

;* Status reg bits

0300     BIT    B_C,0,STATUS          ;Carry
          B_C   EQU  STATUS<<8|0

0301     BIT    B_DC,1,STATUS        ;Half carry
          B_DC  EQU  STATUS<<8|1

0302     BIT    B_Z,2,STATUS        ;Zero
          B_Z   EQU  STATUS<<8|2
    
```

A Smart I²C Peripheral

```

0303      BIT      B_PD,3,STATUS      ;Power down
          B_PD      EQU      STATUS<<8|3

0304      BIT      B_TO,4,STATUS      ;Timeout
          B_TO      EQU      STATUS<<8|4

0305      BIT      B_PA0,5,STATUS     ;Page select (56/57 only)
          B_PA0     EQU      STATUS<<8|5

0306      BIT      B_PA1,6,STATUS     ;Page select (56/57 only)
          B_PA1     EQU      STATUS<<8|6

0307      BIT      B_PA2,7,STATUS     ;GP flag
          B_PA2     EQU      STATUS<<8|7

0004 0001  FSR      RES      1      ;4=file select reg 0-4=indi
0005 0001  PORTA    RES      1      ;5=port A I/O register (4 b
0006 0001  PORTB    RES      1      ;6=port B I/O register
          IF      (CPU==1655) || (CPU==1657)
0007 0001  PORTC    RES      1      ;7=I/O port C on 16C54/56 only
          ENDIF
          ;registers used by this code

0007 0001  I2CFLG  RES      1      ;I2C flag reg
          ;-i2c flags-----

0700      BIT      B_RD,0,I2CFLG     ;Flag: 1=read
          B_RD      EQU      I2CFLG<<8|0

0701      BIT      B_UA,1,I2CFLG     ;Flag: 0=reading unit address
          B_UA      EQU      I2CFLG<<8|1

0702      BIT      B_SA,2,I2CFLG     ;Flag: 1=reading subaddress
          B_SA      EQU      I2CFLG<<8|2

0703      BIT      B_ID,3,I2CFLG     ;Flag: 1=reading id
          B_ID      EQU      I2CFLG<<8|3

          ;-----

0008 0001  I2CREG  RES      1      ;I2C I/O register
0009 0001  I2CSUBA RES      1      ;Subaddress
000A 0001  I2CBITS RES      1      ;I2C xmit bit counter

          ;*****
          ;* 8 Pseudo registers accessed by sub-addresses 1-8
          ;* (address 0 accesses the ID string)
          ;* these are read-write registers
          ;*****

000B      I2CR0    EQU      $      ;Sub-address 8

```

A Smart I²C Peripheral

```
000B 0001          RES    1          ;8 pseudo registers

000C          I2CR1  EQU    $          ;Sub-address 1
000C 0001          RES    1

000D          I2CR2  EQU    $          ;Sub-address 2
000D 0001          RES    1

000E          I2CR3  EQU    $          ;Sub-address 3
000E 0001          RES    1

000F          I2CR4  EQU    $          ;Sub-address 4
000F 0001          RES    1

0010          I2CR5  EQU    $          ;Sub-address 5
0010 0001          RES    1

0011          I2CR6  EQU    $          ;Sub-address 6
0011 0001          RES    1

0012          I2CR7  EQU    $          ;Sub-address 7
0012 0001          RES    1

;Constants used by program

00D6          DEVICE_ADDRESS EQU    0D6H ;I2C device address

;*****

;** PORTA DEFINITIONS
;** I2C interface uses PORTA
;** note SDA goes to A0 for code efficiency
;**
;*****

00F7          TAREAD EQU    B'11110111' ;TRISA register for SDA rea
00F6          TAWRITE EQU    B'11110110' ;TRISA register for SDA wri
00F7          TAINIT EQU    TAREAD      ;Initial TRISA value

          BIT    B_SDA,0,PORTA          ;I2C SDA (data) This must be bit 0!
0500          B_SDA EQU    PORTA<<8|0

          BIT    B_SCL,1,PORTA          ;I2C SCL (clock)
0501          B_SCL EQU    PORTA<<8|1

;spare          B_???,2,PORTA ;not used
;spare          B_???,3,PORTA ;not used

;*****

;**
;** Port B definition (Parallel out)
;**
;*****

0000          TBINIT EQU    B'00000000' ;Port B tris (all output)
00FF          PBINIT EQU    B'11111111' ;Port B init

;*****

;* Macros to contain user POLL loop code.
```

A Smart I²C Peripheral

```
;* These are implimented as macros to allow ease of modifc
;* especially in real-time applications. The functions coul
;* in-line code or as subroutines depending on ROM/time tra
;*
;* USER_MAIN:  Decision or code to perform at idle time
;*
;* USER_Q:     'Quick' code for use during transfer - max
;*
;*             I2C Spec. More than 4 Ês may result in I2C
;*             full spec speed.
;*
;* USER_MSG:   Code to execute at receipt of I2C command.
;*
;*****

USER_MAIN      MACRO
;*** This would be user code for idle loop
                ENDM

USER_Q         MACRO
;*** This would be quick user code
                ENDM

USER_MSG       MACRO
;*** This would be user code to process a message
                ENDM

USER_RECV      MACRO
;*** This would be user code to process a received byte
;*** example code sends sub-address 0 to port b
                BBC     B_ID,_NXI_notid           ;Channel 0! Bit set if
;
                MOVFW  I2CREG                    ;get received byte
                MOVWF  PORTB                     ;and write it on portb
                GOTO   IN_CONT
;
                _NXI_notid
                ENDM

USER_XMIT      MACRO
;*** This would be user code to prepare an output byte
;*** example code sends id string to output
                BBC     B_ID,_NXO_notid           ;Channel 0! Bit set if
;
                CALL   GETID                     ;get next byte from ID
                GOTO   OUT_CONT                   ;and send it
;
                _NXO_notid
                ENDM

;*****

; START OF CODE
;*****

                ORG     0
;*****

;* Device ID Table (must be at start)
;* TABLE FOR UNIT ID returns next char in W
;*****

GETID
0000 0209      MOVWF  I2CSUBA                      ;W=I2CSUBA
0001 0E07      ANDLW  07H                          ;Limit to 8 locations
```

A Smart I²C Peripheral

```
0002 01E2          ADDWF  PC,F
;*****
;* Device ID text: read starting at sub-address 0
;*****

0003 0850          RETLW  'P'
0004 0849          RETLW  'I'
0005 0843          RETLW  'C'
0006 0849          RETLW  'I'
0007 0832          RETLW  '2'
0008 0843          RETLW  'C'
0009 0800          RETLW  0
000A 0800          RETLW  0

;*****

;* I2C Device routines
;*
;* Enable must be HIGH, else state goes to 0
;* write is to me, read is from me.
;*
;*          <===== first byte / subsequent write
;*
;* SDA  -|  X-X-X-X-X-X-X-X-
;*
;*          |-X-X-X-X-X-X-X-X-
;* (bit)  s   7   6   5   4   3   2   1   0
;* SCL  -|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
;*
;*          |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
;*
;* STATE: 0 1 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
;*
;*          <===== subsequent reads =====
;*
;* SDA    X-X-X-X-X-X-X-X-
;*
;*          -X-X-X-X-X-X-X-X-
;* (bit)ack  7   6   5   4   3   2   1   0
;* SCL  -|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
;*
;*          |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
;*
;* STATE: 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8
;*
;*          <===== Final READ =====
;*
;* SDA    X-X-X-X-X-X-X-X-
;*
;*          -X-X-X-X-X-X-X-X-
;* (bit)ack  7   6   5   4   3   2   1   0
;* SCL  -|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
```

A Smart I²C Peripheral

```

;*          |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |
;*
;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8
;*
;*
;* STATE B is an ignore bit state for non-addressed bits
;* STATE C indicates last sample had ENA low
;* on rising edge of ENA, DATA LOW = low voltage, DATA&CLOC
;*****

;I2C interface uses PORTA
;note SDA must be on PORTA,0 for code efficiency
;*****

;** INIT
;** Hardware reset entry point
;**
;*****

INIT      ;Power-on entry
;*****

;** RESET
;** software reset entry point
;**
;*****

RESET                                ;Soft reset

000B 0CF7          MOVLW  TAINIT          ;Init ports
000C 0005          TRIS   PORTA
000D 0C00          MOVLW  TBINIT
000E 0006          TRIS   PORTB
000F 0CFE          MOVLW  PBINIT
0010 0026          MOVWF  PORTB

;*****
; Main wait loop while idle. POLL loop should be called her
;
;*****

I2CWAIT

0011 0004          CLRWDT                ;Clear watchdog timer
0012 0427          CLB      B_UA          ;Init state flags
                   BCF      B_UA>>8,B_UA&7

0013 0447          CLB      B_SA          ;Init state flags
                   BCF      B_SA>>8,B_SA&7

0014 0407          CLB      B_RD          ;Init state flags
                   BCF      B_RD>>8,B_RD&7

0015 0004          loop1  CLRWDT                ;Clear watchdog timer

USER_MAIN          ;Call user code while in idle state
;*** This would be user code for idle loop

```


A Smart I²C Peripheral

```

0016 0605          SKBC   B_SDA          ;Wait for SDA&SCL=H
                   BTFS   B_SDA>>8,B_SDA&7

                   loop2
0017 0725          SKBS   B_SCL          ;
                   BTFS   B_SCL>>8,B_SCL&7

0018 0A15          GOTO   loop1          ; No longer valid to wait f

0019 0004          CLRWDT          ;Clear watchdog timer

                   USER_MAIN          ;Call user code while in idle state
;*** This would be user code for idle loop

;** wait for start **
001A 0625          SKBC   B_SCL          ;Clock has dropped
                   BTFS   B_SCL>>8,B_SCL&7

001B 0605          SKBC   B_SDA          ;Data dropped... Start!
                   BTFS   B_SDA>>8,B_SDA&7

001C 0A17          GOTO   loop2

;** START RECEIVED! - wait for first bit!
loop3
001D 0605          BBS    B_SDA,I2CWAIT  ;Data raised before clock dropped -
                   BTFS   B_SDA>>8,B_SDA&7

001E 0A11          GOTO   I2CWAIT

001F 0625          BBS    B_SCL,loop3    ;Wait for clock low
                   BTFS   B_SCL>>8,B_SCL&7

0020 0A1D          GOTO   loop3

NEXTBYTE
0021 0004          CLRWDT          ;Clear watchdog timer
0022 0C01          MOVLW  1            ;Init receive byte so bit f
0023 0028          MOVWF  I2CREG

;** Shift bits! - external poll may be executed during low
;* ENABLE line is checked for loss of enable ONLY during HI

;*** CLOCK IS LOW - DATA MAY CHANGE HERE
;*** We have at least 4 Es before any change can occur

loop4
                   USER_Q
;*** This would be quick user code

loop4A
0024 0725          BBC    B_SCL,loop4A   ;Wait for clock high
                   BTFS   B_SCL>>8,B_SCL&7

```

A Smart I²C Peripheral

```
0025 0A24          GOTO    loop4A

;*** CLOCK IS HIGH - SHIFT BIT - then watch for change
0026 0305          RRF     PORTA,W      ;Move RA0 into C
0027 0368          RLF     I2CREG,F    ;Shift in bit
0028 0603          SKPNC          ;Skip if not done
0029 0A36          GOTO    ACK_I2C    ;Acknowledge byte

002A 0608          BTFSC   I2CREG,0    ;Skip if data bit was 0
002B 0A31          GOTO    ii_1      ;This bit was set
ii_0
002C 0725          BBC     B_SCL,loop4    ;Wait for clock low
BTFSS B_SCL>>8,B_SCL&7

002D 0A24          GOTO    loop4

002E 0705          SKBS    B_SDA          ;Data low-high == stop
BTFSS B_SDA>>8,B_SDA&7

002F 0A2C          GOTO    ii_0
I2CSTOP
USER_MSG          ;process completed message!
;*** This would be user code to process a message

0030 0A11          GOTO    I2CWAIT        ;back to main loop

ii_1
0031 0705          BBC     B_SDA,I2CWAIT    ;Data high-low == start
BTFSS B_SDA>>8,B_SDA&7

0032 0A11          GOTO    I2CWAIT

0033 0725          BBC     B_SCL,loop4    ;Wait for clock low
BTFSS B_SCL>>8,B_SCL&7

0034 0A24          GOTO    loop4

0035 0A31          GOTO    ii_1

ACK_I2C
0036 0727          BBC     B_UA,ACK_UA    ;Not addressed - check unit address
BTFSS B_UA>>8,B_UA&7

0037 0A8B          GOTO    ACK_UA

0038 0647          BBS     B_SA,ACK_SA    ;Reading secondary address
BTFSC B_SA>>8,B_SA&7

0039 0A97          GOTO    ACK_SA

;****
;** Do what must be done with new data bytes here (before A

;** Don't ack if byte can't be processed!
;****
;-----

USER_RECV
;*** This would be user code to process a received byte
;*** example code sends sub-address 0 to port b
```

A Smart I²C Peripheral

```
003A 0767                                BTFSS  B_ID>>8,B_ID&7
003B 0A3F                                GOTO   _NXI_notid

003C 0208          MOVWF  I2CREG          ;get received byte
003D 0026          MOVWF  PORTB          ;and write it on portb
003E 0A47          GOTO   IN_CONT
                    _NXI_notid

003F 0C07          MOVLW  07H              ;Register count
0040 0169          ANDWF  I2CSUBA,f          ;Limit register count
0041 0C0B          MOVLW  I2CRO              ;Pseudo-registers
0042 01C9          ADDWF  I2CSUBA,W          ;Offset from buffer start
0043 02A9          INCF  I2CSUBA          ;Next sub-address
0044 0024          MOVWF  FSR              ;Indirect address
0045 0208          MOVWF  I2CREG
0046 0020          MOVWF  ind              ;Put data into register

                    IN_CONT          ;continue point for interce

                    ACKloop

0047 0625          BBS    B_SCL,ACKloop  ;Wait for clock low
                    BTFSC  B_SCL>>8,B_SCL&7

0048 0A47          GOTO   ACKloop

                    CLB    B_SDA          ;Set ACK
0049 0405          BCF    B_SDA>>8,B_SDA&7

004A 0CF6          MOVLW  TAWRITE
004B 0005          TRIS  PORTA
                    CLB    B_SDA          ;Set ACK (just in case docs are wrong)
004C 0405          BCF    B_SDA>>8,B_SDA&7

                    ACKloop2
                    USER_Q
                    ;*** This would be quick user code

004D 0725          BBC    B_SCL,ACKloop2 ;Wait for clock high
                    BTFSS  B_SCL>>8,B_SCL&7

004E 0A4D          GOTO   ACKloop2

                    ACKloop3
                    USER_Q
                    ;*** This would be quick user code

004F 0625          BBS    B_SCL,ACKloop3 ;Wait for clock low
                    BTFSC  B_SCL>>8,B_SCL&7

0050 0A4F          GOTO   ACKloop3

0051 0CF7          MOVLW  TAREAD          ;End ACK
0052 0005          TRIS  PORTA

                    BBC    B_RD,NEXTBYTE  ;Skip if read (we were acking address on
```

A Smart I²C Peripheral

```
0053 0707                BTFSS  B_RD>>8,B_RD&7
0054 0A21                GOTO   NEXTBYTE

;*****
; I2C Readback (I2C read request)
; Application specific code to get bytes to send may be add
; This routine gets data from location pointed to by I2CSUB
; sends it to I2C. Subsequent reads get sequential addresse
; AND's the register # with 7 to limit to 8 registers (for
; could be modified to do a comparison to an absolute numbe
;
;*****

NEXTOUT
;*** <<< PUT NEXT BYTE INTO I2CREG HERE NOW! >>> ***

USER_XMIT
;*** This would be user code to prepare an output byte
;*** example code sends id string to output

0055 0767                BTFSS  B_ID>>8,B_ID&7
0056 0A59                GOTO   _NXO_notid
0057 0900                CALL   GETID           ;get next byte from ID chan
0058 0A60                GOTO   OUT_CONT       ;and send it
                        _NXO_notid

0059 0C07                MOVLW  07H           ;Register count
005A 0169                ANDWF  I2CSUBA,f     ;Limit register count
005B 0C0B                MOVLW  I2CR0        ;Pseudo-registers
005C 01C9                ADDWF  I2CSUBA,W     ;Offset from buffer start
005D 02A9                INCF  I2CSUBA       ;Next sub-address
005E 0024                MOVWF  FSR          ;Indirect address
005F 0200                MOVWF  ind          ;Get data from register

OUT_CONT
MOVWF  I2CREG
;- add code here to init I2CREG! when B_ID is clear!
0061 0C08                MOVLW  8             ;Bit counter
0062 002A                MOVWF  I2CBITS

;** OUT bits! - external poll may be executed during low c
;
;                may also be executed during high cycle if

;* ENABLE line is checked for loss of enable ONLY during HI

;*** CLOCK IS LOW - CHANGE DATA HERE FIRST!

;*** loop 1: data was 1
iiOUT_loop_1
0063 0368                RLF   I2CREG,F     ;Shift data out, MSB first
```

A Smart I²C Peripheral

```
0064 0603          SKPNC          ;1->0: change
0065 0A79          GOTO    iiOUT_1    ;Output another 1!
                   CLB     B_SDA    ;Output 0
0066 0405          BCF     B_SDA>>8,B_SDA&7

0067 0CF6          MOVLW   TAWRITE
0068 0005          TRIS   PORTA
                   CLB     B_SDA    ;Set data (just in case docs are
0069 0405          BCF     B_SDA>>8,B_SDA&7

iiOUT_0
006A 0004          CLRWDT          ;Clear watchdog timer

USER_Q
;*** This would be quick user code

iiOUT_loop_02
006B 0725          BBC     B_SCL,iiOUT_loop_02 ;Wait for clock high
                   BTFSS  B_SCL>>8,B_SCL&7
006C 0A6B          GOTO    iiOUT_loop_02

USER_Q
;*** This would be quick user code

iiOUT_loop_03
006D 0625          BBS    B_SCL,iiOUT_loop_03 ;Wait for clock low
                   BTFSC  B_SCL>>8,B_SCL&7
006E 0A6D          GOTO    iiOUT_loop_03

006F 02EA          DECFSZ I2CBITS          ;Count bits
0070 0A74          GOTO    iiOUT_loop_0    ;Loop for last bit 0
0071 0CF7          MOVLW   TAREAD          ;Done with last bit 0... Se

0072 0005          TRIS   PORTA
0073 0A80          GOTO    iiOUT_ack      ;Get ACK

iiOUT_loop_0
0074 0368          RLF    I2CREG,F        ;Shift data out, MSB first

0075 0703          SKPC          ;0->1: change
0076 0A6A          GOTO    iiOUT_0        ;Output another 0!

0077 0CF7          MOVLW   TAREAD          ;Set to 1
0078 0005          TRIS   PORTA

iiOUT_1
0079 0004          CLRWDT          ;Clear watchdog timer

USER_Q
;*** This would be quick user code

iiOUT_loop_12
007A 0725          BBC     B_SCL,iiOUT_loop_12 ;Wait for clock high
                   BTFSS  B_SCL>>8,B_SCL&7
```

A Smart I²C Peripheral

```
007B 0A7A          GOTO  iiOUT_loop_12

USER_Q
;*** This would be quick user code

iiOUT_loop_13
BBS  B_SCL,iiOUT_loop_13  ;Wait for clock low
BTFS  B_SCL>>8,B_SCL&7
007C 0625
007D 0A7C          GOTO  iiOUT_loop_13

007E 02EA          DECSZ  I2CBITS          ;Count bits
007F 0A63          GOTO  iiOUT_loop_1          ;Loop for last bit 1

iiOUT_ack
INCF  I2CSUBA          ;Get acknowledge
;Next sub-address
0080 02A9

iiOUT_loop_a2
BBC  B_SCL,iiOUT_loop_a2  ;Wait for clock high
BTFS  B_SCL>>8,B_SCL&7
0081 0725
0082 0A81          GOTO  iiOUT_loop_a2

BBS  B_SDA,I2CWAIT      ;No ACK - wait for restart!
BTFS  B_SDA>>8,B_SDA&7
0083 0605
0084 0A11          GOTO  I2CWAIT

;- prepare next character here!

iiOUT_loop_a3
BBC  B_SCL,NEXTOUT      ;Wait for clock low - output next
BTFS  B_SCL>>8,B_SCL&7
0085 0725
0086 0A55          GOTO  NEXTOUT

BBS  B_SDA,iiOUT_loop_a3  ;Watch out for new start condition
BTFS  B_SDA>>8,B_SDA&7
0087 0605
0088 0A85          GOTO  iiOUT_loop_a3

0089 0A11          GOTO  I2CWAIT          ;Stop received!
008A 0A11          GOTO  I2CWAIT

;*****
;* Unit address received - check for valid address
;*
;*****
ACK_UA
```

A Smart I²C Peripheral

```

008B 0527          SEB    B_UA          ;Flag unit address received
                   BSF    B_UA>>8,B_UA&7

008C 0608          BTFSC  I2CREG,0        ;Skip if data coming in
                   SEB    B_RD          ;Flag - reading from slave
008D 0507          BSF    B_RD>>8,B_RD&7

008E 0208          MOVF   I2CREG,W         ;Get address
008F 0EFE          ANDLW  0FEH           ;Mask direction flage befor

0090 0FD6          XORLW  DEVICE_ADDRESS ;Device address
0091 0743 0A11     BNZ    I2CWAIT      ;Not for me! (skip rest of

                   BBS    B_RD,ACKloop ;Read - no secondary address
0093 0607          BTFSC  B_RD>>8,B_RD&7

0094 0A47          GOTO   ACKloop

                   SEB    B_SA          ;Next is secondary address
0095 0547          BSF    B_SA>>8,B_SA&7

0096 0A47          GOTO   ACKloop      ;Yes! ACK address and conti

```

;* Secondary address received - stow it!
 ;* SA = 0 is converted to 128 to facilitate ID read

```

                   ACK_SA
0097 0447          CLB    B_SA          ;Flag second address received
                   BCF    B_SA>>8,B_SA&7

                   CLB    B_ID
0098 0467          BCF    B_ID>>8,B_ID&7

0099 0208          MOVFW  I2CREG        ;Get subaddress
009A 0643          SKPNZ                ;Not 0
                   SEB    B_ID          ;Flag - id area selected
009B 0567          BSF    B_ID>>8,B_ID&7

009C 0029          MOVWF  I2CSUBA       ;Set subaddress
009D 0A47          GOTO   ACKloop

```

END

```

Errors   :    0
Warnings :    0

```



A Smart I²C Peripheral

NOTES:

PLD Replacement

INTRODUCTION

The PIC16C5X microcontrollers are ideal for implementing low cost combinational and sequential logic circuits that traditionally have been implemented using either numerous TTL gates or using programmable logic chips such as PLAs or EPLDs.

PIC16C5X is a family of high-performance 8-bit microcontrollers from Microchip Technology. It employs Harvard architecture, i.e has a separate data bus (8-bit) and a program bus (12-bit wide). All instructions are single word and execute in one cycle except for program branches. The instruction cycle time is 200 ns at 20 MHz and faster versions with clock frequency of 20 MHz (instruction cycle = 200 ns) are planned. The PIC16C5X microcontrollers are ideal for PLD-type application because:

- * Very low cost. Extremely cost effective to replace TTL gates or expensive EPLD's.
- * Fully programmable. PIC16C5X microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
- * Available off the shelf from distributors.
- * PC board real estate saving can be substantial when replacing multitude of TTL's or several PLD's with PIC16C5X microcontrollers which are packaged in 18 and 28 pin packages (DIP, PLCC, SOIC).
- * Substantial power savings can be attained by using PIC16C5X's SLEEP mode. In this mode typical power consumption of PIC16C5X is less than 1uA.
- * PIC16C5X's I/O ports are bidirectional and software configurable as input or output. The user can mix and match number of inputs or outputs as long as the total does not exceed 20 (PIC16C55/57).
- * PIC16C5X's output pins have large current source/sink capability. They can directly drive LED's.
- * The speed and efficiency of the PIC16C5X allows it to perform other control, timing, and compute functions in addition to implementing a PLA function.

IMPLEMENTING A PLA

To implement a generic combinational logic function, we can simply emulate an AND-OR PLA in software. This will require that the logic outputs be described as sum of products. To describe our algorithm, we will use a simple 8-input, 8-bit output PLA with 24 product terms (Figure 1). We will further use the truth table in Figure 2 as the PLA function being implemented. In this example,

only four inputs (A3: A0) are used and the other four inputs (A7: A4) are don't care. On the output side, seven output pins (Y6: Y0) are used and Y7 is unused. To implement this PLA, the logic inputs A0,A1,....,A7 can be connected to port RB pins RB0,RB1,....,RB7 respectively. The logic outputs Y0,Y1,....,Y7 will appear on port RC pins RC0,RC1,....,RC7 respectively. Port RB is configured as input and port RC will be configured as output. To evaluate each product term one XOR (exclusive OR) and one AND operation will be required. For example, to determine product term P3 = A3.A2.A1.A0, the expression to evaluate is:

$$(A<7:0> .XOR. XXXX0011B) .AND. 00001111B).$$

The constant with which XOR is done will be referred to as P3_x in our discussion. P3_x = XXXX0011B will ensure that if A<3:0> = 0011B, the least significant 4 bits of the result will be 0000b. The AND constant, referred to here as P3_a (Product term 3, AND constant) basically eliminates the don't care inputs (here A<7:4>) by masking them. Therefore, if the result of the XOR-AND operation is zero then P3 = 1 else P3 = 0. Once the Product terms are evaluated they are stored in four product registers Preg_0 to Preg_3. To determine an output term:

$$Y0 = P0 + P2 + P3 + P5 + P6 + P7 + P8 + P9 + P10 + P12 + P13 + P14$$

we need to evaluate the following expression:

$$(Preg_a .AND. OR_a0) .OR. (Preg_b .AND. OR_b0) .OR. (Preg_c .AND. OR_c0)$$

In our case the constant values to implement Y0 are as follows:

$$OR_a0 = \begin{matrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ & P7 & P6 & P5 & P3 & P2 & P0 & \end{matrix}$$

$$OR_b0 = 11010111$$

$$OR_c0 = 00000000$$

For larger number of inputs, outputs or product terms, the evaluation will be more complex but following the same principle. Appendix A shows the assembly code to implement this 8 input X 8 output X 24 Product PLA. This example optimizes speed as well as program memory requirement. Appendix B shows a slightly different implementation (only EVAL_Y MACRO is different) that optimizes program memory usage over speed. Table 1 shows time and resources required to implement different size PLAs.



PLD Replacement

FIGURE 1 - A SIMPLE PLA

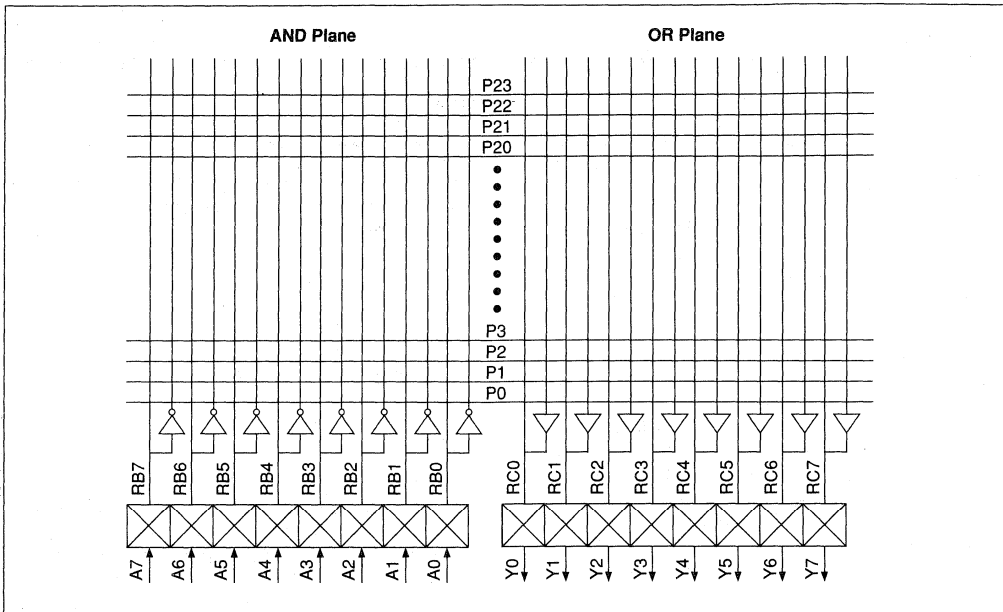
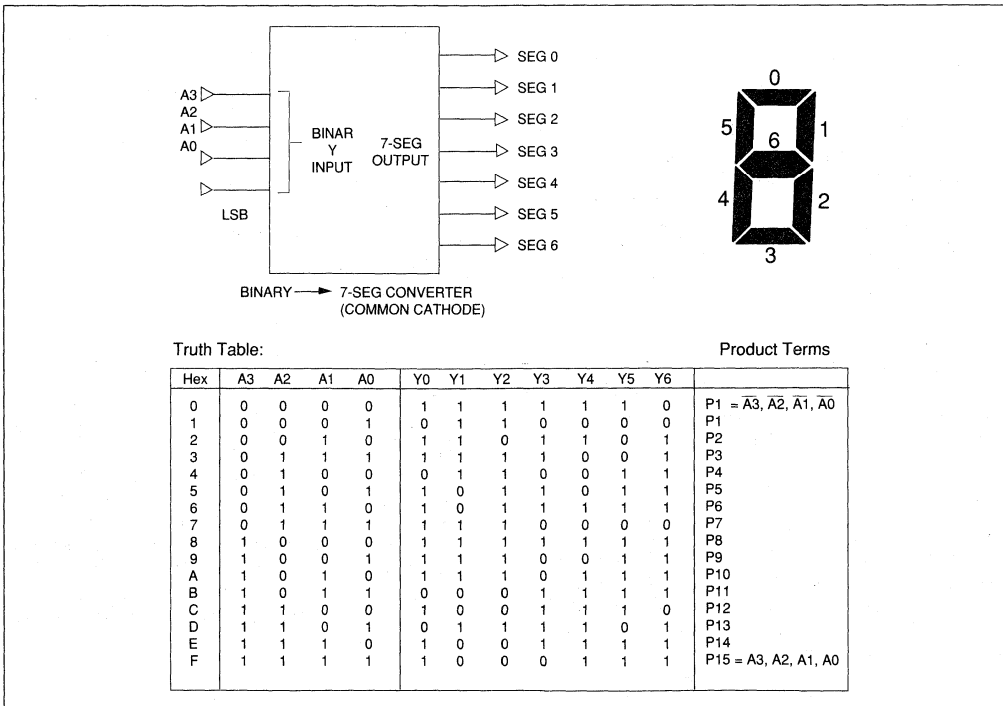


FIGURE 2 - BINARY TO 7-SEGMENT CONVERSION EXAMPLE



SPEED/RESPONSE TIME

The worst case response time of a PLA implemented in this fashion can be calculated as follows. First, we define t_d = time required to execute the PLA program assuming the worst case program branches are taken. Then the maximum propagation delay time from input change to valid output = $2t_d$. This is because if an input changes just after the program reads input port, its effect will not show up until the program completes the current execution cycle, re-reads input and recalculates output. This is shown in Figure 3. There are ways to improve the delay time such as sample inputs several times throughout the PLA program and if input change is sensed, return to the beginning rather than execute the rest of the evaluation code.

A Table Look-Up Method For Small PLA Implementation

If the number of inputs is small (8 or less) then a simple table look-up method can be used to implement the PLA. This will improve execution time to around 5 μ s (@ 8 MHz input clock). The following code implements the BCD to 7-segment conversion (Figure 2) using this technique.

```

begin    movlw    0ffh        ;
        tris     6           ;Port_b = input
        clrw     ;
        tris     7           ;Port_c = output
pla 88   movf     Port_b, w    ;Read input
        andlw   0fh         ;Mask off bits 7:4
        call    op_tbl       ;
        movwf   Port_c       ;Write output
        goto    pla88        ;
op_tbl   addw    pc           ;Computed jump for
        ;                   table look-up

        retlw   b"00111111";
        retlw   b"00000110";
        retlw   b"01011011";
        retlw   b"01001111";
        retlw   b"01100110";
        retlw   b"01101101";
        retlw   b"01111101";
        retlw   b"00000111";
        retlw   b"01111111";
        retlw   b"01100111";
        retlw   b"01110111";
        retlw   b"01111000";
        retlw   b"00111001";
        retlw   b"01011110";
        retlw   b"01111001";
        retlw   b"01110001";
    
```



TABLE 1 - EXECUTION TIME AND RESOURCES NECESSARY FOR DIFFERENT SIZE PLA's

Number of Inputs Including fdbk	Number of Outputs Including fdbk and o/e Control	Number of Products	Number of RAM Locations Required NRAM	Number of Program Memory Locations Required NROM	Number of Instruction Cycle To Execute PLA NCYC	Real Time @ 20 MHz to Execute PLA	
8	8	24	5	228	228	45.6 μ s	Time Efficient
			10	222	352	70.4 μ s	Code Efficient
8	8	48	8	447	447	89.4 μ s	Time Efficient
			13	384	535	107 μ s	Code Efficient
16	16	64	12	1042	1042	208.4 μ s	Time Efficient
			22	843	1250	250 μ s	Code Efficient
20	24	80	16	1661	1661	372.2 μ s	Time Efficient
			40	1462	2221	444.2 μ s	Code Efficient

If Ni = Number of inputs NIW = Number of input words, $NIW = \lceil \frac{Ni}{8} \rceil$ NP = Number of products NPW = Number of product words, i.e. $NPW = \lceil \frac{NP}{8} \rceil$ NO = Number of outputs NOW = Number of output words, i.e. $NOW = \lceil \frac{NO}{8} \rceil$	Then, NRAM @ NIW + NOW + N PW : Time efficient NRAM @ NIW + NOW + 2 N PW + 2: Code efficient NROM @ 8 + N PW + NOW + N P [2 + 3 NIW] + NO * NPW + 4: Time efficient NROM @ 17 + N PW + NOW + N P [2 + 3 NIW] + NO [NPW + 3]: Code efficient NCYC @ 8 + N PW + NOW + N P [2 + 3 NIW] + NO [2 NPW + 4]: Time efficient NCYC @ 8 + N PW + NOW + N P [2 + 3 NIW] + 5 N [NPW + 1]: Code efficient
---	---

PLD Replacement

FIGURE 3 - PLA PROGRAM FLOW

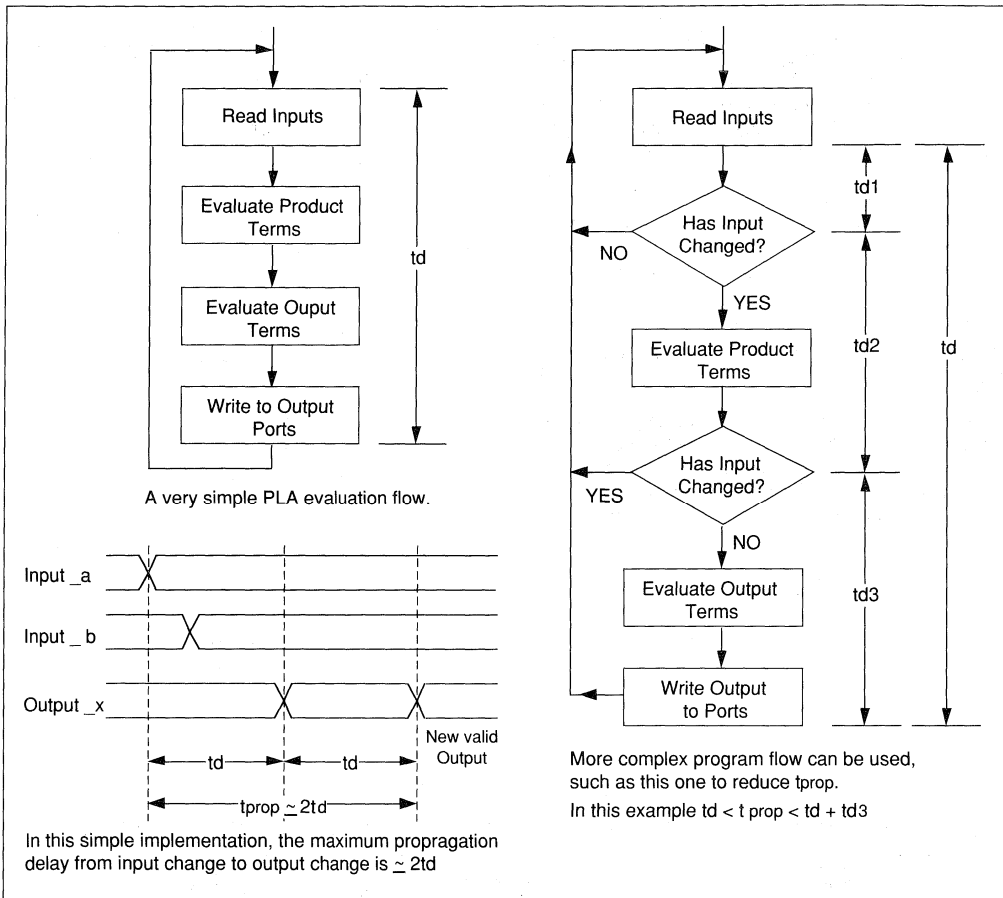
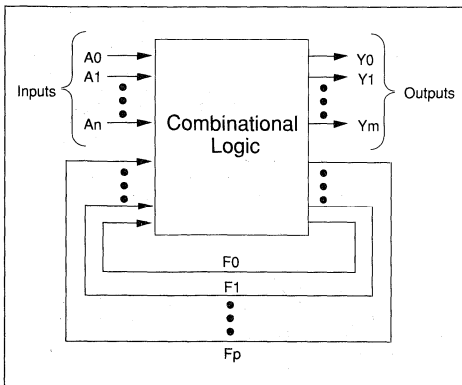


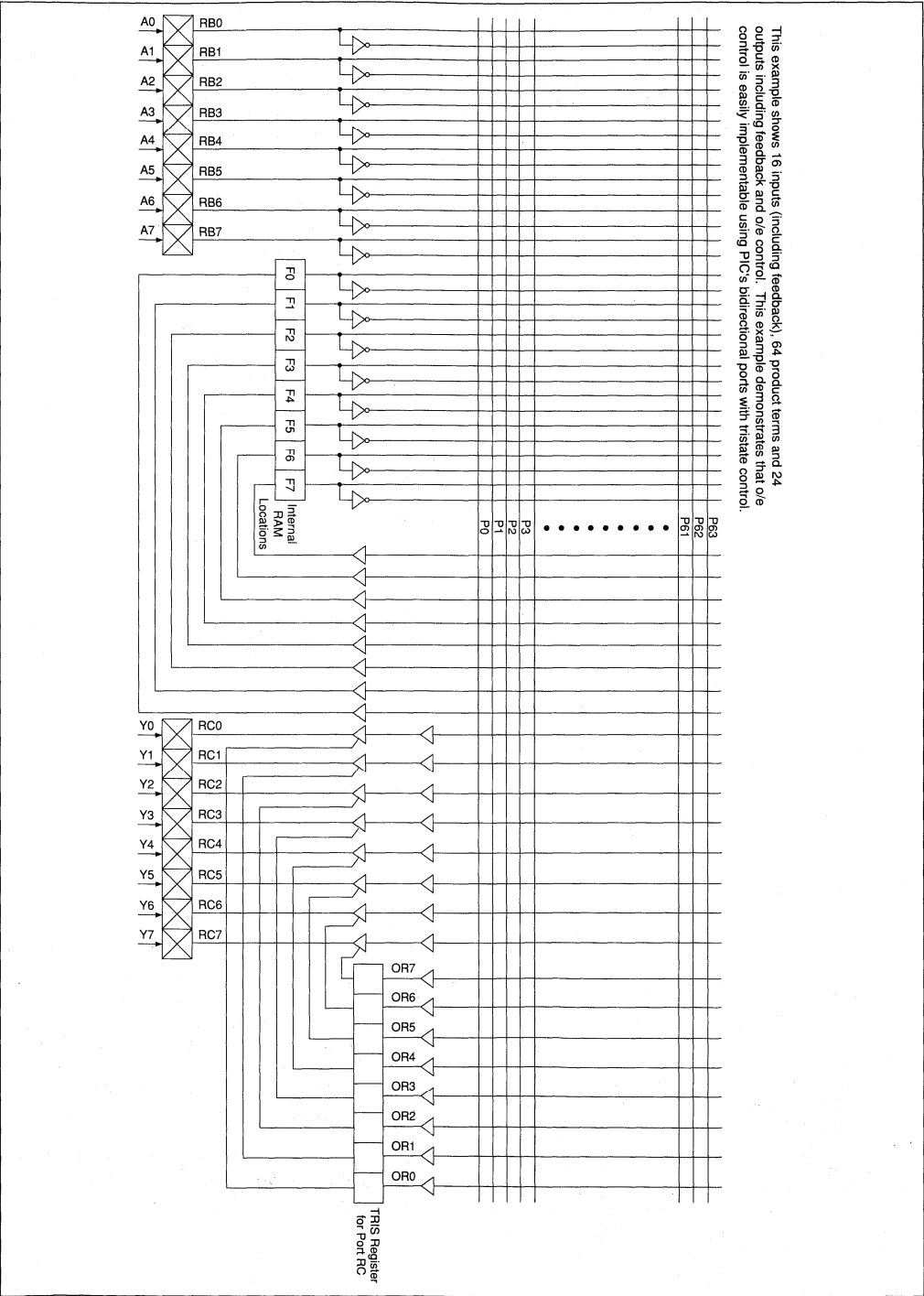
FIGURE 4 - AN ASYNCHRONOUS STATE MACHINE



IMPLEMENTING AN ASYNCHRONOUS STATE MACHINE

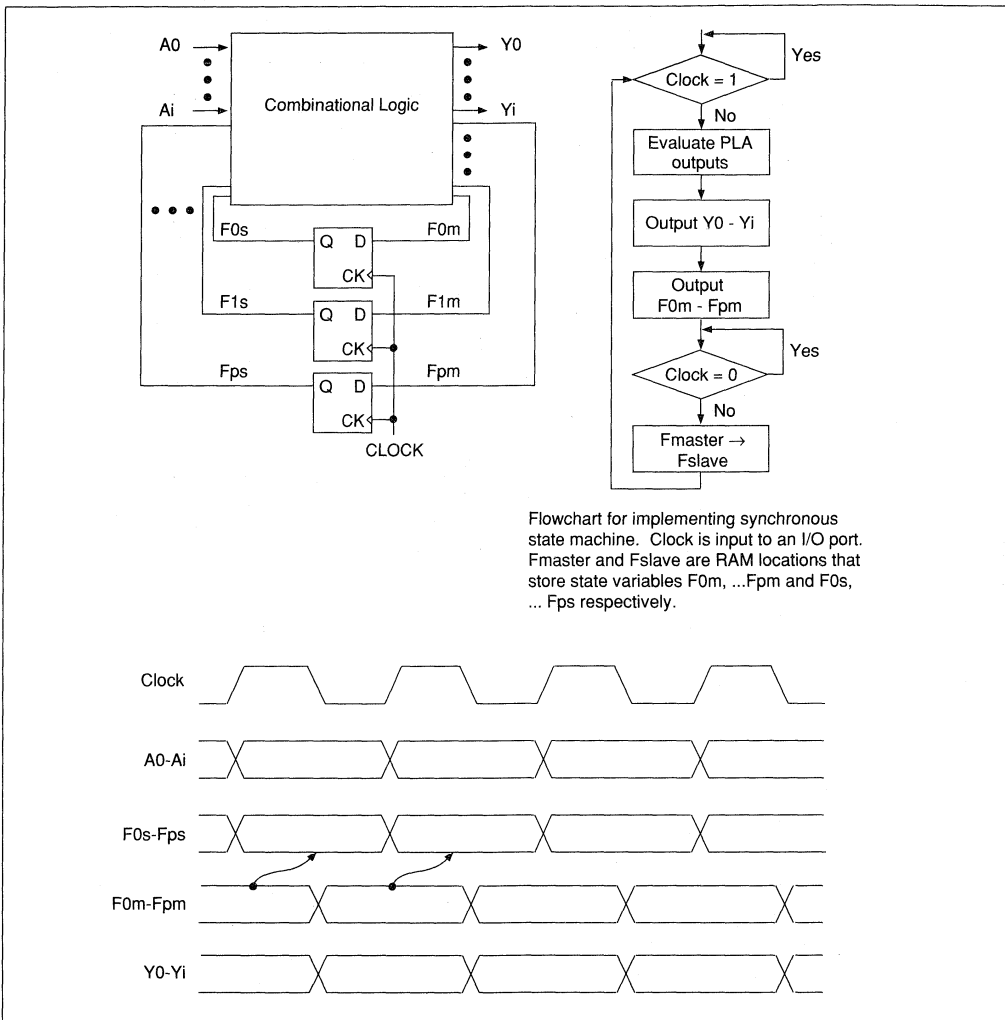
The concept can be easily extended to implement sequential logic i.e. a state machine. Figure 4 shows a state machine with n inputs (A_0 - A_n), m outputs (Y_0 - Y_m) and p states that feedback as inputs to the PLA (F_0 - F_p). In PIC16C5X the states will be stored as bits in RAM location. Input will now mean input from a port as well as from the feedback registers. Figure 5 shows an example PLA with 8 inputs A_0, A_1, \dots, A_7 that are connected to port RB pins RB_0, RB_1, \dots, RB_7 . This example PLA has a total of 24 outputs of which 8 are actual outputs, another 8 are output enable control for the outputs and the other 8 are feedbacks (or states). The PLA shown here, therefore, in essence implements an asynchronous state machine (i.e. there is no system clock).

FIGURE 5 - EXAMPLE OF A LARGER PLA IMPLEMENTATION



PLD Replacement

FIGURE 6 - A SYNCHRONOUS STATE MACHINE IMPLEMENTATION



Flowchart for implementing synchronous state machine. Clock is input to an I/O port. Fmaster and Fslave are RAM locations that store state variables F0m, ...Fpm and F0s, ... Fps respectively.

IMPLEMENTING A SYNCHRONOUS STATE MACHINE

In a synchronous system (see Figure 6) usually all inputs are stable at the falling edge (or rising) edge of the system clock. The state machine samples input on the falling edge, evaluates state and output information. The state outputs are latched by the rising edge of the clock before feeding them back to the input (so that they are stable at the falling edge of the clock). To implement such a state machine, the system clock will have to be polled by an input pin. When a falling edge is detected, the PLA evaluation procedure will be invoked to compute outputs and write them to output pins. The PLA

procedure will also determine the new state variables, F0m, F1m, ..., Fpm and store them in RAM. The program will then wait until a rising edge on the clock input is detected and copy the "master" state variables (F0m, ..., Fpm) to slave state variables (F0s, F1s, ..., Fps). This step emulates the feedback flip-flops.

SUMMARY

In conclusion, the PIC16C5X can implement a generic PLA equation and provide quick, low cost solution where system operation speed is not critical.

Author: *Sumit Mitra*
 Logic Products Division

PLD Replacement

APPENDIX A: PLA IMPLEMENTATION: TIME EFFICIENT APPROACH

MPASM B0.54

PAGE 1

```
*****
; plala.asm :
; This procedure implements a simple AND-OR PLA with:
;
;      8 inputs      := A7 A6 A5 A4 A3 A2 A1 A0
;     24 product terms := P23 P22 ..... P0
;      8 outputs     := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
;
; The eight inputs are assumed to be connected to PORT RB such that
; RB0 = A0, RB1 = A1, ... , RB7 = A7.
; The outputs are programmed to appear on port RC such that
; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
;
; This implementation optimizes both speed & program memory usage
;
*****
; define RAM locations used:
;
LIST      P=16C57
000C      input      equ     d'12'      ;RAM location 12 holds input
000D      Y_reg      equ     d'13'      ;holds output result

000E      Preg_a     equ     d'14'      ;Product terms P0 to P7. Preg_a<0> = P0
000F      Preg_b     equ     d'15'      ;Product terms P8 to P15. Preg_b<0> = P8
0010      Preg_c     equ     d'16'      ;Product terms P16 to P23. Preg_c<0> =
P16

; define some constants and file addresses:
;
0000      bit0       equ     0          ;
0001      bit1       equ     1          ;
0002      bit2       equ     2          ;
0003      bit3       equ     3          ;
0004      bit4       equ     4          ;
0005      bit5       equ     5          ;
0006      bit6       equ     6          ;
0007      bit7       equ     7          ;
;
0003      status     equ     3          ;
0006      port_b     equ     6          ;
0007      port_c     equ     7          ;
;
; define the AND plane programming variables:
;
0000      P0_x       equ     b'00000000' ;
000F      P0_a       equ     b'00001111' ;
0001      P1_x       equ     b'00000001' ;
000F      P1_a       equ     b'00001111' ;
0002      P2_x       equ     b'00000010' ;
000F      P2_a       equ     b'00001111' ;
0003      P3_x       equ     b'00000011' ;
000F      P3_a       equ     b'00001111' ;
0004      P4_x       equ     b'00000100' ;
000F      P4_a       equ     b'00001111' ;
0005      P5_x       equ     b'00000101' ;
000F      P5_a       equ     b'00001111' ;
0006      P6_x       equ     b'00000110' ;
000F      P6_a       equ     b'00001111' ;
0007      P7_x       equ     b'00000111' ;
000F      P7_a       equ     b'00001111' ;
0008      P8_x       equ     b'00001000' ;
000F      P8_a       equ     b'00001111' ;
0009      P9_x       equ     b'00001001' ;
```

2

PLD Replacement

```
000F      P9_a      equ      b'00001111'  ;
000A      P10_x     equ      b'00001010'  ;
000F      P10_a     equ      b'00001111'  ;
000B      P11_x     equ      b'00001011'  ;
000F      P11_a     equ      b'00001111'  ;
000C      P12_x     equ      b'00001100'  ;
000F      P12_a     equ      b'00001111'  ;
000D      P13_x     equ      b'00001101'  ;
000F      P13_a     equ      b'00001111'  ;
000E      P14_x     equ      b'00001110'  ;
000F      P14_a     equ      b'00001111'  ;
000F      P15_x     equ      b'00001111'  ;
000F      P15_a     equ      b'00001111'  ;
0000      P16_x     equ      b'00000000'  ;
0000      P16_a     equ      b'00000000'  ;
0000      P17_x     equ      b'00000000'  ;
0000      P17_a     equ      b'00000000'  ;
0000      P18_x     equ      b'00000000'  ;
0000      P18_a     equ      b'00000000'  ;
0000      P19_x     equ      b'00000000'  ;
0000      P19_a     equ      b'00000000'  ;
0000      P20_x     equ      b'00000000'  ;
0000      P20_a     equ      b'00000000'  ;
0000      P21_x     equ      b'00000000'  ;
0000      P21_a     equ      b'00000000'  ;
0000      P22_x     equ      b'00000000'  ;
0000      P22_a     equ      b'00000000'  ;
0000      P23_x     equ      b'00000000'  ;
0000      P23_a     equ      b'00000000'  ;

      ; define OR plane programming variables:
      x
00ED      OR_a0     equ      b'11101101'  ; for output Y0
00D7      OR_b0     equ      b'11010111'  ;
0000      OR_c0     equ      b'00000000'  ;
009F      OR_a1     equ      b'10011111'  ; for output Y1
0027      OR_b1     equ      b'00100111'  ;
0000      OR_c1     equ      b'00000000'  ;
00FB      OR_a2     equ      b'11111011'  ; for output Y2
002F      OR_b2     equ      b'00101111'  ;
0000      OR_c2     equ      b'00000000'  ;
006D      OR_a3     equ      b'01101101'  ; for output Y3
0079      OR_b3     equ      b'01111001'  ;
0000      OR_c3     equ      b'00000000'  ;
0045      OR_a4     equ      b'01000101'  ; for output Y4
00FD      OR_b4     equ      b'11111101'  ;
0000      OR_c4     equ      b'00000000'  ;
0071      OR_a5     equ      b'01110001'  ; for output Y5
00DF      OR_b5     equ      b'11011111'  ;
0000      OR_c5     equ      b'00000000'  ;
007C      OR_a6     equ      b'01111100'  ; for output Y6
00EF      OR_b6     equ      b'11101111'  ;
0000      OR_c6     equ      b'00000000'  ;
0000      OR_a7     equ      b'00000000'  ; for output Y7
0000      OR_b7     equ      b'00000000'  ;
0000      OR_c7     equ      b'00000000'  ;

      org      01ffh      ;
01FF 0A00      begin      goto      main      ;

      org      000h      ;

      ; define macro to evaluate 1 product (AND) term:
      ;
0000 0902      main      call      pla88      ;
0001 0A00      goto      main      ;
      ;
```



```

EVAL_P MACRO Preg_x,bit_n,Pn_x,Pn_a
    movf    input,W          ;
    xorlw  Pn_x              ;
    andlw  Pn_a              ;
    btfsc status,bit2       ; skip if zero bit not set
    bsf   Preg_x,bit_n      ; product term = 1
    ENDM

; define macro to load OR term constants:
;
EVAL_Y MACRO OR_an,OR_bn,OR_cn,bit_n
    LOCAL SETBIT            ;
    movf   Preg_a,W         ;
    andlw  OR_an            ;
    btfss status,bit2       ;
    goto   SETBIT           ;

    movf   Preg_b,W         ;
    andlw  OR_bn            ;
    btfss status,bit2       ;
    goto   SETBIT           ;

    movf   Preg_c,W         ;
    andlw  OR_cn            ;
    btfss status,bit2       ;
SETBIT   bsf   Y_reg,bit_n  ;
    ENDM

; now the PLA evaluation procedure:
;
0002 0CFF      pla88      movlw   0ffh          ;
0003 0006          tris    6              ; port_b = input
0004 0206          movf   port_b,W       ; read input
0005 002C          movwf  input          ; store input in a register
0006 006E          clrf   Preg_a         ; clear Product register a
0007 006F          clrf   Preg_b         ; clear Product register b
0008 0070          clrf   Preg_c         ; clear Product register c
0009 006D          clrf   Y_reg          ; clear output register

EVAL_P Preg_a,bit0,P0_x,P0_a
000A 020C          movf   input,W         ;
000B 0F00          xorlw  P0_x           ;
000C 0E0F          andlw  P0_a           ;
000D 0643          btfsc status,bit2      ; skip if zero bit not set
000E 050E          bsf   Preg_a,bit0      ; product term = 1

EVAL_P Preg_a,bit1,P1_x,P1_a
000F 020C          movf   input,W         ;
0010 0F01          xorlw  P1_x           ;
0011 0E0F          andlw  P1_a           ;
0012 0643          btfsc status,bit2      ; skip if zero bit not set
0013 052E          bsf   Preg_a,bit1      ; product term = 1

EVAL_P Preg_a,bit2,P2_x,P2_a
0014 020C          movf   input,W         ;
0015 0F02          xorlw  P2_x           ;
0016 0E0F          andlw  P2_a           ;
0017 0643          btfsc status,bit2      ; skip if zero bit not set
0018 054E          bsf   Preg_a,bit2      ; product term = 1

EVAL_P Preg_a,bit3,P3_x,P3_a
0019 020C          movf   input,W         ;
001A 0F03          xorlw  P3_x           ;
001B 0E0F          andlw  P3_a           ;
001C 0643          btfsc status,bit2      ; skip if zero bit not set
001D 056E          bsf   Preg_a,bit3      ; product term = 1

```

PLD Replacement

```
001E 020C          EVAL_P Preg_a,bit4,P4_x,P4_a
001F 0F04          movf   input,W      ;
0020 0E0F          xorlw  P4_x         ;
0021 0643          andlw  P4_a         ;
0022 058E          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_a,bit4  ; product term = 1

0023 020C          EVAL_P Preg_a,bit5,P5_x,P5_a
0024 0F05          movf   input,W      ;
0025 0E0F          xorlw  P5_x         ;
0026 0643          andlw  P5_a         ;
0027 05AE          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_a,bit5  ; product term = 1

0028 020C          EVAL_P Preg_a,bit6,P6_x,P6_a
0029 0F06          movf   input,W      ;
002A 0E0F          xorlw  P6_x         ;
002B 0643          andlw  P6_a         ;
002C 05CE          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_a,bit6  ; product term = 1

002D 020C          EVAL_P Preg_a,bit7,P7_x,P7_a
002E 0F07          movf   input,W      ;
002F 0E0F          xorlw  P7_x         ;
0030 0643          andlw  P7_a         ;
0031 05EE          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_a,bit7  ; product term = 1

0032 020C          EVAL_P Preg_b,bit0,P8_x,P8_a
0033 0F08          movf   input,W      ;
0034 0E0F          xorlw  P8_x         ;
0035 0643          andlw  P8_a         ;
0036 050F          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_b,bit0  ; product term = 1

0037 020C          EVAL_P Preg_b,bit1,P9_x,P9_a
0038 0F09          movf   input,W      ;
0039 0E0F          xorlw  P9_x         ;
003A 0643          andlw  P9_a         ;
003B 052F          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_b,bit1  ; product term = 1

003C 020C          EVAL_P Preg_b,bit2,P10_x,P10_a
003D 0F0A          movf   input,W      ;
003E 0E0F          xorlw  P10_x        ;
003F 0643          andlw  P10_a        ;
0040 054F          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_b,bit2  ; product term = 1

0041 020C          EVAL_P Preg_b,bit3,P11_x,P11_a
0042 0F0B          movf   input,W      ;
0043 0E0F          xorlw  P11_x        ;
0044 0643          andlw  P11_a        ;
0045 056F          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_b,bit3  ; product term = 1

0046 020C          EVAL_P Preg_b,bit4,P12_x,P12_a
0047 0F0C          movf   input,W      ;
0048 0E0F          xorlw  P12_x        ;
0049 0643          andlw  P12_a        ;
004A 058F          btfsc status,bit2  ; skip if zero bit not set
                   bsf   Preg_b,bit4  ; product term = 1

004B 020C          EVAL_P Preg_b,bit5,P13_x,P13_a
004C 0F0D          movf   input,W      ;
004D 0E0F          xorlw  P13_x        ;
004E 0643          andlw  P13_a        ;
                   btfsc status,bit2  ; skip if zero bit not set
```

PLD Replacement

```
004F 05AF          bsf    Preg_b,bit5    ; product term = 1

EVAL_P Preg_b,bit6,P14_x,P14_a
0050 020C          movf   input,W        ;
0051 0F0E          xorlw  P14_x          ;
0052 0E0F          andlw  P14_a          ;
0053 0643          btfs   status,bit2   ; skip if zero bit not set
0054 05CF          bsf    Preg_b,bit6    ; product term = 1

EVAL_P Preg_b,bit7,P15_x,P15_a
0055 020C          movf   input,W        ;
0056 0F0F          xorlw  P15_x          ;
0057 0E0F          andlw  P15_a          ;
0058 0643          btfs   status,bit2   ; skip if zero bit not set
0059 05EF          bsf    Preg_b,bit7    ; product term = 1

EVAL_P Preg_c,bit0,P16_x,P16_a
005A 020C          movf   input,W        ;
005B 0F00          xorlw  P16_x          ;
005C 0E00          andlw  P16_a          ;
005D 0643          btfs   status,bit2   ; skip if zero bit not set
005E 0510          bsf    Preg_c,bit0    ; product term = 1

EVAL_P Preg_c,bit1,P17_x,P17_a
005F 020C          movf   input,W        ;
0060 0F00          xorlw  P17_x          ;
0061 0E00          andlw  P17_a          ;
0062 0643          btfs   status,bit2   ; skip if zero bit not set
0063 0530          bsf    Preg_c,bit1    ; product term = 1

EVAL_P Preg_c,bit2,P18_x,P18_a
0064 020C          movf   input,W        ;
0065 0F00          xorlw  P18_x          ;
0066 0E00          andlw  P18_a          ;
0067 0643          btfs   status,bit2   ; skip if zero bit not set
0068 0550          bsf    Preg_c,bit2    ; product term = 1

EVAL_P Preg_c,bit3,P19_x,P19_a
0069 020C          movf   input,W        ;
006A 0F00          xorlw  P19_x          ;
006B 0E00          andlw  P19_a          ;
006C 0643          btfs   status,bit2   ; skip if zero bit not set
006D 0570          bsf    Preg_c,bit3    ; product term = 1

EVAL_P Preg_c,bit4,P20_x,P20_a
006E 020C          movf   input,W        ;
006F 0F00          xorlw  P20_x          ;
0070 0E00          andlw  P20_a          ;
0071 0643          btfs   status,bit2   ; skip if zero bit not set
0072 0590          bsf    Preg_c,bit4    ; product term = 1

EVAL_P Preg_c,bit5,P21_x,P21_a
0073 020C          movf   input,W        ;
0074 0F00          xorlw  P21_x          ;
0075 0E00          andlw  P21_a          ;
0076 0643          btfs   status,bit2   ; skip if zero bit not set
0077 05B0          bsf    Preg_c,bit5    ; product term = 1

EVAL_P Preg_c,bit6,P22_x,P22_a
0078 020C          movf   input,W        ;
0079 0F00          xorlw  P22_x          ;
007A 0E00          andlw  P22_a          ;
007B 0643          btfs   status,bit2   ; skip if zero bit not set
007C 05D0          bsf    Preg_c,bit6    ; product term = 1

EVAL_P Preg_c,bit7,P23_x,P23_a
007D 020C          movf   input,W        ;
007E 0F00          xorlw  P23_x          ;
007F 0E00          andlw  P23_a          ;
```

PLD Replacement

```
0080 0643          btfscl status,bit2      ; skip if zero bit not set
0081 05F0          bsf      Preg_c,bit7    ; product term = 1

          or_pl          EVAL_Y  OR_a0,OR_b0,OR_c0,bit0
          LOCAL SETBIT          ;
0082 020E          movf    Preg_a,W        ;
0083 0EED          andlw   OR_a0            ;
0084 0743          btfscl status,bit2    ;
0085 0A8D          goto    SETBIT          ;

          0086 020F          movf    Preg_b,W        ;
          0087 0ED7          andlw   OR_b0            ;
          0088 0743          btfscl status,bit2    ;
          0089 0A8D          goto    SETBIT          ;

          008A 0210          movf    Preg_c,W        ;
          008B 0E00          andlw   OR_c0            ;
          008C 0743          btfscl status,bit2    ;
          008D 050D          SETBIT bsf      Y_reg,bit0    ;

          EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
          LOCAL SETBIT          ;
          008E 020E          movf    Preg_a,W        ;
          008F 0E9F          andlw   OR_a1            ;
          0090 0743          btfscl status,bit2    ;
          0091 0A99          goto    SETBIT          ;

          0092 020F          movf    Preg_b,W        ;
          0093 0E27          andlw   OR_b1            ;
          0094 0743          btfscl status,bit2    ;
          0095 0A99          goto    SETBIT          ;

          0096 0210          movf    Preg_c,W        ;
          0097 0E00          andlw   OR_c1            ;
          0098 0743          btfscl status,bit2    ;
          0099 052D          SETBIT bsf      Y_reg,bit1    ;

          EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
          LOCAL SETBIT          ;
          009A 020E          movf    Preg_a,W        ;
          009B 0EFB          andlw   OR_a2            ;
          009C 0743          btfscl status,bit2    ;
          009D 0AA5          goto    SETBIT          ;

          009E 020F          movf    Preg_b,W        ;
          009F 0E2F          andlw   OR_b2            ;
          00A0 0743          btfscl status,bit2    ;
          00A1 0AA5          goto    SETBIT          ;

          00A2 0210          movf    Preg_c,W        ;
          00A3 0E00          andlw   OR_c2            ;
          00A4 0743          btfscl status,bit2    ;
          00A5 054D          SETBIT bsf      Y_reg,bit2    ;

          EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
          LOCAL SETBIT          ;
          00A6 020E          movf    Preg_a,W        ;
          00A7 0E6D          andlw   OR_a3            ;
          00A8 0743          btfscl status,bit2    ;
          00A9 0AB1          goto    SETBIT          ;

          00AA 020F          movf    Preg_b,W        ;
          00AB 0E79          andlw   OR_b3            ;
          00AC 0743          btfscl status,bit2    ;
          00AD 0AB1          goto    SETBIT          ;

          00AE 0210          movf    Preg_c,W        ;
          00AF 0E00          andlw   OR_c3            ;
```

PLD Replacement

```
00B0 0743          btfss  status,bit2  ;
00B1 056D          SETBIT   bsf      Y_reg,bit3  ;

                                EVAL_Y   OR_a4,OR_b4,OR_c4,bit4
                                LOCAL    SETBIT  ;
00B2 020E          movf   Preg_a,W      ;
00B3 0E45          andlw  OR_a4         ;
00B4 0743          btfss  status,bit2  ;
00B5 0ABD          goto   SETBIT       ;

00B6 020F          movf   Preg_b,W      ;
00B7 0EFD          andlw  OR_b4         ;
00B8 0743          btfss  status,bit2  ;
00B9 0ABD          goto   SETBIT       ;

00BA 0210          movf   Preg_c,W      ;
00BB 0E00          andlw  OR_c4         ;
00BC 0743          btfss  status,bit2  ;
00BD 058D          SETBIT   bsf      Y_reg,bit4  ;

                                EVAL_Y   OR_a5,OR_b5,OR_c5,bit5
                                LOCAL    SETBIT  ;
00BE 020E          movf   Preg_a,W      ;
00BF 0E71          andlw  OR_a5         ;
00C0 0743          btfss  status,bit2  ;
00C1 0AC9          goto   SETBIT       ;

00C2 020F          movf   Preg_b,W      ;
00C3 0EDF          andlw  OR_b5         ;
00C4 0743          btfss  status,bit2  ;
00C5 0AC9          goto   SETBIT       ;

00C6 0210          movf   Preg_c,W      ;
00C7 0E00          andlw  OR_c5         ;
00C8 0743          btfss  status,bit2  ;
00C9 05AD          SETBIT   bsf      Y_reg,bit5  ;

                                EVAL_Y   OR_a6,OR_b6,OR_c6,bit6
                                LOCAL    SETBIT  ;
00CA 020E          movf   Preg_a,W      ;
00CB 0E7C          andlw  OR_a6         ;
00CC 0743          btfss  status,bit2  ;
00CD 0AD5          goto   SETBIT       ;

00CE 020F          movf   Preg_b,W      ;
00CF 0EEF          andlw  OR_b6         ;
00D0 0743          btfss  status,bit2  ;
00D1 0AD5          goto   SETBIT       ;

00D2 0210          movf   Preg_c,W      ;
00D3 0E00          andlw  OR_c6         ;
00D4 0743          btfss  status,bit2  ;
00D5 05CD          SETBIT   bsf      Y_reg,bit6  ;

                                EVAL_Y   OR_a7,OR_b7,OR_c7,bit7
                                LOCAL    SETBIT  ;
00D6 020E          movf   Preg_a,W      ;
00D7 0E00          andlw  OR_a7         ;
00D8 0743          btfss  status,bit2  ;
00D9 0AE1          goto   SETBIT       ;

00DA 020F          movf   Preg_b,W      ;
00DB 0E00          andlw  OR_b7         ;
00DC 0743          btfss  status,bit2  ;
00DD 0AE1          goto   SETBIT       ;

00DE 0210          movf   Preg_c,W      ;
00DF 0E00          andlw  OR_c7         ;
00E0 0743          btfss  status,bit2  ;
```

PLD Replacement

```

00E1 05ED          SETBIT      bsf      Y_reg,bit7      ;

; Y_reg now contains 8 output values:
00E2 0040          wr_out    clrw          ;
00E3 0007          tris      7              ; port_c = output
00E4 020D          movf     Y_reg,W          ;
00E5 0027          movwf   port_c         ; Y_reg -> port_c
00E6 0800          retlw   0              ;

00E7 0000          ZZZ          nop

                                END

```

```

Errors : 0
Warnings : 0

```

APPENDIX B: PLA IMPLEMENTATION: CODE EFFICIENT APPROACH

MPASM B0.54

PAGE 1

```

;*****
; plalb.asm :
; This procedure implements a simple AND-OR PLA with:
;
;      8 inputs      := A7 A6 A5 A4 A3 A2 A1 A0
;      24 product terms := P23 P22 ..... P0
;      8 outputs     := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
;
; The eight inputs are assumed to be connected to PORT RB such that
; RB0 = A0, RB1 = A1, ... , RB7 = A7.
; The outputs are programmed to appear on port RC such that
; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
;
; This implementation optimizes program memory usage over
; speed
;*****
; define RAM locations used:
;
;          LIST      P=16C57
000C      input      equ      d"12"      ; RAM location 12 holds input
000D      Y_reg      equ      d"13"      ; holds output result

000E      Preg_a      equ      d"14"      ; Product terms P0 to P7. Preg_a<0> = P0
000F      Preg_b      equ      d"15"      ; Product terms P8 to P15. Preg_b<0> = P8
0010      Preg_c      equ      d"16"      ; Product terms P16 to P23. Preg_c<0> = P16
0012      Pn_x        equ      d"18"      ;
0013      Pn_a        equ      d"19"      ;
0014      OR_a        equ      d"20"      ;
0015      OR_b        equ      d"21"      ;
0016      OR_c        equ      d"22"      ;

; define some constants and file addresses:
;
0000      bit0        equ      0          ;
0001      bit1        equ      1          ;
0002      bit2        equ      2          ;
0003      bit3        equ      3          ;
0004      bit4        equ      4          ;
0005      bit5        equ      5          ;
0006      bit6        equ      6          ;
0007      bit7        equ      7          ;
;
0003      status      equ      3          ;
0006      port_b      equ      6          ;
0007      port_c      equ      7          ;

```



```

;
; define the AND plane programming variables:
;
0000      P0_x      equ      b"00000000"      ;
000F      P0_a      equ      b"00001111"      ;
0001      P1_x      equ      b"00000001"      ;
000F      P1_a      equ      b"00001111"      ;
0002      P2_x      equ      b"00000010"      ;
000F      P2_a      equ      b"00001111"      ;
0003      P3_x      equ      b"00000011"      ;
000F      P3_a      equ      b"00001111"      ;
0004      P4_x      equ      b"00000100"      ;
000F      P4_a      equ      b"00001111"      ;
0005      P5_x      equ      b"00000101"      ;
000F      P5_a      equ      b"00001111"      ;
0006      P6_x      equ      b"00000110"      ;
000F      P6_a      equ      b"00001111"      ;
0007      P7_x      equ      b"00000111"      ;
000F      P7_a      equ      b"00001111"      ;
0008      P8_x      equ      b"00001000"      ;
000F      P8_a      equ      b"00001111"      ;
0009      P9_x      equ      b"00001001"      ;
000F      P9_a      equ      b"00001111"      ;
000A      P10_x     equ      b"00001010"      ;
000F      P10_a     equ      b"00001111"      ;
000B      P11_x     equ      b"00001011"      ;
000F      P11_a     equ      b"00001111"      ;
000C      P12_x     equ      b"00001100"      ;
000F      P12_a     equ      b"00001111"      ;
000D      P13_x     equ      b"00001101"      ;
000F      P13_a     equ      b"00001111"      ;
000E      P14_x     equ      b"00001110"      ;
000F      P14_a     equ      b"00001111"      ;
000F      P15_x     equ      b"00001111"      ;
000F      P15_a     equ      b"00001111"      ;
0000      P16_x     equ      b"00000000"      ;
0000      P16_a     equ      b"00000000"      ;
0000      P17_x     equ      b"00000000"      ;
0000      P17_a     equ      b"00000000"      ;
0000      P18_x     equ      b"00000000"      ;
0000      P18_a     equ      b"00000000"      ;
0000      P19_x     equ      b"00000000"      ;
0000      P19_a     equ      b"00000000"      ;
0000      P20_x     equ      b"00000000"      ;
0000      P20_a     equ      b"00000000"      ;
0000      P21_x     equ      b"00000000"      ;
0000      P21_a     equ      b"00000000"      ;
0000      P22_x     equ      b"00000000"      ;
0000      P22_a     equ      b"00000000"      ;
0000      P23_x     equ      b"00000000"      ;
0000      P23_a     equ      b"00000000"      ;

; define OR plane programming variables:

00ED      OR_a0     equ      b"11101101"      ; for output Y0
00D7      OR_b0     equ      b"11010111"      ;
0000      OR_c0     equ      b"00000000"      ;
009F      OR_a1     equ      b"10011111"      ; for output Y1
0027      OR_b1     equ      b"00100111"      ;
0000      OR_c1     equ      b"00000000"      ;
00FB      OR_a2     equ      b"11111011"      ; for output Y2
002F      OR_b2     equ      b"00101111"      ;
0000      OR_c2     equ      b"00000000"      ;
006D      OR_a3     equ      b"01101101"      ; for output Y3
0079      OR_b3     equ      b"01111001"      ;
0000      OR_c3     equ      b"00000000"      ;
0045      OR_a4     equ      b"01000101"      ; for output Y4
00FD      OR_b4     equ      b"11111101"      ;
0000      OR_c4     equ      b"00000000"      ;

```

PLD Replacement

```
0071          OR_a5          equ      b"01110001"      ; for output Y5
00DF          OR_b5          equ      b"11011111"      ;
0000          OR_c5          equ      b"00000000"      ;
007C          OR_a6          equ      b"01111100"      ; for output Y6
00EF          OR_b6          equ      b"11101111"      ;
0000          OR_c6          equ      b"00000000"      ;
0000          OR_a7          equ      b"00000000"      ; for output Y7
0000          OR_b7          equ      b"00000000"      ;
0000          OR_c7          equ      b"00000000"      ;

          .org      01ffh          ;
01FF 0A00     begin          goto     main          ;

          .org      000h          ;
; define macro to evaluate 1 product (AND) term:
0000 090B     main          call     pla88         ;
0001 0A00     goto         main          ;
;
EVAL_P MACRO   Preg_x,bit_n,Pn_x,Pn_a
          movf     input,W          ;
          xorlw   Pn_x              ;
          andlw   Pn_a              ;
          btfs   status,bit2        ; skip if zero bit not set
          bsf    Preg_x,bit_n       ; product term = 1
          ENDM

; define macro to load OR term constants:
;
EVAL_Y MACRO   OR_an,OR_bn,OR_cn,bit_n
          movlw   OR_an              ; load constants
          movwf   OR_a              ;
          movlw   OR_bn              ;
          movwf   OR_b              ;
          movlw   OR_cn              ;
          movwf   OR_c              ;
          call    EVAL1             ;
          btfs   status,bit2        ;
          bsf    Y_reg,bit_n        ;
          ENDM

; define procedure to evaluate 1 output (OR) term:
;
0002 020E     EVAL1         movf     Preg_a,W          ;
0003 0174     andwf        OR_a,1          ;

0004 020F     movf        Preg_b,W          ;
0005 0175     andwf        OR_b,1          ;

0006 0210     movf        Preg_c,W          ;
0007 0156     andwf        OR_c,W          ;

0008 0114     iorwf       OR_a,W          ;
0009 0115     iorwf       OR_b,W          ;
000A 0800     retlw       0                ; W = 1 implies Yn = 1

; now the PLA evaluation procedure:
;
000B 0CFF     pla88        movlw   0ffh          ;
000C 0006     tris        6                ; port_b = input
000D 0206     movf        port_b,W         ; read input
000E 002C     movwf       input           ; store input in a register
000F 006E     clrf       Preg_a          ; clear Product register a
0010 006F     clrf       Preg_b          ; clear Product register b
0011 0070     clrf       Preg_c          ; clear Product register c
0012 006D     clrf       Y_reg           ; clear output register
```


PLD Replacement

```
and_pl      EVAL_P  Preg_a,bit0,P0_x,P0_a
0013 020C      movf   input,W      ;
0014 0F00      xorlw  P0_x          ;
0015 0E0F      andlw  P0_a          ;
0016 0643      btfs   status,bit2  ; skip if zero bit not set
0017 050E      bsf    Preg_a,bit0  ; product term = 1

EVAL_P  Preg_a,bit1,P1_x,P1_a
0018 020C      movf   input,W      ;
0019 0F01      xorlw  P1_x          ;
001A 0E0F      andlw  P1_a          ;
001B 0643      btfs   status,bit2  ; skip if zero bit not set
001C 052E      bsf    Preg_a,bit1  ; product term = 1

EVAL_P  Preg_a,bit2,P2_x,P2_a
001D 020C      movf   input,W      ;
001E 0F02      xorlw  P2_x          ;
001F 0E0F      andlw  P2_a          ;
0020 0643      btfs   status,bit2  ; skip if zero bit not set
0021 054E      bsf    Preg_a,bit2  ; product term = 1

EVAL_P  Preg_a,bit3,P3_x,P3_a
0022 020C      movf   input,W      ;
0023 0F03      xorlw  P3_x          ;
0024 0E0F      andlw  P3_a          ;
0025 0643      btfs   status,bit2  ; skip if zero bit not set
0026 056E      bsf    Preg_a,bit3  ; product term = 1

EVAL_P  Preg_a,bit4,P4_x,P4_a
0027 020C      movf   input,W      ;
0028 0F04      xorlw  P4_x          ;
0029 0E0F      andlw  P4_a          ;
002A 0643      btfs   status,bit2  ; skip if zero bit not set
002B 058E      bsf    Preg_a,bit4  ; product term = 1

EVAL_P  Preg_a,bit5,P5_x,P5_a
002C 020C      movf   input,W      ;
002D 0F05      xorlw  P5_x          ;
002E 0E0F      andlw  P5_a          ;
002F 0643      btfs   status,bit2  ; skip if zero bit not set
0030 05AE      bsf    Preg_a,bit5  ; product term = 1

EVAL_P  Preg_a,bit6,P6_x,P6_a
0031 020C      movf   input,W      ;
0032 0F06      xorlw  P6_x          ;
0033 0E0F      andlw  P6_a          ;
0034 0643      btfs   status,bit2  ; skip if zero bit not set
0035 05CE      bsf    Preg_a,bit6  ; product term = 1

EVAL_P  Preg_a,bit7,P7_x,P7_a
0036 020C      movf   input,W      ;
0037 0F07      xorlw  P7_x          ;
0038 0E0F      andlw  P7_a          ;
0039 0643      btfs   status,bit2  ; skip if zero bit not set
003A 05EE      bsf    Preg_a,bit7  ; product term = 1

EVAL_P  Preg_b,bit0,P8_x,P8_a
003B 020C      movf   input,W      ;
003C 0F08      xorlw  P8_x          ;
003D 0E0F      andlw  P8_a          ;
003E 0643      btfs   status,bit2  ; skip if zero bit not set
003F 050F      bsf    Preg_b,bit0  ; product term = 1

EVAL_P  Preg_b,bit1,P9_x,P9_a
0040 020C      movf   input,W      ;
0041 0F09      xorlw  P9_x          ;
0042 0E0F      andlw  P9_a          ;
0043 0643      btfs   status,bit2  ; skip if zero bit not set
0044 052F      bsf    Preg_b,bit1  ; product term = 1
```

PLD Replacement

```

                                EVAL_P Preg_b,bit2,P10_x,P10_a
0045 020C                       movf   input,W           ;
0046 0F0A                       xorlw  P10_x           ;
0047 0E0F                       andlw  P10_a           ;
0048 0643                       btfs   status,bit2    ; skip if zero bit not set
0049 054F                       bsf    Preg_b,bit2    ; product term = 1

                                EVAL_P Preg_b,bit3,P11_x,P11_a
004A 020C                       movf   input,W           ;
004B 0F0B                       xorlw  P11_x           ;
004C 0E0F                       andlw  P11_a           ;
004D 0643                       btfs   status,bit2    ; skip if zero bit not set
004E 056F                       bsf    Preg_b,bit3    ; product term = 1

                                EVAL_P Preg_b,bit4,P12_x,P12_a
004F 020C                       movf   input,W           ;
0050 0F0C                       xorlw  P12_x           ;
0051 0E0F                       andlw  P12_a           ;
0052 0643                       btfs   status,bit2    ; skip if zero bit not set
0053 058F                       bsf    Preg_b,bit4    ; product term = 1

                                EVAL_P Preg_b,bit5,P13_x,P13_a
0054 020C                       movf   input,W           ;
0055 0F0D                       xorlw  P13_x           ;
0056 0E0F                       andlw  P13_a           ;
0057 0643                       btfs   status,bit2    ; skip if zero bit not set
0058 05AF                       bsf    Preg_b,bit5    ; product term = 1

                                EVAL_P Preg_b,bit6,P14_x,P14_a
0059 020C                       movf   input,W           ;
005A 0F0E                       xorlw  P14_x           ;
005B 0E0F                       andlw  P14_a           ;
005C 0643                       btfs   status,bit2    ; skip if zero bit not set
005D 05CF                       bsf    Preg_b,bit6    ; product term = 1

                                EVAL_P Preg_b,bit7,P15_x,P15_a
005E 020C                       movf   input,W           ;
005F 0F0F                       xorlw  P15_x           ;
0060 0E0F                       andlw  P15_a           ;
0061 0643                       btfs   status,bit2    ; skip if zero bit not set
0062 05EF                       bsf    Preg_b,bit7    ; product term = 1

                                EVAL_P Preg_c,bit0,P16_x,P16_a
0063 020C                       movf   input,W           ;
0064 0F00                       xorlw  P16_x           ;
0065 0E00                       andlw  P16_a           ;
0066 0643                       btfs   status,bit2    ; skip if zero bit not set
0067 0510                       bsf    Preg_c,bit0    ; product term = 1

                                EVAL_P Preg_c,bit1,P17_x,P17_a
0068 020C                       movf   input,W           ;
0069 0F00                       xorlw  P17_x           ;
006A 0E00                       andlw  P17_a           ;
006B 0643                       btfs   status,bit2    ; skip if zero bit not set
006C 0530                       bsf    Preg_c,bit1    ; product term = 1

                                EVAL_P Preg_c,bit2,P18_x,P18_a
006D 020C                       movf   input,W           ;
006E 0F00                       xorlw  P18_x           ;
006F 0E00                       andlw  P18_a           ;
0070 0643                       btfs   status,bit2    ; skip if zero bit not set
0071 0550                       bsf    Preg_c,bit2    ; product term = 1

                                EVAL_P Preg_c,bit3,P19_x,P19_a
0072 020C                       movf   input,W           ;
0073 0F00                       xorlw  P19_x           ;
0074 0E00                       andlw  P19_a           ;
0075 0643                       btfs   status,bit2    ; skip if zero bit not set
0076 0570                       bsf    Preg_c,bit3    ; product term = 1

```

```

0077 020C          EVAL_P Preg_c,bit4,P20_x,P20_a
                   movf   input,W          ;
0078 0F00          xorlw  P20_x          ;
0079 0E00          andlw  P20_a          ;
007A 0643          btfsc  status,bit2      ; skip if zero bit not set
007B 0590          bsf    Preg_c,bit4      ; product term = 1

007C 020C          EVAL_P Preg_c,bit5,P21_x,P21_a
                   movf   input,W          ;
007D 0F00          xorlw  P21_x          ;
007E 0E00          andlw  P21_a          ;
007F 0643          btfsc  status,bit2      ; skip if zero bit not set
0080 05B0          bsf    Preg_c,bit5      ; product term = 1

0081 020C          EVAL_P Preg_c,bit6,P22_x,P22_a
                   movf   input,W          ;
0082 0F00          xorlw  P22_x          ;
0083 0E00          andlw  P22_a          ;
0084 0643          btfsc  status,bit2      ; skip if zero bit not set
0085 05D0          bsf    Preg_c,bit6      ; product term = 1

0086 020C          EVAL_P Preg_c,bit7,P23_x,P23_a
                   movf   input,W          ;
0087 0F00          xorlw  P23_x          ;
0088 0E00          andlw  P23_a          ;
0089 0643          btfsc  status,bit2      ; skip if zero bit not set
008A 05F0          bsf    Preg_c,bit7      ; product term = 1

008B 0CED          or_pl  EVAL_Y OR_a0,OR_b0,OR_c0,bit0
                   movlw  OR_a0          ; load constants
008C 0034          movwf  OR_a          ;
008D 0CD7          movlw  OR_b0          ;
008E 0035          movwf  OR_b          ;
008F 0C00          movlw  OR_c0          ;
0090 0036          movwf  OR_c          ;
0091 0902          call  EVAL1          ;
0092 0743          btfss  status,bit2      ;
0093 050D          bsf    Y_reg,bit0      ;

0094 0C9F          EVAL_Y OR_a1,OR_b1,OR_c1,bit1
                   movlw  OR_a1          ; load constants
0095 0034          movwf  OR_a          ;
0096 0C27          movlw  OR_b1          ;
0097 0035          movwf  OR_b          ;
0098 0C00          movlw  OR_c1          ;
0099 0036          movwf  OR_c          ;
009A 0902          call  EVAL1          ;
009B 0743          btfss  status,bit2      ;
009C 052D          bsf    Y_reg,bit1      ;

009D 0CFB          EVAL_Y OR_a2,OR_b2,OR_c2,bit2
                   movlw  OR_a2          ; load constants
009E 0034          movwf  OR_a          ;
009F 0C2F          movlw  OR_b2          ;
00A0 0035          movwf  OR_b          ;
00A1 0C00          movlw  OR_c2          ;
00A2 0036          movwf  OR_c          ;
00A3 0902          call  EVAL1          ;
00A4 0743          btfss  status,bit2      ;

```

PLD Replacement

```
00A5 054D          bsf      Y_reg,bit2      ;
                    EVAL_Y OR_a3,OR_b3,OR_c3,bit3
00A6 0C6D          movlw   OR_a3            ; load constants
00A7 0034          movwf  OR_a             ;
00A8 0C79          movlw   OR_b3          ;
00A9 0035          movwf  OR_b             ;
00AA 0C00          movlw   OR_c3          ;
00AB 0036          movwf  OR_c             ;
00AC 0902          call   EVAL1           ;
00AD 0743          btfss  status,bit2     ;
00AE 056D          bsf      Y_reg,bit3     ;
                    EVAL_Y OR_a4,OR_b4,OR_c4,bit4
00AF 0C45          movlw   OR_a4          ; load constants
00B0 0034          movwf  OR_a             ;
00B1 0CFD          movlw   OR_b4          ;
00B2 0035          movwf  OR_b             ;
00B3 0C00          movlw   OR_c4          ;
00B4 0036          movwf  OR_c             ;
00B5 0902          call   EVAL1           ;
00B6 0743          btfss  status,bit2     ;
00B7 058D          bsf      Y_reg,bit4     ;
                    EVAL_Y OR_a5,OR_b5,OR_c5,bit5
00B8 0C71          movlw   OR_a5          ; load constants
00B9 0034          movwf  OR_a             ;
00BA 0CDF          movlw   OR_b5          ;
00BB 0035          movwf  OR_b             ;
00BC 0C00          movlw   OR_c5          ;
00BD 0036          movwf  OR_c             ;
00BE 0902          call   EVAL1           ;
00BF 0743          btfss  status,bit2     ;
00C0 05AD          bsf      Y_reg,bit5     ;
                    EVAL_Y OR_a6,OR_b6,OR_c6,bit6
00C1 0C7C          movlw   OR_a6          ; load constants
00C2 0034          movwf  OR_a             ;
00C3 0CEF          movlw   OR_b6          ;
00C4 0035          movwf  OR_b             ;
00C5 0C00          movlw   OR_c6          ;
00C6 0036          movwf  OR_c             ;
00C7 0902          call   EVAL1           ;
00C8 0743          btfss  status,bit2     ;
00C9 05CD          bsf      Y_reg,bit6     ;
                    EVAL_Y OR_a7,OR_b7,OR_c7,bit7
00CA 0C00          movlw   OR_a7          ; load constants
00CB 0034          movwf  OR_a             ;
00CC 0C00          movlw   OR_b7          ;
00CD 0035          movwf  OR_b             ;
00CE 0C00          movlw   OR_c7          ;
00CF 0036          movwf  OR_c             ;
00D0 0902          call   EVAL1           ;
00D1 0743          btfss  status,bit2     ;
00D2 05ED          bsf      Y_reg,bit7     ;
                    ; Y_reg now contains 8 output values:
00D3 0040          wr_out  clrw            ;
00D4 0007          tris   7               ; port_c = output
00D5 020D          movf   Y_reg,W         ;
00D6 0027          movwf  port_c          ; Y_reg -> port_c
00D7 0800          retlw  0               ;
00D8 0000          ZZZ      nop
                    END
Errors   : 0
Warnings : 0
```

Analog to Digital Conversion

INTRODUCTION

This application note describes a method for implementing analog to digital conversion on the PIC16C5X series of microcontrollers. The converter requires only four external components and is software and hardware configurable for conversion resolutions from 6 bits up to 10 bits and conversion times of 250us or longer. The method is useable for both voltage and current conversion and uses a software calibration technique that compensates for time and temperature drift as well as component errors. The PIC16C5X microcontrollers are ideal for simple analog applications because:

- * Very low cost.
- * Few external components required.
- * Fully programmable. PIC16C5X microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
- * Available off the shelf from distributors.
- * Calibration in software for improved measurement accuracy.
- * Power savings using PIC16C5X's SLEEP mode.
- * PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

The application uses a capacitive charging circuit (see Figure 1) to convert the input voltage to time, which can be easily measured using a microcontroller. First, the reference voltage is applied to the input voltage to current converter (U1). The equivalent circuit is shown in Figure 2. This circuit provides a linearly variable current as a function of input voltage. The logarithmic characteristic that would occur if the input voltage was applied directly to an RC is not present. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the resistor and capacitor, changes in the input threshold voltage and temperature variations. After the software calibration value is measured, the capacitor is discharged (see Figure 3) and the input voltage is connected to V_{in} . The time to trip the threshold is measured for the input voltage and compared to the calibration value to determine the actual input voltage.

FIGURE 1 - VOLTMETER A TO D CONVERTER

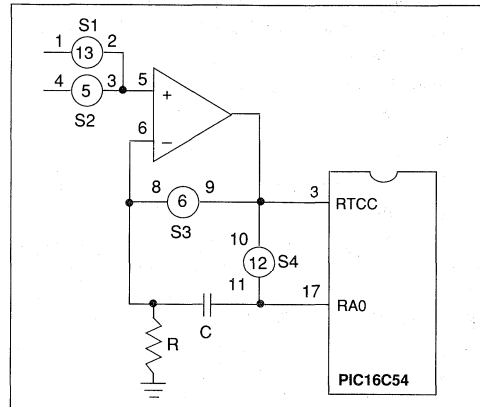


FIGURE 2 - VOLTMETER MEASUREMENT CYCLE

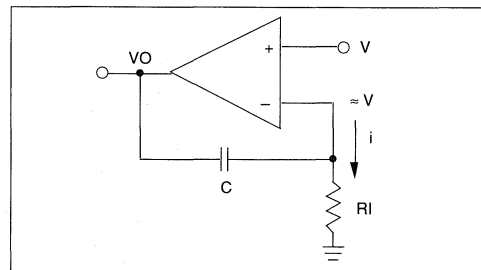
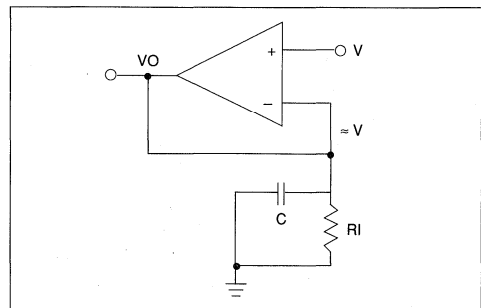


FIGURE 3 - VOLTMETER DISCHARGE CYCLE



Analog to Digital Conversion

CIRCUIT CONFIGURATION

The values of R and C are selected based upon the number of bits of resolution required.

$$RC = (V_i \cdot T) / V_t$$

Where:

V_i = Lowest voltage to be measured (at least ten I_{sb} 's)

T = Time to do the number of bits of resolution desired

V_t = Threshold voltage of the PIC16C5X input being used

Actual value for RC should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use a 3 volt input and 8 bits resolution with a 8 MHz clock and 6 instruction cycles per count:

$$V_i = 100 \text{ mV}$$

$$T = 256 \cdot 1/8 \text{ MHz} \cdot 4 \text{ clocks/cycle} \cdot 6 \text{ cycles} = 768 \text{ } \mu\text{s}$$

$$V_t = 3.0V \text{ (est)}$$

For input voltages greater than 3 volts a resistor divider network should be used to keep the maximum voltage on V_{in} to less than 3 Volts. For best performance the reference voltage should be between 2 and 3 volts.

The circuit can also be used as a current mode A to D converter. In this case the input voltage to current converter is not needed and the reference current and input current are both routed via analog switches directly into the capacitor.

CIRCUIT PERFORMANCE

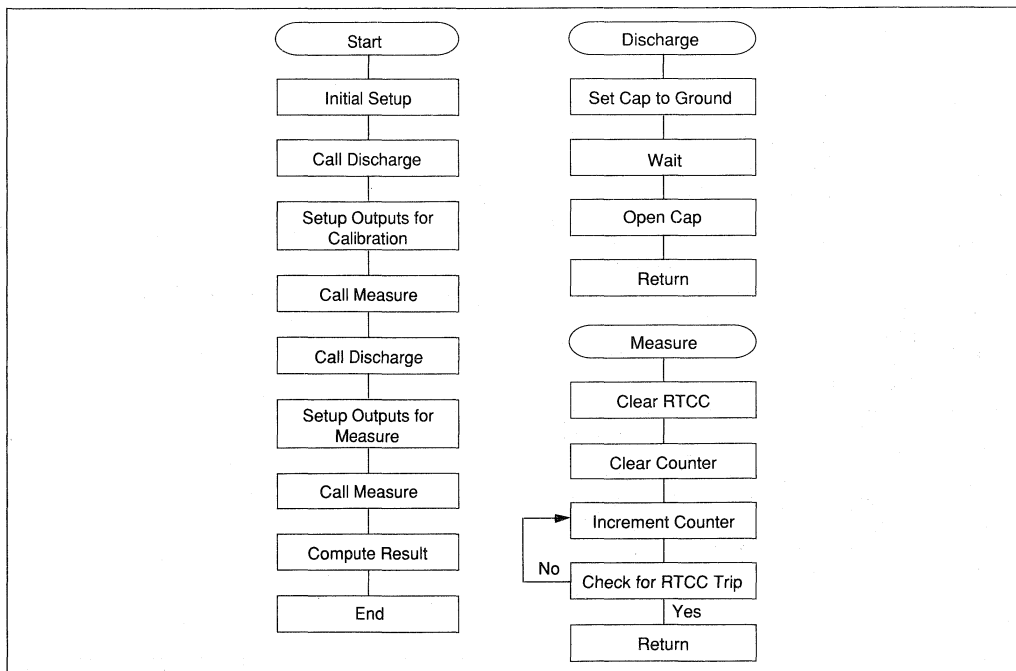
The calibration cycle removes all first order errors (offset, gain, R and C inaccuracy, power supply voltage and temperature) except the reference voltage drift. Any change in the reference voltage, including noise, between the calibration cycle and the measurement cycle may result in measurement errors. Other error sources are analog switch leakage, resistor and capacitor non-linearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the converter to be accurate within +/- 1% of full scale.

Example

Assembly code implementing the circuit of figure 1 is listed in Appendix A: This code measures the time up to 16 bits and calculates the results using 16 bit multiply and divide subroutines. In actual applications, if measurement accuracy permits, it may be advantageous to use 8 bits. The math code can be substantially reduced and the measure time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division

FIGURE 4 - TRANSMISSION FLOW CHART



Analog to Digital Conversion

APPENDIX A:

MPASM B0.54

PAGE 1

VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90

```
TITLE 'VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90'
LIST P=16C54,F=inhx16,n=0

0008          ACCA    EQU    8
000A          ACCB    EQU    0A
000C          ACCC    EQU    0C
000E          ACCD    EQU    0E
0010          ACCE    EQU    10
0012          TMEAS   EQU    12
0014          TEMP    EQU    14

0060          VCALMS  EQU    60          ;VCAL MSB VALUE IN HEX
00A4          VCALLS  EQU    0A4        ;VCAL LSB VALUE IN HEX

          ORG 1FF
01FF 0A58     GOTO    VOLTS          ;PROGRAM CODE
          ORG    0                  ;SUBROUTINES

0000 0209     MADD    MOVF    ACCA+1,W
0001 01EB     ADDWF   ACCB+1          ;ADD LSB
0002 0603     BTFSC  3,0            ;ADD IN CARRY
0003 02AA     INCF   ACCB
0004 0208     MOVF   ACCA,W
0005 01EA     ADDWF  ACCB          ;ADD MSB
0006 0800     RETLW  0
0007 0000     NOP

0008 0915     MPY    CALL    SETUP    ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E     MLOOP  RRF    ACCD      ;ROTATE D RIGHT
000A 032F     RRF    ACCD+1
000B 0603     SKPNC                ;NEED TO ADD?
000C 0900     CALL   MADD
000D 032A     RRF    ACCB
000E 032B     RRF    ACCB+1
000F 032C     RRF    ACCC
0010 032D     RRF    ACCC+1
0011 02F4     DECFSZ TEMP          ;LOOP UNTIL ALL BITS CHECKED
0012 0A09     GOTO   MLOOP
0013 0800     RETLW  0

0014 0000     NOP
0015 0C10     SETUP  MOVLW  10
0016 0034     MOVWF  TEMP
0017 020A     MOVF   ACCB,W          ;MOVE B TO D
0018 002E     MOVWF  ACCD
0019 020B     MOVF   ACCB+1,W
001A 002F     MOVWF  ACCD+1
001B 020C     MOVF   ACCC,W
001C 0030     MOVWF  ACCE
001D 020D     MOVF   ACCC+1,W
001E 0031     MOVWF  ACCE+1
001F 006A     CLRF  ACCB
0020 006B     CLRF  ACCB+1
0021 0800     RETLW  0

0022 0000     NOP
0023 0915     DIV    CALL    SETUP
0024 0C20     MOVLW  20
0025 0034     MOVWF  TEMP
0026 006C     CLRF  ACCC
0027 006D     CLRF  ACCC+1
```

Analog to Digital Conversion

```

0028 0403      DLOOP      CLRC
0029 0371      RLF        ACCE+1
002A 0370      RLF        ACCE
002B 036F      RLF        ACCD+1
002C 036E      RLF        ACCD
002D 036D      RLF        ACCC+1
002E 036C      RLF        ACCC
002F 0208      MOVF       ACCA,W
0030 008C      SUBWF     ACCC,W           ;CHECK IF A>C
0031 0743      SKPZ
0032 0A35      GOTO     NOCHK
0033 0209      MOVF       ACCA+1,W
0034 008D      SUBWF     ACCC+1,W       ;IF MSB EQUAL THEN CHECK LSB
0035 0703      NOCHK     SKPC           ;CARRY SET IF C>A
0036 0A3E      GOTO     NOGO
0037 0209      MOVF       ACCA+1,W       ;C-A INTO C
0038 00AD      SUBWF     ACCC+1
0039 0703      BTFSS    3,0
003A 00EC      DECF     ACCC
003B 0208      MOVF       ACCA,W
003C 00AC      SUBWF     ACCC
003D 0503      SETC           ;SHIFT A 1 INTO B (RESULT)
003E 036B      NOGO     RLF        ACCB+1
003F 036A      RLF        ACCB
0040 02F4      DECF     TEMP           ;LOOP UNTILL ALL BITS CHECKED
0041 0A28      GOTO     DLOOP
0042 0800      RETLW    0

0043 0C0E      DSCHRG   MOVLW   B'00001110'   ;DISCHARGE C (RA0 ON)
0044 0005      TRIS    5
0045 0CFF      MOVLW   OFF
0046 0034      MOVWF   TEMP
0047 02F4      LOOP    DECF     TEMP           ;WAIT
0048 0A47      GOTO     LOOP
0049 0C0F      MOVLW   B'00001111'   ;ALL RA HIGH Z
004A 0005      TRIS    5
004B 0800      RETLW    0

004C 0061      M_TIME  CLRF     1           ;CLEAR RTCC REGISTER
004D 0069      CLRF     ACCA+1       ;CLEAR 16 BIT COUNTER
004E 0068      CLRF     ACCA
004F 03E9      TLOOP   INCF     ACCA+1
0050 0A54      GOTO     ENDCHK
0051 03E8      INCF     ACCA
0052 0A54      GOTO     ENDCHK
0053 0A56      GOTO     END_M
0054 0701      ENDCHK  BTFSS    1,0       ;CHECK FOR RTCC TRIP
0055 0A4F      GOTO     TLOOP
0056 0201      END_M   MOVF     1,W
0057 0800      RETLW    0

0058 0C06      VOLTS   MOVLW   B'00000110'   ;SET S2 AND S3 HIGH(ON WHEN ACTIVATED)
0059 0026      MOVWF   6
005A 0CF0      MOVLW   B'11110000'   ;ACTIVATE SWITCHES S1-S4
005B 0006      TRIS    6
005C 0C28      MOVLW   B'00101000'   ;SELECT POSITIVE EDGE FOR RTCC
005D 0002      OPTION
005E 0C00      MOVLW   B'00000000'
005F 0025      MOVWF   5           ;SET RA0 LOW (ON WHEN ACTIVATED)

0060 0943      MEAS    CALL     DSCHRG       ;CHARGE CAPACITOR TO VIN
0061 0C0A      MOVLW   B'00001010'   ;S2 AND S4 ON
0062 0026      MOVWF   6
0063 094C      CALL     M_TIME       ;MEASURE TIME
0064 0209      MOVF     ACCA+1,W
0065 0033      MOVWF   TMEAS+1       ;STORE LSB
0066 0208      MOVF     ACCA,W
0067 0032      MOVWF   TMEAS         ;STORE MSB

```


Analog to Digital Conversion

```
0126 064      209      MOVF  ACCA+1,W
0127 065      033      MOVWF TMEAS+1      ;STORE LSB
0128 066      208      MOVF  ACCA,W
0129 067      032      MOVWF TMEAS      ;STORE MSB
0130
0131 068      C05      CAL   MOVLW B'00000101'      ;S1 AND S3 ON
0132 069      026      MOVWF 6
0133 06A      943      CALL  DSCHRG      ;CHARGE CAPACITOR TO VREF
0134 06B      C09      MOVLW B'00001001'      ;S1 AND S4 ON
0135 06C      026      MOVWF 6
0136 06D      94C      CALL  M_TIME      ;MEASURE TIME
0137
0138 06E      CA4      MOVLW VCALLS
0139 06F      02B      MOVWF ACCB+1
0140 070      C60      MOVLW VCALMS
0141 071      02A      MOVWF ACCB
0142
0143 072      908      CALL  MPY      ;MULTIPLY ACCA(TCAL) * ACCB(VREF)
0144 073      213      MOVF  TMEAS+1,W
0145 074      029      MOVWF ACCA+1
0146 075      212      MOVF  TMEAS,W
0147 076      028      MOVWF ACCA
0148
0149 077      923      CALL  DIV      ;DIVIDE ACCB(TCAL *
0149      ;V) BY ACCA(TMEAS) 0150
0151 078      A58      GOTO  VOLTS
0152
0153      END
%ASM-I, No Errors, No Warnings
```

Analog to Digital Conversion

NOTES:

Implementing Ohmmeter/Temperature Sensor

INTRODUCTION

This application note describes a method for implementing an ohmmeter or resistance type temperature sensor using the PIC16C5X series of microcontrollers. The ohmmeter requires only two external components and is software and hardware configurable for resistance measurement with resolutions from 6 bits up to 10 bits with measurement times of 250μs (6 bits at 8 MHz) or longer. The method uses a software calibration technique that compensates for voltage, time, and temperature drift as well as component errors. The PIC16C5X Microcontrollers are ideal for simple analog applications because:

- * Very low cost.
- * Few external components required.
- * Fully programmable. PIC16C5X Microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
- * Available off the shelf from distributors.
- * Calibration in software for improved measurement accuracy.
- * Power savings using PIC16C5X's SLEEP mode.
- * PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

The application uses a capacitive charging circuit (Figure 1) to convert the resistance to time, which can be easily measured using a microcontroller. First, a reference voltage (usually VDD) is applied to a calibration resistor, Rc. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the capacitor, changes in the input threshold voltage and temperature variations. After C is discharged, the reference voltage is applied to the resistance to be measured (or thermistor). The time to trip the threshold is then measured and compared to the calibration value to determine the actual resistance (Figure 2). In the temperature sensing mode, the temperature is calculated using a lookup table.

FIGURE 1 - OHMMETER/TEMPERATURE SENSOR

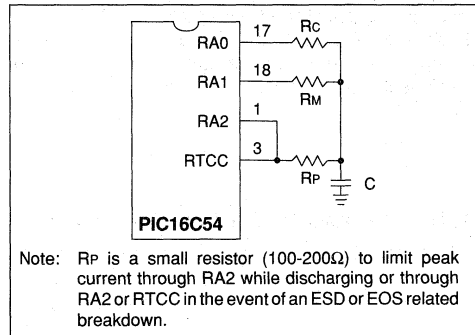
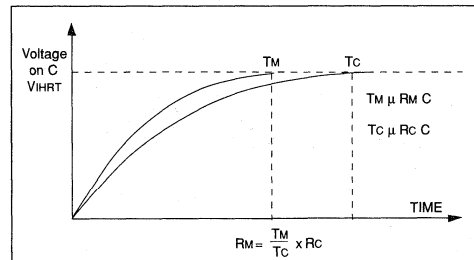


FIGURE 2 - OHMMETER/TEMPERATURE SENSOR



CIRCUIT CONFIGURATION

The values of Rc and C are selected based upon the number of bits of resolution required. Rc should be approximately one half the largest value resistance to be measured and:

$$C = \frac{-T}{R_m \cdot \ln \left(1 - \frac{V_t}{V_r} \right)}$$

Where:

- Vr = Reference voltage
- T = Time to do the number of bits of resolution desired
- Vt = Threshold voltage of the PIC input being used
- Rm = Maximum resistance value to be measured

Actual value for C should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

Ohmmeter/Temperature Sensor

For example use $R_m=200K$ for 8 bits resolution with a 8 MHz clock, $V_r = 5V$, $V_t = 3V$, $R_c = 100K$ and 6 instruction cycles per count:

$$T = 256 \text{ counts} * 1/8 \text{ MHz} * 4 \text{ clocks/instruction} * 6 \text{ instructions/count} = 768 \mu\text{s}$$

$$C = 4200 \text{ pF [Use } 3900 \text{ pF]}$$

CIRCUIT PERFORMANCE

The calibration cycle removes all first order errors (offset, gain, C inaccuracy, power supply voltage and temperature) except R absolute accuracy. A low drift resistor should be selected for R and its value stored in software to reduce measurement errors. Other error sources are I/O pin leakage, resistor and capacitor non-linearities,

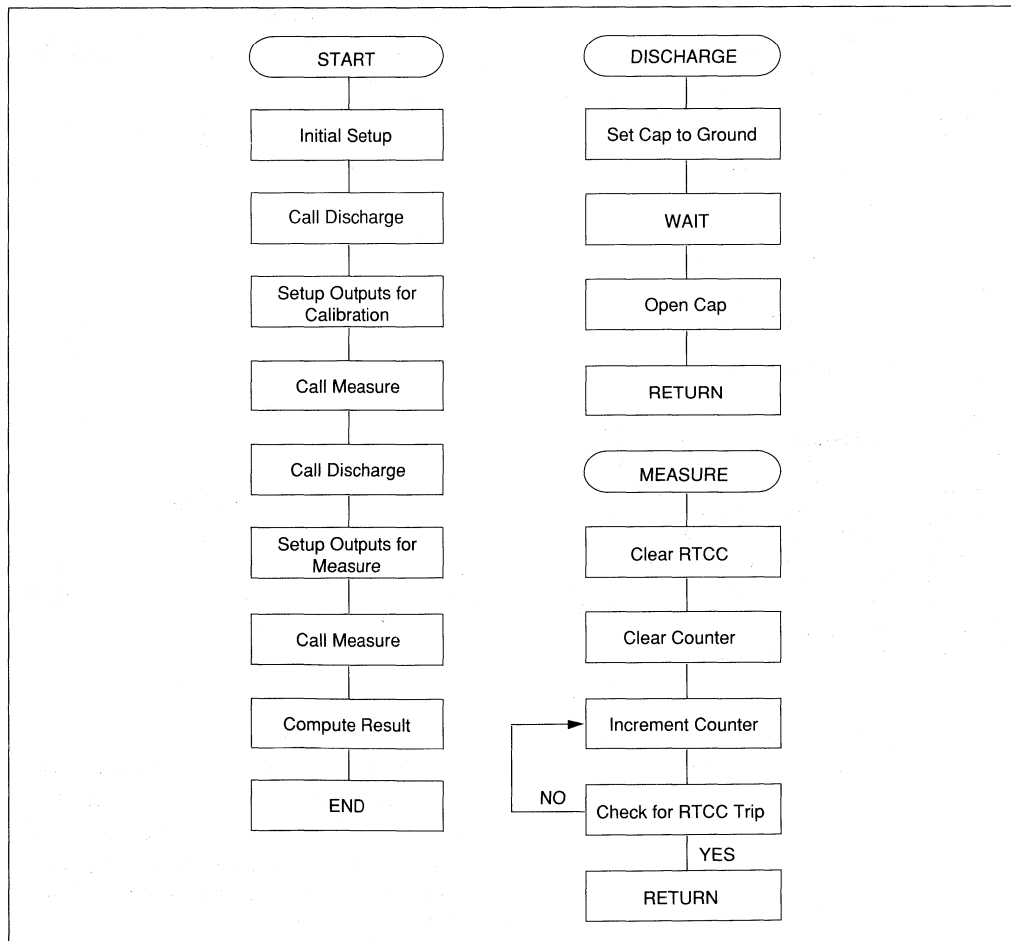
input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the ohmmeter to be accurate within +/- 1% over one decade.

Example

The assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures time up to 16 bits (65535 measure cycles) and calculates the results using 16 bit multiply and divide subroutines. In actual applications, it is more efficient to use 8 bit measurements if application accuracies permit. The math code will be substantially reduced and measurement time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division

FIGURE 3 - TRANSMISSION FLOW CHART



Ohmeter/Temperature Sensor

APPENDIX A

MPASM B0.54

PAGE 1

```
LIST P=16C54,F=inhx8M

0008      ACCA EQU 8
000A      ACCB EQU 0A
000C      ACCC EQU 0C
000E      ACCD EQU 0E
0010      ACCE EQU 10
0012      TCAL EQU 12
0014      TEMP EQU 14

002F      RCALMS EQU 2F      ;RCAL MSB VALUE IN HEX
003C      RCALLS EQU 3C      ;RCAL LSB VALUE IN HEX

                                ORG 1FF
01FF 0A58      GOTO OHMS
                                ORG 0

0000 0209      MADD MOVF ACCA+1,W
0001 01EB      ADDWF ACCB+1      ;ADD LSB
0002 0603      BTFSC 3,0      ;ADD IN CARRY
0003 02AA      INCF ACCB
0004 0208      MOVF ACCA,W
0005 01EA      ADDWF ACCB      ;ADD MSB
0006 0800      RETLW 0
0007 0000      NOP

0008 0915      MPY CALL SETUP      ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E      MLOOP RRF ACCD      ;ROTATE D RIGHT
000A 032F      RRF ACCD+1
000B 0603      SKPNC      ;NEED TO ADD?
000C 0900      CALL MADD
000D 032A      RRF ACCB
000E 032B      RRF ACCB+1
000F 032C      RRF ACCC
0010 032D      RRF ACCC+1
0011 02F4      DECFSZ TEMP      ;LOOP UNTIL ALL BITS CHECKED
0012 0A09      GOTO MLOOP
0013 0800      RETLW 0

0014 0000      NOP
0015 0C10      SETUP MOVLW 10
0016 0034      MOVWF TEMP
0017 020A      MOVF ACCB,W      ;MOVE B TO D
0018 002E      MOVWF ACCD
0019 020B      MOVF ACCB+1,W
001A 002F      MOVWF ACCD+1
001B 020C      MOVF ACCC,W
001C 0030      MOVWF ACCC
001D 020D      MOVF ACCC+1,W
001E 0031      MOVWF ACCE+1
001F 006A      CLRF ACCB
0020 006B      CLRF ACCB+1
0021 0800      RETLW 0
0022 0000      NOP
0023 0915      DIV CALL SETUP
0024 0C20      MOVLW 20
0025 0034      MOVWF TEMP
0026 006C      CLRF ACCC
0027 006D      CLRF ACCC+1
0028 0403      DLOOP CLRC
0029 0371      RLF ACCE+1
002A 0370      RLF ACCE
002B 036F      RLF ACCD+1
```

Ohmeter/Temperature Sensor

```

002C 036E          RLF  ACCD
002D 036D          RLF  ACCC+1
002E 036C          RLF  ACCC
002F 0208          MOVF ACCA,W
0030 008C          SUBWF ACCC,W          ;CHECK IF A>C
0031 0743          SKPZ
0032 0A35          GOTO NOCHK
0033 0209          MOVF ACCA+1,W
0034 008D          SUBWF ACCC+1,W          ;IF MSB EQUAL THEN CHECK LSB
0035 0703          NOCHK SKPC          ;CARRY SET IF C>A
0036 0A3E          GOTO NOGO
0037 0209          MOVF ACCA+1,W          ;C-A INTO C
0038 00AD          SUBWF ACCC+1
0039 0703          BTFSS 3,0
003A 00EC          DECF  ACCC
003B 0208          MOVF ACCA,W
003C 00AC          SUBWF ACCC
003D 0503          SETC          ;SHIFT A 1 INTO B (RESULT)
003E 036B          NOGO  RLF  ACCB+1
003F 036A          RLF  ACCB
0040 02F4          DECFSZ TEMP          ;LOOP UNTILL ALL BITS CHECKED
0041 0A28          GOTO  DLOOP
0042 0800          RETLW 0

0043 0C0B          DSCHRG MOVLW B'00001011'          ;ACTIVATE RA2
0044 0005          TRIS 5
0045 0CFF          MOVLW OFF
0046 0034          MOVWF TEMP
0047 02F4          LOOP  DECFSZ TEMP          ;WAIT
0048 0A47          GOTO  LOOP
0049 0C0F          MOVLW B'00001111'          ;ALL OUTPUTS OFF
004A 0005          TRIS 5
004B 0800          RETLW 0

004C 0061          M_TIME CLR 1          ;CLEAR RTCC
004D 0069          CLR  ACCA+1
004E 0068          CLR  ACCA
004F 03E9          TLOOP INCFSZ ACCA+1
0050 0A54          GOTO  ENDCHK
0051 03E8          INCFSZ ACCA
0052 0A54          GOTO  ENDCHK
0053 0A56          GOTO  END_M
0054 0701          ENDCHK BTFSS 1,0          ;CHECK FOR RTCC TRIP
0055 0A4F          GOTO  TLOOP
0056 0201          END_M MOVF 1,W
0057 0800          RETLW 0

0058 0C03          OHMS  MOVLW B'00000011'          ;SET RAO AND RAI HIGH (ON WHEN ACTIVATED)
0059 0025          MOVWF 5
005A 0C28          MOVLW B'00101000'          ;SELECT POSITIVE EDGE FOR RTCC
005B 0002          OPTION

005C 0943          CAL   CALL  DSCHRG          ;DISCHARGE CAPACITOR
005D 0C0E          MOVLW B'00001110'          ;ACTIVATE RAO
005E 0005          TRIS 5
005F 094C          CALL  M_TIME          ;MEASURE TIME
0060 0209          MOVF  ACCA+1,W
0061 0033          MOVWF TCAL+1          ;STORE LSB
0062 0208          MOVF  ACCA,W
0063 0032          MOVWF TCAL          ;STORE MSB

0064 0943          MEAS  CALL  DSCHRG          ;DISCHARGE CAPACITOR
0065 0C0D          MOVLW B'00001101'          ;ACTIVATE RAI
0066 0005          TRIS 5
0067 094C          CALL  M TIME          ;MEASURE TIME

0068 0C3C          MOVLW RCALLS          ;CALIBRATION LSB VALUE
0069 002B          MOVWF ACCB+1
006A 0C2F          MOVLW RCALMS          ;CALIBRATION MSB VALUE

```

Ohmeter/Temperature Sensor

```
006B 002A      MOVWF  ACCB
006C 0908      CALL   MPY           ;MULTIPLY ACCA(MEAS) * ACCB(RCAL)
006D 0213      MOVF   TCAL+1,W
006E 0029      MOVWF  ACCA+1
006F 0212      MOVF   TCAL,W
0070 0028      MOVWF  ACCA
0071 0923      CALL   DIV           ;DIVIDE ACCB(MEAS * R) BY ACCA(TCAL)
0072 0A58      GOTO   OHMS
END
```

```
Errors   : 0
Warnings : 0
```

Ohmeter/Temperature Sensor

NOTES:



Interfacing to AC Power Lines

INTRODUCTION

This application note describes a simple method for measuring parameters from the AC power line. Parameters such as zero crossing, frequency, and relative phase can be measured. The method is useful for measurements on 50, 60, and 400 Hz power systems with voltages up to several hundred volts. The method requires only one external component, a resistor, and is more reliable than previously published methods using capacitors or bulky, expensive transformers.

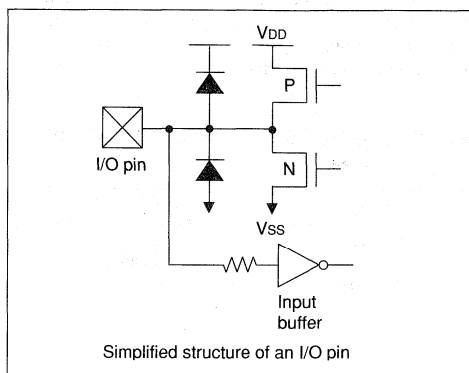
APPLICATIONS

This measurement method can be used in any application where power line parameters are used for system measurements or control. Typical applications are for switch timing (what part of the power cycle should the system be activated), power factor correction, power measurement, and power line monitor. An additional application is to generate timing or clock functions using the relatively stable power line frequency. The method is also useful for calibrating the oscillator frequency for accurate timing measurements when an inaccurate reference such as an RC oscillator is used to clock the PIC16C5X.

THEORY OF OPERATION

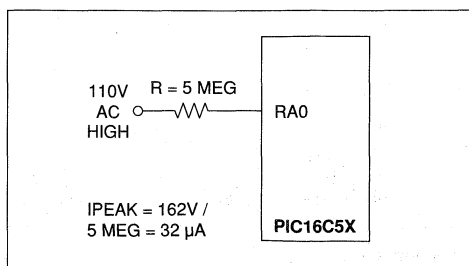
The application takes advantage of the input static protection circuitry that exists on all I/O pins of a CMOS PIC16C5X. These protection circuits are designed to short the inputs to the power supplies when a large overvoltage is applied, thus protecting the chip from static electricity spikes. On the PIC16C5X microcontrollers, this protection circuit is two large P-N diodes on each input (see Figure 1). These diodes will short any voltage higher than V_{DD} to the V_{DD} supply and any voltage less than V_{SS} to the V_{SS} supply. They can take several milliamps of current without any damage to the chip. High voltages can be applied directly to the chip inputs as long as they are current limited.

FIGURE 1 - PIC16C5X SERIES INPUT PROTECTION CIRCUIT ON I/O PINS



The least expensive method of current limiting is using a high value resistor. This method is shown schematically in Figure 2. The power line voltage is current limited by the resistor and then clamped by the input protection diodes internal to the PIC16C5X. A typical input waveform is shown in Figure 3. A 115V AC, 60 cycle sine wave will traverse from 0 to 2 volts in $32 \mu s$ so a typical threshold of 2 volts on the PIC16C5X I/O port will permit zero crossing detection accuracy of about $30 \mu s$. If the typical capacitance on an I/O pin is $5 pF$, then R should be $(T = RC)$ 6 MEGohm or less for best zero crossing accuracy. A 5 MEGohm resistor with 115V AC applied to it will limit current to $32 \mu A$, a value which is well within the safety margin of the PIC16C5X.

FIGURE 2 - CURRENT LIMITING USING AN EXTERNAL RESISTOR



Interfacing to AC Power Lines

The user needs to be aware that the circuit required to connect the RTCC input to AC power line is slightly different. Each of the I/O pins has two diodes for input protection whereas RTCC pin has only one protection diode connecting to Vss (see figure 3). Therefore, it is necessary to connect a diode externally between RTCC pin and VDD in order to clamp the voltage on RTCC pin to VDD + 0.6V (approx.). See Figure 4. It is also recommended that resistor R be at least 2MΩ.

RELIABILITY

Reliability of production devices that are directly connected to AC power is always a concern. Two failure modes are possible. First, the series resistor of Figure 1 might fail short, destroying the microcontroller.

This is the most unlikely failure mode of a resistor, and resistors are more reliable than transformers or capacitors, which are the alternate components for measuring line parameters. This reliability can be enhanced even further by using two resistors in series. Both would have to fail short to cause catastrophic failure, a very unlikely event. The second possible failure mode is that excessive injection of current into the PIC16C5X input might cause the protection diode to open. This would allow the input to go to the power line peak voltage (162V) and short the input transistor gate oxide, causing device failure. The maximum continuous injection current into an I/O pin is specified ±500 μA. An I/O pin is also capable of handling larger injection current (>100 mA) for a very short period (transient) of time. Therefore higher transient currents due to line voltage surges will be easily handled.

FIGURE 3 - INPUT STRUCTURE AND RTCC PIN

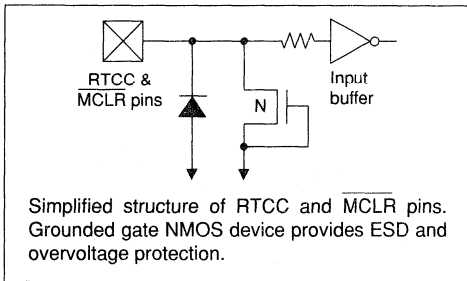


FIGURE 4 - CONNECTING AC POWER LINE TO RTCC PIN

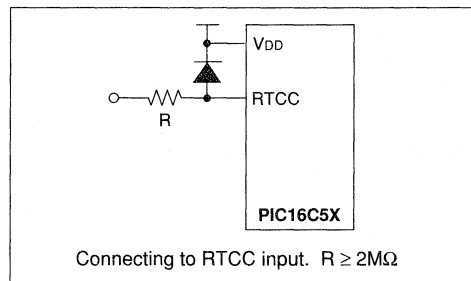
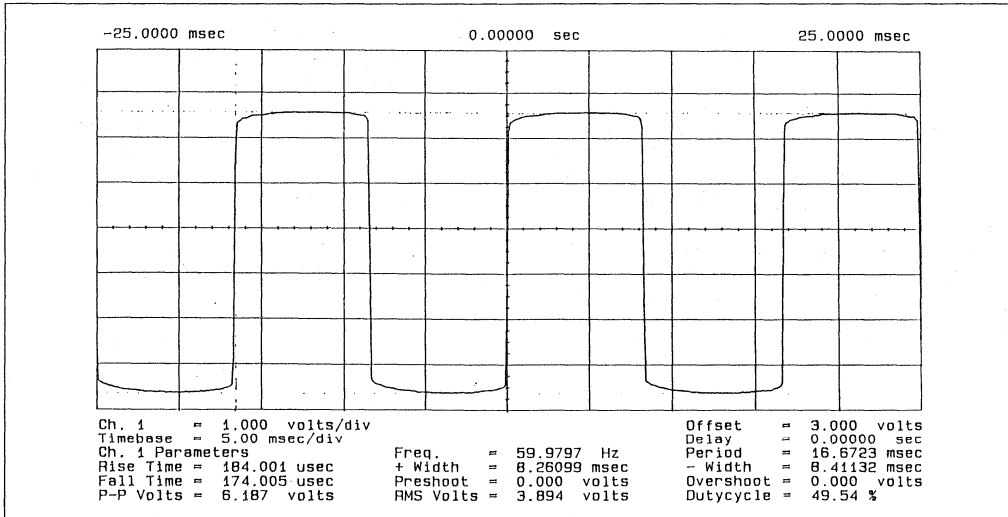


FIGURE 5 - INPUT WAVEFORM



Waveform at part pin (RA0) •

R = 100K; Line: 60 hz, 110 V

Author: Doug Cox
Logic Products Division

Implementing Wake-Up on Key Stroke

INTRODUCTION

In certain applications, the PIC16CXX is exercised only when a key is pressed, eg. remote keyless entry. In such applications, the battery life can be extended by putting the PIC16CXX to sleep during the inactive state and when a key is pressed, the PIC16CXX wakes up does its task and then goes back to sleep.

IMPLEMENTATION

The circuit in Figure 1 depicts an application with two keys. The PIC16C54 is normally in SLEEP mode consuming very little operating current. If either of the two keys is pressed, the PIC16C5X 'wakes up', scans the keys and turns on one of the two LED's. When SW1 is pressed, the green LED is turned on and when SW2 is pressed the red LED is turned on. The LED's are used purely for demonstration purposes. In real life application, a transmission will be completed before putting the PIC16C5X back in sleep. This example can be extended to handle more than two keys.

In the sleep mode, the scan outputs (SCAN1 and SCAN2) are both set to a low logic level. In this state, the capacitor C is fully charged and a high logic level is present at the MCLR pin of the PIC16C5X. When a key

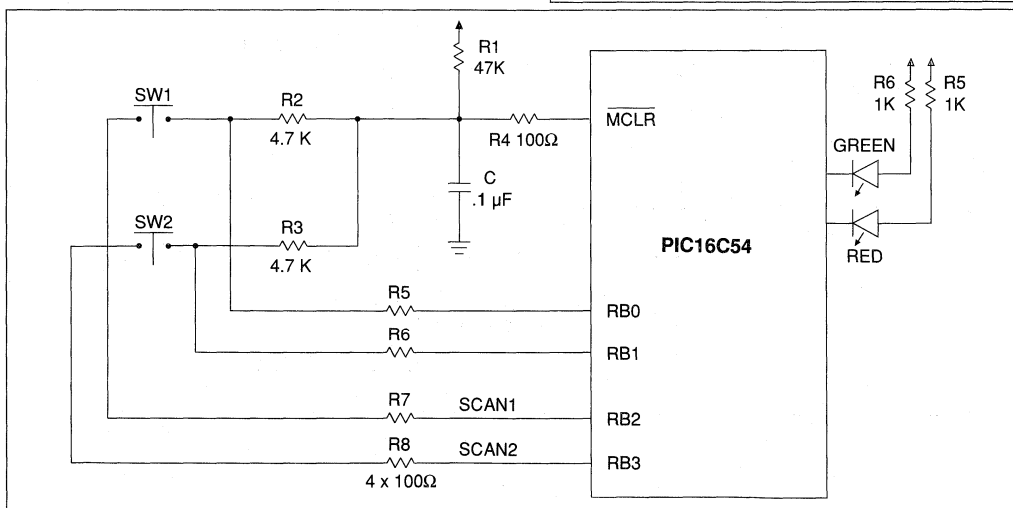
is pressed, C discharges through either R2 or R3 (depending on SW1 or SW2 being pressed) and the voltage across C falls rapidly (approx. 1 ms), causing a low at the MCLR pin of the PIC16C5X, which in turn causes the PIC16C5X to wake up and enter its reset state. In reset, the SCAN1 and SCAN2 outputs default to a high impedance mode, so the discharge path for capacitor C is blocked and it charges to a high level through resistor R1. Note that the RC values have been chosen such, that the discharge and charge cycles times are less than the reset time for the PIC16C5X (approx. 18 ms), and certainly far less than the minimum duration of a key-press (approx. 50-100 ms).

After the reset cycle is completed, the code execution momentarily takes the SCAN1 and SCAN2 outputs low in order to sample the key stroke(s). This does not cause the capacitor to discharge since the duration of the low is of the order of 10 micro seconds.

Once the keystroke function has been executed, the program loops until the key has been released, sets the SCAN1 and SCAN2 outputs low and "goes back to sleep". Resistors R4-R8 are not required for functionality. These are recommended to provide protection from electrostatic discharge (ESD). Switches SW1 and SW2, when pressed may frequently pass ESD to the PIC16C54.

Author: Stan D'Souza
Logic Products Division

FIGURE 1 - TWO KEY INTERFACE TO PIC16C5X



Key Stroke Wake-Up

MPASM B0.54
Key Stroke Wake Up

PAGE 1

```

                TITLE      "Key Stroke Wake Up"
                LIST P = 16C54
;*****
; Program demonstrating key stroke wake up for
; the PIC16CXX. Program has been implemented for
; two keys, but can be extended for more keys.
; When SW1 is pressed a green LED lights up.
; When SW2 is pressed a red LED lights up.
;*****
;
; Define equates
;
0002          PC      EQU    2
0006          PORT_B EQU    6
0002          SCAN1  EQU    2
0003          SCAN2  EQU    3
0000          SW1    EQU    0
0001          SW2    EQU    1
0004          GRN_LED EQU    4
0005          RED_LED EQU    5
0014          MSEC_20 EQU    D'20'
0008          DB1    EQU    8
0008          GP     EQU    8
0009          DB2    EQU    9
;
;PORT_B ASSIGNMENTS:
; 0 -> SW1      INPUT
; 1 -> SW2      INPUT
; 2 -> SCAN1    OUTPUT
; 3 -> SCAN2    OUTPUT
; 4 -> GRN_LED  OUTPUT
; 5 -> RED_LED  OUTPUT
; 6&7 -> ASSIGNED AS DUMMY OUTPUTS
;
;
;          ORG      0
;
; START
0000 0910     CALL    INIT_PORT_B    ;INITIALIZE PORT B
0001 0920     CALL    DELAY          ;DELAY 20 MSECS
0002 0915     CALL    SCAN_KEYS      ;GET KEY VALUES
0003 0028     MOVWF   GP              ;SAVE IN RAM
0004 0608     BTFSC  GP,SW1          ;SKIP IF SW1 NOT PRESSED
0005 0929     CALL    TURN_GREEN_ON  ;ELSE DO ROUTINE
0006 0628     BTFSC  GP,SW2          ;SKIP IF SW2 NOT PRESSED
0007 092B     CALL    TURN_RED_ON    ;ELSE DO ROUTINE
;
; CHK_FOR_KEY
0008 0920     CALL    DELAY          ;DELAY FOR 20 MSEC
0009 0915     CALL    SCAN_KEYS      ;GET KEY HIT
000A 0F00     XORLW  0               ;EXCL. OR WITH 0
000B 0743 0A08 BNZ    CHK_FOR_KEY    ;KEY STILL PRESSED
;                                     ;THEN LOOP
;
; NO_KEY_PRESSED
000D 0446     BCF    PORT_B,SCAN1    ;SET SCAN LINES LOW
000E 0466     BCF    PORT_B,SCAN2    ;
000F 0003     SLEEP                    ;SLEEP
;
;
; INIT_PORT_B
0010 0C03     MOVLW  B'00000011'    ; config RB0, 1 as i/p's
0011 0006     TRIS  PORT_B          ; and RB2-7 as o/p's
0012 0CFF     MOVLW  0FFh
0013 0026     MOVWF  PORT_B          ;DEFAULT VALUES FOR PORT_B

```

Key Stroke Wake-Up

```
0014 0800          RETLW  0          ;RETURN WITH NO ERROR
;
;This routine, scans two keys and returns the following:
; 0 if no key is pressed
; 1 if SW1 is pressed
; 2 if SW2 is pressed
; 3 if SW1 and SW2 are pressed
;
SCAN_KEYS
0015 0446          BCF     PORT_B,SCAN1  ;ENABLE SCAN FOR SW1
0016 0466          BCF     PORT_B,SCAN2  ;EABLE SCAN FOR SW2
0017 0C03          MOVLW  B'00000011'  ;LOAD MASK IN W
0018 0146          ANDWF  PORT_B,0      ;AND WITH PORT
0019 0546          BSF     PORT_B,SCAN1  ;DISABLE SCAN
001A 0566          BSF     PORT_B,SCAN2  ;
001B 01E2          ADDWF  PC,1          ;GET OFFSET TO TABLE
001C 0803          RETLW  3          ;SW1 AND SW2 PRESSED
001D 0802          RETLW  2          ;SW2 PRESSED
001E 0801          RETLW  1          ;SW1 PRESSED
001F 0800          RETLW  0          ;NO KEY PRESSED
;
;DELAY, IS A APPROX. WAIT FOR 20.4mSECS, FOR A SYSTEM
;USING A 2 Mhz CRYSTAL CLOCK.
DELAY
0020 0C14          MOVLW  MSEC_20
0021 0028          MOVWF  DB1
DLY1
0022 0069          CLRF  DB2
0023 02E8          DECFSZ DB1
0024 0A26          GOTO  DLY2
0025 0800          RETLW  0
DLY2
0026 02E9          DECFSZ DB2          ;INNER LOOP = 1.02 MSEC.
0027 0A26          GOTO  DLY2          ;
0028 0A22          GOTO  DLY1
;
;
TURN_GREEN_ON
0029 0486          BCF     PORT_B,GRN_LED
002A 0800          RETLW  0
;
TURN_RED_ON
002B 04A6          BCF     PORT_B,RED_LED
002C 0800          RETLW  0
;
END

Errors   :   0
Warnings :   0
```

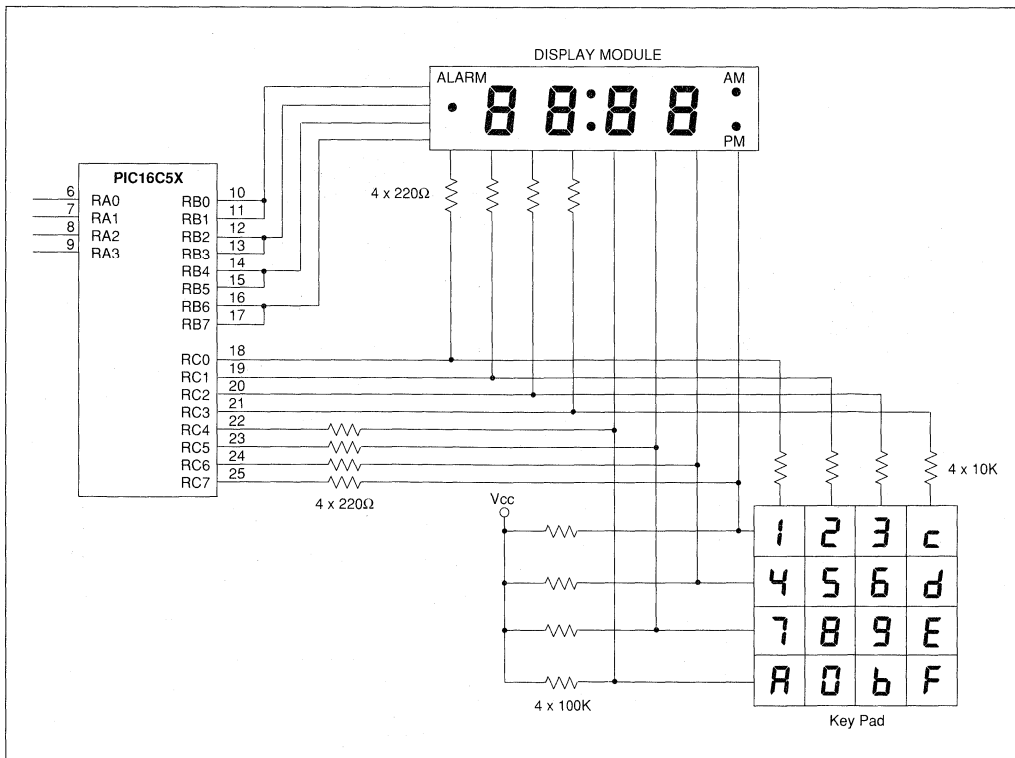
Multiplexing LED Drive and a 4x4 Keypad Sampling

INTRODUCTION

Many applications require drive to LEDs along with an interface to a keypad. Implementing such designs usually involves using up significant amounts of the processors I/O lines. This application note describes a method which uses only 16 I/O pins of a PIC16C5X microcontroller to sample a 4x4 keypad matrix, and directly drive four 7 segment LEDs (see Figure 1). Direct drive of the LEDs is possible, because of the high sink and source capabilities of the PIC16C5X microcontroller, thus eliminating the use of external drive transistor, and resulting in reduced cost and complexity of the overall circuit.

Typically applications having LEDs and keypads also keep track of real time, in order to synchronize certain key events. An Industrial Clock/Timer example has been used in this ap note as a demonstration of this technique. The software overhead to keep track of real time is minimal and the user can modify the code to significantly expand the functionality of this circuit.

FIGURE 1 - PIC16C5X INTERFACE TO A SEGMENT DISPLAY AND 4X4 KEYPAD



Multiplexing LEDs/Keypad

PART A: 4X4 KEY MATRIX SAMPLING

Implementation

The 4x4 Key Matrix is connected to port C of the PIC16C5X (Figure 2a). The 4 columns are connected to RC0-RC3 and the 4 rows are connected to RC4-RC7. Each digit is refreshed every 20 ms. with a 5 ms pulse. The keypad is sampled every 20 ms with four 3 μs pulses (Figure 3).

The Keypad sampling is as follows:

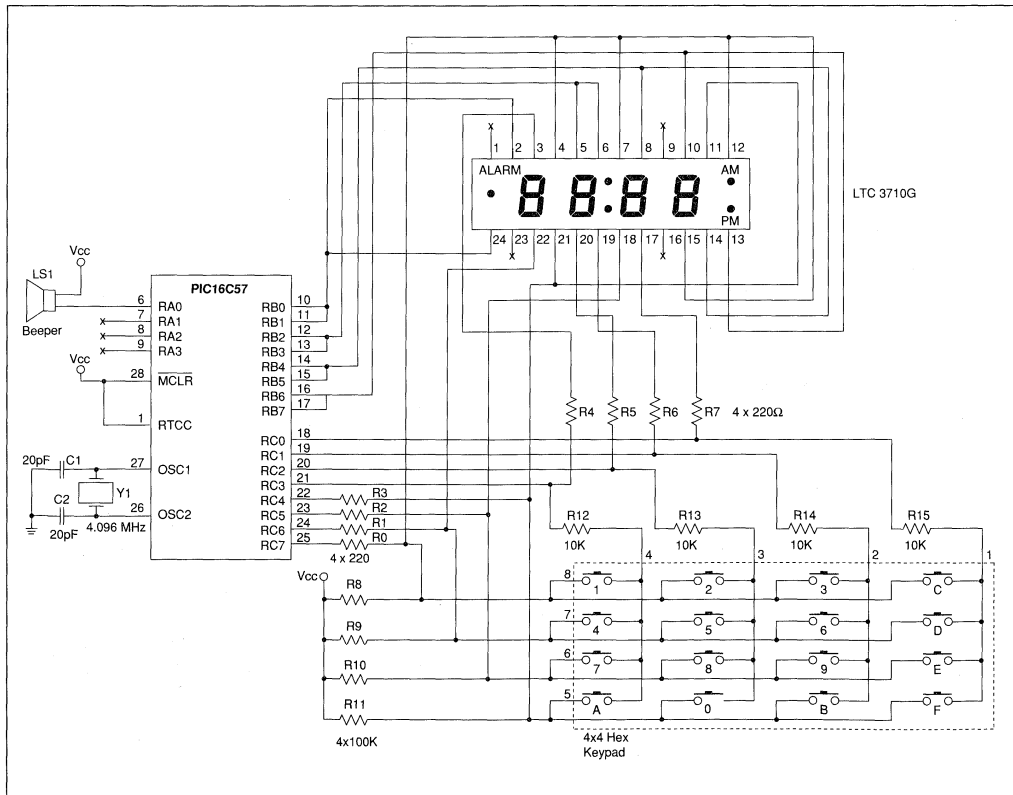
1. The columns are connected to output pins, and the rows are connected to input pins.
2. Each column is sequentially driven to a low voltage while at the same instance the four rows are sampled. Since the rows are all held high with pull-up resistors, all four inputs will normally be high. If a key is pressed in a column which is at a low level, that low level will be conducted to the input pin through the closed key and the corresponding row will be sensed as a low.

3. Before a new column is brought low, care should be taken to discharge the input pins (see code section for details).
4. A 50 ms key debounce technique has been implemented in the software, in order to eliminate multiple key strokes.

Notes:

1. Resistors R8-R11 and R12-R14 have been selected such that their ratio is 1:10. This will insure a 0.5 Volt level at the input, when a key is pressed. Also R8-R14 should have a value such that their current contribution to the LEDs segments is negligible.
2. In circuits where there is substantial interference between the key matrix and the LED drive circuit, the alternative circuit (Figure 2b) should be utilized. Diodes in the path of all pins connected to the keypad insure that there is minimal interference from the keypad, when it is not being sampled.

FIGURE 2A - PIC16C5X INDUSTRIAL CLOCK/TIMER SCHEMATIC



PART B: INDUSTRIAL CLOCK/TIMER

Clock Selection

The 4.096 MHz crystal oscillator is the time base. The PIC16C5X internally divides the clock by 4 to give an internal clock of 1.024 MHz. This clock is further divided by 32 (by the prescaler in the OPTIONS register) to give a clock of 32 KHz which is used to increment the RTCC in the PIC16C5X. If the RTCC is initialized to 96, it would overflow to 0 in 5 ms.

$$(256-96) \times (1/32000) = 5.000 \text{ ms}$$

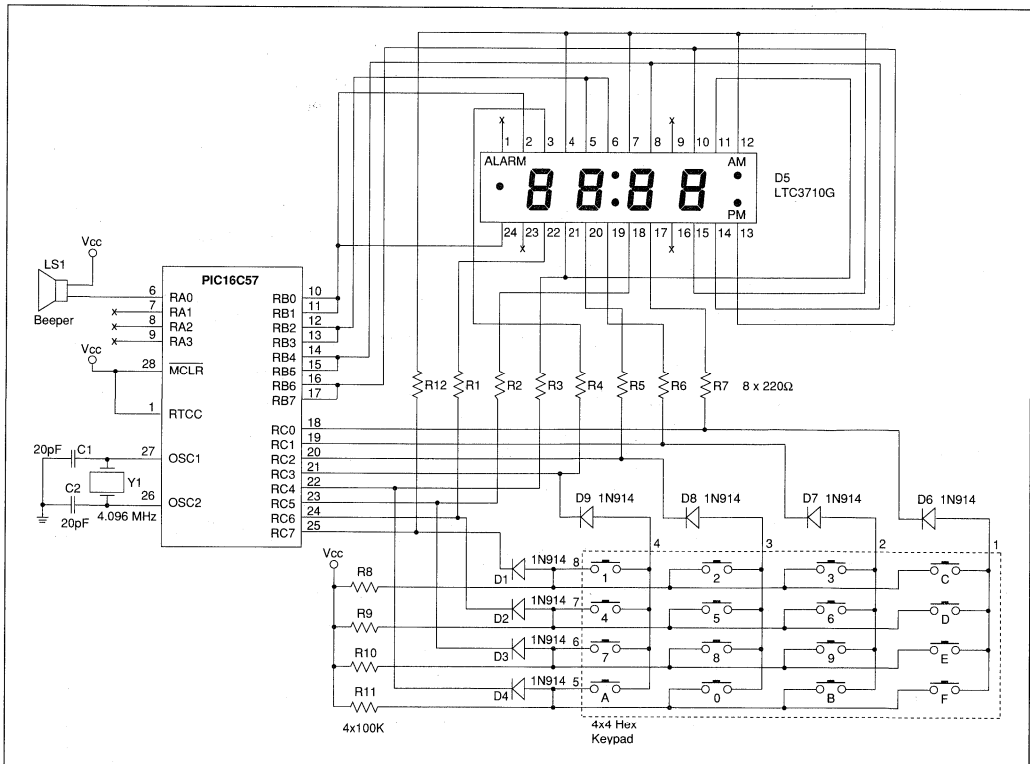
This 5 ms is used to count the seconds, minutes and hours in the clock/timer. It is also used as a time base to update the display digits and sample the keyboard. The clock speed being 4.096 MHz, each instruction will

execute in 1 μ s. Therefore in 5 ms, approximately 5000 instruction can be executed. This gives us sufficient time to execute a large section of code and not miss the overflow in the RTCC.

Using a 3.579545 MHz color burst crystal oscillator as a time base

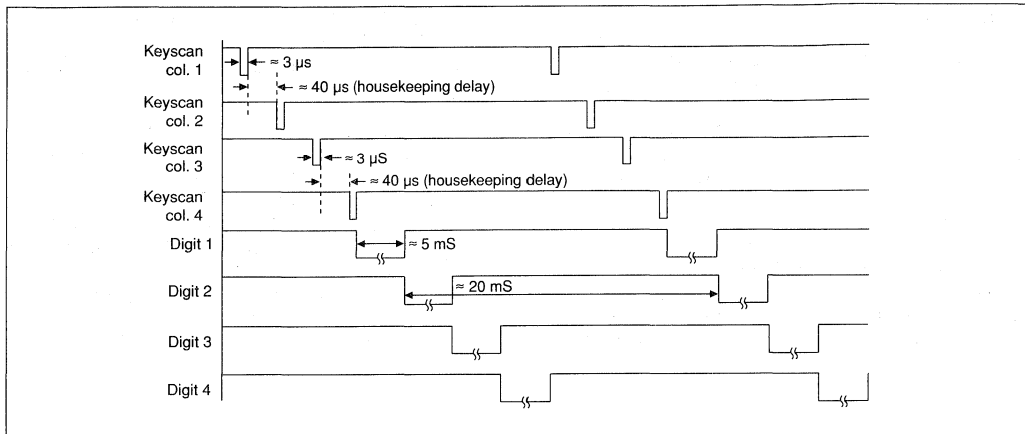
Some users may want to use a color burst crystal oscillator as a time base, because of its low cost. If a 3.579545 MHz crystal is used, then the internal clock will be 1.117 μ s. If this is prescaled by 32, the RTCC will be incremented every 35.758 μ s. Initializing the RTCC with 116 will cause it to overflow to 0 in 5.006 ms. giving us an error of 0.12%. This error can be corrected in software by making time adjustments every minute and/or every hour.

FIGURE 2B - PIC16C5X ALARM CLOCK SCHEMATIC (USING DIODES)



Multiplexing LEDs/Keypad

FIGURE 3 - KEY SCAN AND LED DIGIT SELECT TIMING



FEATURES

The Flow Chart (Figure 4) shows the sequence of events in the clock/timer software. The clock has the following features:

1. 12 hour clock with a.m./p.m.
2. 12 hour alarm with a.m./p.m.
3. Full function hex keypad (fig. 5).
4. AA audible alarm for 1 minute.
5. 10 minute alarm disable.

SETTING CLOCK/TIMER FUNCTIONS

Function	Key Sequence to Activate Function
Set Real Time	Set → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
View Alarm Time	Alarm (alarm time is displayed for 5 seconds)
Set Alarm Time	Alarm → Set (must be pressed when alarm LED is flashing) → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
Enable/Disable Alarm	Alarm → Alarm (toggles alarm status)
Disable AA alarm	Disable Alarm (disable audible beep for 10 minutes)
Clear Alarm	Clear Alarm (clears audible alarm)
Abort Entry	Clear Entry (aborts data entry mode when setting real and alarm time)

- Notes:
1. Valid key strokes will be acknowledged with a beep.
 2. Hours and minutes used above correspond to digits 0 - 9 on the keypad.

FIGURE 4 - TIMER/CLOCK FLOW CHART

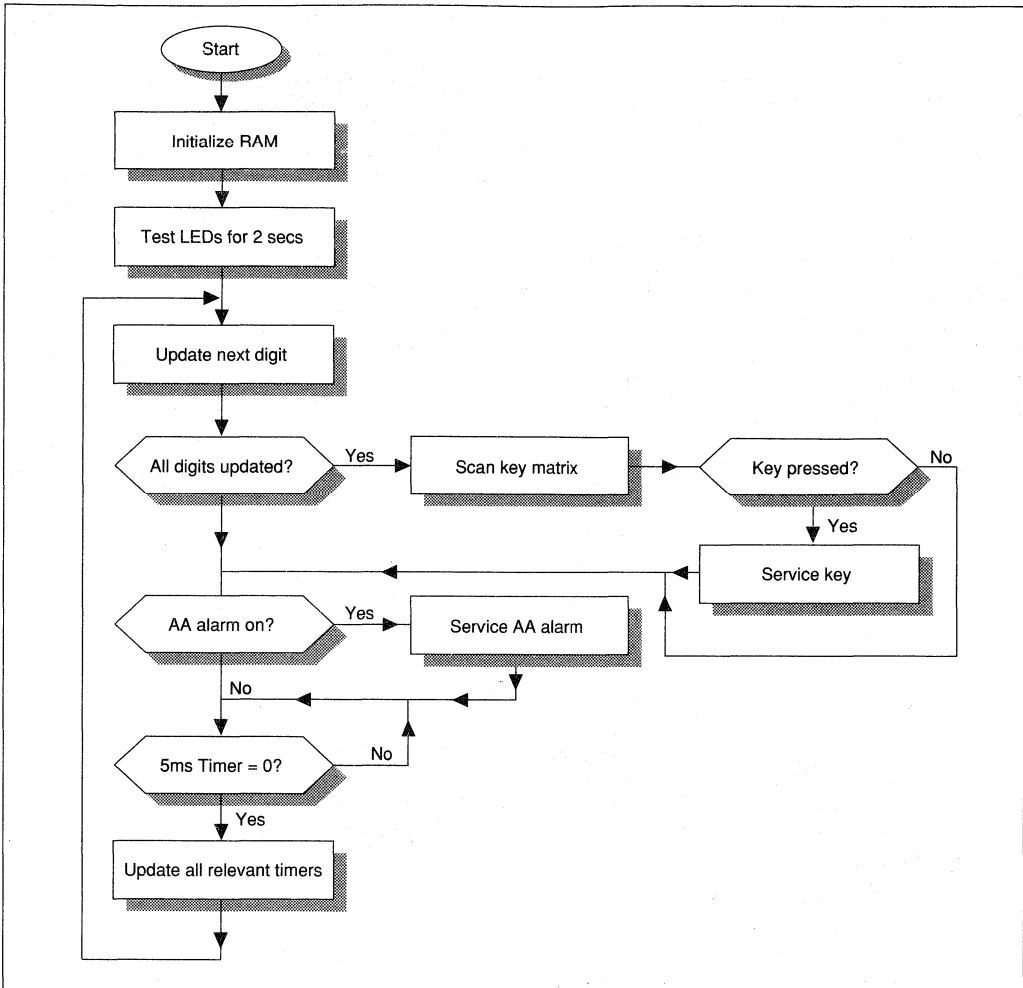
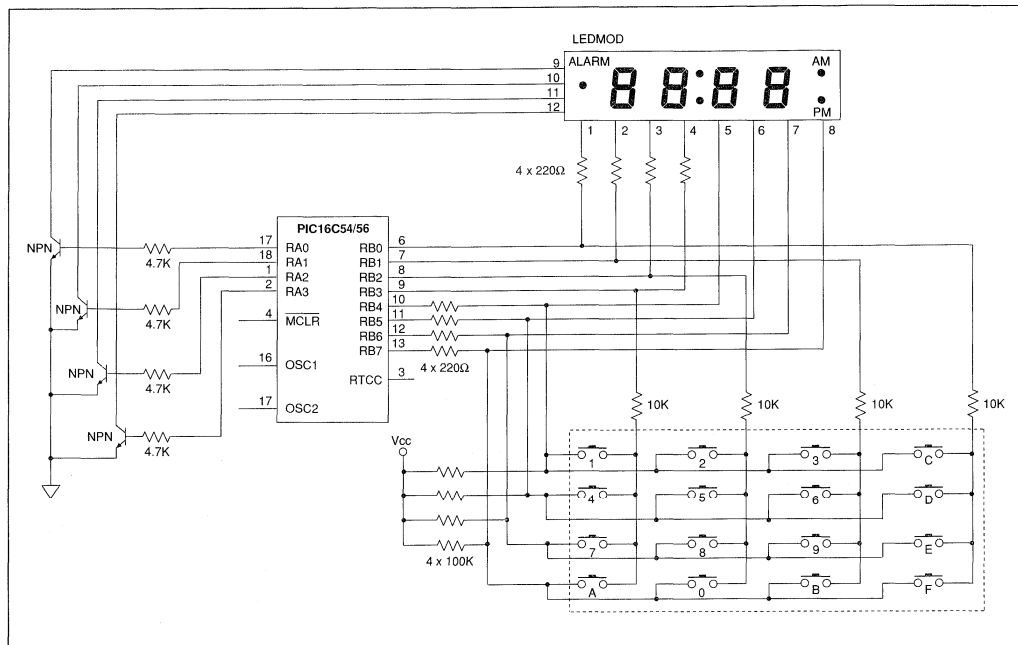


FIGURE 5 - KEYPAD

1	2	3	DISABLE ALARM
4	5	6	AM/PM
7	8	9	CLEAR ALARM
ALARM TIME	0	CLEAR ENTRY	SET

Multiplexing LEDs/Keypad

FIGURE 6 - INTERFACE TO PIC16C54/56



SUMMARY

This Application Note demonstrates a simple method of interfacing the PIC16C5X to 7 segment LEDs and a keypad. The key features of the PIC16C5X which made this possible are:

1. High sink/source of the I/O ports.
2. Fast instruction cycle for quick key-scan.
3. RISC processor allowing minimal overhead for real time clock maintenance.
4. Re-configurable I/O ports, enabling dual functionality of ports.

Figure 6 depicts a block diagram connecting a PIC16C54/56 to a 4 digit 7 segment LED display and a 4x4 hex keypad. Since only 12 I/O pins are available in the PIC16C54/56, external npn transistor will have to be utilized to sink the current from each digit.

CODE SIZE

Key scan → 97 bytes

Display update → 113 bytes

Author: Stan D'Souza
Logic Products Division

Multiplexing LEDs/Keypad

APPENDIX A: CODE LISTING

MPASM B0.54
Alarm Clock

PAGE 1

```

                                TITLE      "Alarm Clock"
                                LIST P = 16C57
;
;Define Equates:
;
07FF          PIC57 EQU      7FFH
;*****
;External Ossc. used = 4.096Mhz. Prescaler of 32 used, which gives a
;31.25 microSec increment of the RTCC. If RTCC is intially loaded with 96,
;it would overflow to 0 in 5.000 milliSecs. Giving a 0.00% error.
0060          MSEC5 EQU      D'96'
;*****
0000          C EQU          0
0000          BEP EQU        0
0000          RTATS EQU      0
0001          DC EQU         1
0001          HR10 EQU        1
0002          Z EQU          2
0002          HR EQU          2
0003          MIN10 EQU       3
0004          MIN EQU         4
0004          FLASH EQU       4
0005          PA0 EQU         5
0005          KEY_BEEP EQU    5
0005          AMPM EQU        5
0006          PA1 EQU         6
0000          F0 EQU          0
0006          KEY_HIT EQU     6
0006          ALED EQU        6
0007          AM_PM EQU       7
0003          COLON EQU       3
0002          ALRMLED EQU     2
0007          SERVICED EQU    7
0000          ALONOF EQU      0
0001          INAL EQU        1
0002          SILNC EQU       2
0003          INAA EQU        3
0005          INKEYBEP EQU    5
;
;DEFINE RAM LOCATIONS:
0001          RTCC EQU        1
0002          PC EQU          2
0003          STATUS EQU      3
0004          FSR EQU         4
0005          PORT_A EQU       5
0006          PORT_B EQU       6
0007          PORT_C EQU       7
;DEFINE REAL TIME MODE REGS (RTM)
0008          MSTMR EQU        8 ;MILLI SEC. TIMER
0009          STMR EQU         9 ;SEC. TIMER
;*****
;DO NOT CHANGE RELATIVE POSITION OF NEXT 6 BYTES
000A          MTMR EQU         0A ;MIN. TIMER
000B          HTMR EQU         0B ;HOUR TIMER
;DEFINE ALARM TIME MODE REGS (ATM)
000C          MALARM EQU       0C ;MIN. ALARM
000D          HALARM EQU       0D ;HOUR ALARM
;DEFINE DATA ENTRY MODE REGS (DEM)
000E          MENTRY EQU       0E ;MIN. ENTRY
000F          HENTRY EQU       0F ;HOUR ENTRY
;*****
```

2

Multiplexing LEDs/Keypad

```

;
;DEFINE FLAG REG AND FUNCTION:
0010      FLAG EQU 10
;          BIT # 7|6|5|4|3|2|1|0|
;          -----|---|---|---|---|
;          X|X|X|X|X|X|X|0|0|0| -> REAL TIME MODE (RTM)
;          X|X|X|X|X|X|X|0|1|1| -> ALARM TIME MODE (ATM)
;          X|X|X|X|X|X|X|1|0|1| -> DATA ENTRY MODE (DEM)
;          X|X|X|X|X|X|X|1|1|1| -> TEST MODE (TM)
;          X|X|X|X|X|Y|X|X|X| -> ALRMLD ON/OFF
;          X|X|X|X|Y|X|X|X|X| -> COLON LED ON/OFF
;          X|X|X|Y|X|X|X|X|X| -> FLASH DISPLAY
;          X|X|Y|X|X|X|X|X|X| -> KEY_BEEP
;          X|Y|X|X|X|X|X|X|X| -> KEY_HIT (0/1)
;          Y|X|X|X|X|X|X|X|X| -> SERVICED
; X = DEFINED ELSEWHERE IN TABLE
; Y = DEFINED AS SHOWN (0/1)
;
0011      TEMP EQU 11
0012      DIGIT EQU 12
0013      NEW_KEY EQU 13
0014      KEY_NIBL EQU 14
0015      DEBOUNCE EQU 15
0016      MIN_SEC EQU 16 ;MIN/SECONDS TIMER
0017      ENTFLG EQU 17
;flag dedicated to the key entry mode
;          BIT # 7|6|5|4|3|2|1|0|
;          -----|---|---|---|---|
;          X|X|X|X|X|X|X|X|Y| -> REAL/ALARM TIME STATUS
;          X|X|X|X|X|X|X|Y|X| -> HR10 DONE
;          X|X|X|X|X|Y|X|X|X| -> HR DONE
;          X|X|X|X|Y|X|X|X|X| -> MIN10 DONE
;          X|X|X|Y|X|X|X|X|X| -> MIN DONE
;          X|X|Y|X|X|X|X|X|X| -> INKEYBEP
;          X|Y|X|Y|X|X|X|X|X| -> NOT USED
;          Y|X|X|X|X|X|X|X|X| -> NOT USED
;
;
0018      ALFLAG EQU 18
;flag dedicated to the alarm
;          BIT # 7|6|5|4|3|2|1|0|
;          -----|---|---|---|---|
;          X|X|X|X|X|X|X|X|Y| -> ALONOF
;          X|X|X|X|X|X|X|Y|X| -> INAL
;          X|X|X|X|X|Y|X|X|X| -> SILNC
;          X|X|X|X|Y|X|X|X|X| -> INAA
;          X|X|X|Y|X|X|X|X|X| -> NOT USED
;          X|X|Y|X|X|X|X|X|X| -> NOT USED
;          X|Y|X|Y|X|X|X|X|X| -> NOT USED
;          Y|X|X|X|X|X|X|X|X| -> NOT USED
;
;
0019      AAFLAG EQU 19
;flag dedicated to the AA alarm
001A      AATMR EQU 1A

```


Multiplexing LEDs/Keypad

```

;
RESET_ATM
001C 0216      MOVF      MIN_SEC,W      ;GET MIN/SEC
001D 0E0F      ANDLW     B'00001111'      ;
001E 0743      BTFSS     STATUS,Z      ;Z THEN SKIP
001F 0800      RETLW     0              ;ELSE RETURN
0020 0410      BCF       FLAG,0          ;SET TO RTM
0021 0450      BCF       FLAG,ALRMLED    ;CLEAR LED
0022 0618      BTFSC     ALFLAG,ALONOF    ;TEST STAT
0023 0550      BSF       FLAG,ALRMLED    ;SET LED
0024 0800      RETLW     0              ;RETURN
;
;
UPDATE_DISPLAY
0025 0C00      MOVLW     B'00000000'      ;CLEAR SEG DRIVE
0026 0027      MOVWF     PORT_C          ;
0027 0C3F      MOVLW     B'00111111'      ;SEE IF LAST DIGIT
0028 0186      XORWF     PORT_B,0          ;
0029 0643      BTFSC     STATUS,Z      ;NO THEN SKIP
002A 0A6F      GOTO      SCAN_KP      ;ELSE SCAN KEYPAD
UP_DSP_1
;SELECT DIGIT TO BE DISPLAYED
002B 0246      COMF      PORT_B,0          ;GET COMPL. PORT B IN W
002C 0643      BTFSC     STATUS,Z      ;NO DIGIT SELECTED?
002D 0CC0      MOVLW     B'11000000'      ;THEN SELECT DEFAULT
002E 0031      MOVWF     TEMP              ;SAVE IN TEMP
002F 0271      COMF      TEMP              ;COMPLEMENT VALUE
0030 0503      BSF       STATUS,C          ;SET CARRY
0031 0371      RLF       TEMP              ;SHIFT LEFT
0032 0703      BTFSS     STATUS,C          ;IF C=1 THEN SKIP
0033 0371      RLF       TEMP              ;ELSE 3 TIMES...
0034 0371      RLF       TEMP              ;THRU CARRY
0035 0211      MOVF      TEMP,0           ;GET IN W
0036 0026      MOVWF     PORT_B          ;OUTPUT TO PORT
;NOW THAT DIGIT IS SELECTED, SELECT SEG VALUES FOR THAT DIGIT
;FIRST FIND MODE OF OPERATION.
0037 0C0A      MOVLW     MTMR              ;LOAD FSR WITH MTMR
0038 0024      MOVWF     FSR              ;
0039 0210      MOVF      FLAG,0          ;GET FLAG IN W
003A 0E03      ANDLW     B'00000011'      ;MASK OTHER BITS
003B 0031      MOVWF     TEMP              ;SAVE IN TEMP
003C 0F03      XORLW     B'00000011'      ;IN TEST MODE
003D 0643      BTFSC     STATUS,Z      ;NO THEN SKIP
003E 0A4B      GOTO      DO_TM            ;ELSE TEST MODE
003F 0403      BCF       STATUS,C          ;CLEAR CARRY
0040 0371      RLF       TEMP              ;LEFT SHIFT TEMP
0041 0211      MOVF      TEMP,0           ;GET IN W
0042 01E4      ADDWF     FSR              ;CHANGE INDIRECT POINTER
0043 0954      CALL     GET_7_SEG         ;GET 7 SEG DATA IN W
0044 0032      MOVWF     DIGIT           ;SAVE IN DIGIT LOC.
0045 09D1      CALL     MASK_ANNC         ;MASK ANNC TO DIGIT
0046 0690      BTFSC     FLAG,FLASH      ;NO FLASH THEN SKIP
0047 094E      CALL     CHK_HALF_SEC     ;ELSE CHK. IF ON
0048 0212      MOVF      DIGIT,0         ;GET BACK DIGIT
0049 0027      MOVWF     PORT_C          ;OUTPUT TO PORT
004A 0800      RETLW     0              ;RETURN
;
DO_TM
004B 0CFF      MOVLW     B'11111111'      ;LIGHT ALL SEGMENTS
004C 0027      MOVWF     PORT_C          ;
004D 0800      RETLW     0              ;RETURN FROM UPDATE DISPLAY
;
CHK_HALF_SEC
004E 0770      BTFSS     FLAG,COLON      ;IF COLON ON THEN DO
004F 0A51      GOTO      BLANK_DSP       ;ELSE BLANK DISPLAY
0050 0800      RETLW     0
BLANK_DSP
0051 0C00      MOVLW     B'00000000'      ;MAKE PORT C LOW
0052 0032      MOVWF     DIGIT           ;
0053 0800      RETLW     0

```


Multiplexing LEDs/Keypad

```

;ON ENTRY FSR POINTS TO THE REAL TIME MODE'S MINUTES REGISTER.
;ON RETURN FSR POINTS TO THE TIMER REGISTER TO BE DISPLAYED.
;W REG. CONTAINS THE DECODED 7 SEG. INFO OF THE DIGIT
;TO BE DISPLAYED
;
GET_7_SEG
0054 0246      COMF    PORT_B,0      ;COMPLEMENT B -> W
0055 0EF0      ANDLW   B'11110000'      ;MASK LO NIBBLE
0056 0643      BTFSC  STATUS,Z          ;NZ THEN SKIP
0057 02A4      INCF    FSR                ;INC POINTER
0058 0200      MOVF    F0,0              ;MOVE INDIRECT TO W
0059 0031      MOVWF  TEMP              ;GET INTO TEMP
005A 0246      COMF    PORT_B,0      ;COMPL.B -> W
005B 0EF0      ANDLW   B'11110000'      ;MASK LO NIBBLE
005C 0643      BTFSC  STATUS,Z          ;IF D1/2 THEN
005D 04F1      BCF    TEMP,AM_PM        ;CLEAR AM/PM BIT
005E 0246      COMF    PORT_B,0      ;GET PORT B AGAIN
005F 0ECC      ANDLW   B'11001100'      ;SEE IF D2 OR D4
0060 0643      BTFSC  STATUS,Z          ;YES THEN SKIP
0061 03B1      SWAPF  TEMP              ;SWAP TEMP
0062 0C0F      MOVLW  B'00001111'      ;MASK HI NIBBLE
0063 0151      ANDWF  TEMP,0           ;
0064 01E2      ADDWF  PC                ;ADD TO PC
0065 083F      RETLW  B'00111111'      ;CODE FOR 0
0066 0806      RETLW  B'00000110'      ;CODE FOR 1
0067 085B      RETLW  B'01011011'      ;CODE FOR 2
0068 084F      RETLW  B'01001111'      ;CODE FOR 3
0069 0866      RETLW  B'01100110'      ;CODE FOR 4
006A 086D      RETLW  B'01101101'      ;CODE FOR 5
006B 087D      RETLW  B'01111101'      ;CODE FOR 6
006C 0807      RETLW  B'00000111'      ;CODE FOR 7
006D 087F      RETLW  B'01111111'      ;CODE FOR 8
006E 0867      RETLW  B'01100111'      ;CODE FOR 9
;
;This routine scans the 4x4 hex key pad for a key hit.
;If key is pressed, KEY_HIT flag is set and the value of
;the hex key is returned in reg NEW_KEY
;If no key is detected, then a 0xff value is returned in
;register NEW_KEY and the flag KEY_HIT is reset.
;
SCAN_KP
006F 06D0      BTFSC  FLAG,KEY_HIT     ;KEY UNDER SERVICE?
0070 0A2B      GOTO   UP_DSP_1        ;YES SKIP ROUTINE
0071 0CFE      MOVLW  B'11111111'      ;SET DIGIT SINKS ...
0072 0026      MOVWF  PORT_B          ;TO HIGH
0073 0CF7      MOVLW  B'11110111'      ;SET KEY COL LOW
0074 0031      MOVWF  TEMP            ;SAVE IN TEMP

SKP1
0075 0C00      MOVLW  B'00000000'      ;SET PORT C AS OUTPUTS
0076 0007      TRIS  PORT_C          ;
0077 0211      MOVF  TEMP,W           ;
0078 0E0F      ANDLW  B'00001111'      ;DISCHARGE PINS FOR MEMBRANE KEYPADS
0079 0027      MOVWF  PORT_C          ;
007A 0CF0      MOVLW  B'11110000'      ;SET AS I/O
007B 0007      TRIS  PORT_C          ;
007C 0211      MOVF  TEMP,W           ;GET OLD VALUE
007D 0027      MOVWF  PORT_C          ;OUTPUT TO PORT
007E 0207      MOVF  PORT_C,W         ;INPUT PORT VALUE
007F 0EF0      ANDLW  B'11110000'      ;MASK LO BYTE
0080 0FF0      XORLW  B'11110000'      ;SEE IF KEY HIT
0081 0743      BTFSS  STATUS,Z        ;NO KEY THEN SKIP
0082 0A8D      GOTO   DET_KEY        ;LOAD KEY VALUE

SKP3
0083 0503      BSF    STATUS,C         ;SET CARRY
0084 0331      RRF    TEMP            ;MAKE NEXT COL. LOW
0085 0603      BTFSC  STATUS,C        ;ALL DONE THEN SKIP
0086 0A75      GOTO   SKP1
0087 0073      CLR    NEW_KEY         ;SET NEW_KEY = FF
0088 00F3      DECF  NEW_KEY         ;

```



Multiplexing LEDs/Keypad

```

SKP2
0089 0067      CLRF   PORT_C      ;SETPORT C AS ...
008A 0C00      MOVLW  B'00000000'    ;OUTPUTS
008B 0007      TRIS   PORT_C      ; /
008C 0A2B      GOTO   UP_DSP_1    ;RETURN

DET_KEY
;key is detected
008D 0293      INCF   NEW_KEY,W      ;CHK IF KEY ...
008E 0743      BTFSS  STATUS,Z      ;WAS RELEASED
008F 0A89      GOTO   SKP2          ;NO THEN RETURN
0090 0207      MOVF   PORT_C,W      ;GET RAW KEY...
0091 0D0F      IORLW  B'00001111'   ;VALUE.
0092 0151      ANDWF  TEMP,W        ; /
0093 0033      MOVWF  NEW_KEY       ;SAVE IN NEW_KEY
0094 0998      CALL  GET_KEY_VAL   ;GET ACTUAL KEY ...
0095 0033      MOVWF  NEW_KEY       ;VALUE
0096 05D0      BSF   FLAG,KEY_HIT  ;SET KEY HIT FLAG
0097 0A89      GOTO   SKP2          ;RETURN

;
;This routine decodes the hex value from the "raw" data got
;from scanning the rows and cols.
; actual key value      raw hex value
; ONE                   EQU    77
; TWO                   EQU    7B
; THREE                 EQU    7D
; C                     EQU    7E
; FOUR                 EQU    0B7
; FIVE                 EQU    0BB
; SIX                  EQU    0BD
; D                    EQU    0BE
; SEVEN                EQU    0D7
; EIGHT               EQU    0DB
; NINE                 EQU    0DD
; E                    EQU    0DE
; A                    EQU    0E7
; ZERO                 EQU    0EB
; B                    EQU    0ED
; F                    EQU    0EE
;
;
GET_KEY_VAL
0098 0E0F      ANDLW  B'00001111'   ;GET LO NIBBLE
0099 0034      MOVWF  KEY_NIBL     ;SAVE
009A 0C04      MOVLW  4            ;SET COUNT TO 4
009B 0031      MOVWF  TEMP        ; /

GKV1
009C 0503      BSF   STATUS,C      ;SET CARRY
009D 0334      RRF   KEY_NIBL     ;ROTATE NIBBLE
009E 0703      BTFSS  STATUS,C      ;SKIP IF NOT Z
009F 0AA5      GOTO  GET_HI_KEY   ;GOTO NEXT PART
00A0 02F1      DECF  SZ            ;DEC COUNT
00A1 0A9C      GOTO  GKV1         ;LOOP

GO_RESET
00A2 05A3      BSF   STATUS,PA0    ;SET MSB
00A3 05C3      BSF   STATUS,PA1    ; /
00A4 0BFF      GOTO  SYS_RESET    ;ELSE BIG ERROR

GET_HI_KEY
00A5 00F1      DECF  TEMP          ;REDUCE BY 1
00A6 0393      SWAP  NEW_KEY,W     ;GET HI NIBBLE
00A7 0E0F      ANDLW  B'00001111'   ; /
00A8 0034      MOVWF  KEY_NIBL     ;SAVE
00A9 0211      MOVF  TEMP,W        ;GET OFFSET TO TBL
00AA 01E2      ADDWF  PC           ;LOAD IN PC
00AB 0AAE      GOTO  GET147A      ;JUMP TO NEXT PART
00AC 0AB8      GOTO  GET2580      ; /
00AD 0ABA      GOTO  GET369B      ; /
00AE 0ABC      GOTO  GETCDEF      ; /
;

```

Multiplexing LEDs/Keypad

```

GET147A
00AF 0C04      MOV LW 4          ;SET COUNT TO 4
               GETCOM
00B0 0031      MOV WF TEMP      ;
               GETCOM1
00B1 0503      BSF STATUS,C     ;SET CARRY
00B2 0334      RRF KEY_NIBL ;ROTATE RIGHT
00B3 0703      BTFSS STATUS,C  ;CHECK IF DONE
00B4 0ABE      GOTO KEY_TBL ;JUMP TO TABLE
00B5 02F1      DECFSZ TEMP     ;DEC COUNT
00B6 0AB1      GOTO GETCOM1 ;LOOP
00B7 0AA2      GOTO GO_RESET ;ELSE ERROR
;
GET2580
00B8 0C08      MOV LW 8          ;SET COUNT TO 8
00B9 0AB0      GOTO GETCOM
;
GET369B
00BA 0C0C      MOV LW D'12'     ;SET COUNT TO 12
00BB 0AB0      GOTO GETCOM
;
GETCDEF
00BC 0C10      MOV LW D'16'     ;SET COUNT TO 16
00BD 0AB0      GOTO GETCOM
;
KEY_TBL
00BE 00F1      DECF TEMP      ;REDUCE BY 1
00BF 0211      MOVF TEMP,W     ;GET IN W
00C0 01E2      ADDWF PC        ;JUMP TO TABLE
00C1 0801      RETLW 1         ;KEY 1
00C2 0804      RETLW 4         ;KEY 4
00C3 0807      RETLW 7         ;KEY 7
00C4 080A      RETLW 0A        ;KEY A
00C5 0802      RETLW 2         ;KEY 2
00C6 0805      RETLW 5         ;KEY 5
00C7 0808      RETLW 8         ;KEY 8
00C8 0800      RETLW 0         ;KEY 0
00C9 0803      RETLW 3         ;KEY 3
00CA 0806      RETLW 6         ;KEY 6
00CB 0809      RETLW 9         ;KEY 9
00CC 080B      RETLW 0B        ;KEY B
00CD 080C      RETLW 0C        ;KEY C
00CE 080D      RETLW 0D        ;KEY D
00CF 080E      RETLW 0E        ;KEY E
00D0 080F      RETLW 0F        ;KEY F
;
;
MASK_ANNC
00D1 0CFC      MOV LW B'11111100' ;CHK IF DIGIT 1
00D2 0186      XORWF PORT_B,0    ;
00D3 0643      BTFSC STATUS,Z   ;NO THEN SKIP
00D4 0AE5      GOTO MASK_ALARM ;ELSE MASK ALARM
00D5 0CF3      MOV LW B'11110011' ;CHK IF DIGIT 2
00D6 0186      XORWF PORT_B,0    ;
00D7 0643      BTFSC STATUS,Z   ;NO THEN SKIP
00D8 0AE8      GOTO MASK_COLON ;ELSE MASK COLON
00D9 0CCF      MOV LW B'11001111' ;CHK IF DIGIT 3
00DA 0186      XORWF PORT_B,0    ;
00DB 0643      BTFSC STATUS,Z   ;NO THEN SKIP
00DC 0AE1      GOTO MASK_PM   ;ELSE MASK PM
;
MASK_AM
00DD 02A4      INCF FSR         ;INC FSR
00DE 07E0      BTFSS F0,AM_PM  ;IF 0 THEN AM
00DF 05F2      BSF DIGIT,7     ;SET MSB
00E0 0AEB      GOTO BLNK_LEAD_0 ;NEXT
;
MASK_PM
00E1 02A4      INCF FSR         ;INC FSR
00E2 06E0      BTFSC F0,AM_PM  ;IF 1 THEN PM
00E3 05F2      BSF DIGIT,7     ;SET MSB
00E4 0AEB      GOTO BLNK_LEAD_0 ;NEXT

```

Multiplexing LEDs/Keypad

```

                                MASK_ALARM
00E5 0650          BTFSC  FLAG,ALRMLED  ;1 THEN LIGHT LED
00E6 05F2          BSF    DIGIT,7      ; /
00E7 0AEB          GOTO   BLNK_LEAD_0  ;NEXT

                                MASK_COLON
00E8 0670          BTFSC  FLAG,COLON   ;1 THEN LIGHT LED
00E9 05F2          BSF    DIGIT,7      ; /
00EA 0AEB          GOTO   BLNK_LEAD_0  ;NEXT

;
BLNK_LEAD_0
00EB 0210          MOVF   FLAG,W        ;GET IN W
00EC 0E03          ANDLW  B'00000011'    ;SEE IF IN DEM
00ED 0F02          XORLW  B'00000010'    ;CHECK
00EE 0643          BTFSC  STATUS,Z      ;NO THEN DO
00EF 0800          RETLW  0              ;ELSE RETURN
00F0 0CFC          MOVLW  B'11111100'    ;SEE IF DIGIT 1
00F1 0186          XORWF  PORT_B,0      ; /
00F2 0743          BTFSS  STATUS,Z      ;YES THEN SKIP
00F3 0800          RETLW  0              ;RETURN
00F4 0C3F          MOVLW  B'00111111'    ;ELSE MASK G AND ANUNC
00F5 0152          ANDWF  DIGIT,0       ;GET IN W
00F6 0F3F          XORLW  B'00111111'    ;SEE IF 0
00F7 0743          BTFSS  STATUS,Z      ;YES THEN SKIP
00F8 0800          RETLW  0              ;RETURN
00F9 0C80          MOVLW  B'10000000'    ;ELSE BLANK D1
00FA 0172          ANDWF  DIGIT        ; /
00FB 0800          RETLW  0              ;RETURN

;
;
;
;
;THIS ROUTINE SETS UP PORTS A,B,C AND THE INTERNAL
;REAL TIME CLOCK COUNTER.
INIT_CLK
00FC 0C0F          MOVLW  B'00001111'    ;MAKE ACTIVE HIGH
00FD 0025          MOVWF  PORT_A        ; /
00FE 0C00          MOVLW  B'00000000'    ;SET PORT A AS OUTPUTS
00FF 0005          TRIS   PORT_A

;
0100 0CFF          MOVLW  B'11111111'    ;SET LEVELS HIGH
0101 0026          MOVWF  PORT_B        ; /
0102 0C00          MOVLW  B'00000000'    ;SET PORT B AS OUTPUTS
0103 0006          TRIS   PORT_B

;
0104 0C00          MOVLW  B'00000000'    ;SET LEVELS LOW
0105 0027          MOVWF  PORT_C        ; /
0106 0C00          MOVLW  B'00000000'    ;SET PORT C AS OUTPUTS
0107 0007          TRIS   PORT_C

;
0108 0C04          MOVLW  B'00000100'    ;SET UP PRESCALER
0109 0002          OPTION ; /

;
010A 0C60          MOVLW  MSEC5        ;RTCC = 5 mSEC
010B 0021          MOVWF  RTCC          ; /
010C 0068          CLRF   MSTMR        ;CLEAR MSTMR
010D 0069          CLRF   STMTR        ; & SEC TMR
010E 006A          CLRF   MTMR        ;& MINUTES
010F 0C12          MOVLW  12H          ;MAKE HRS = 12
0110 002B          MOVWF  HTMR        ; /
0111 002D          MOVWF  HALARM       ;MAKE HRS = 12
0112 006C          CLRF   MALARM       ; /
0113 0C03          MOVLW  B'00000011'    ;SET TO TEST MODE
0114 0030          MOVWF  FLAG          ; /
0115 0078          CLRF   ALFLAG       ;CLEAR ALL FLAG
0116 0079          CLRF   AAFLAG       ; /
0117 0077          CLRF   ENTFLG       ; /
0118 0A01          GOTO   TEST_HARDWARE

```

Multiplexing LEDs/Keypad

```

;
;All routines related to timer updates are located at
;address 200 and above.
        ORG     0200

;
UPDATE_TIMERS
0200 0201        MOVF     RTCC,W           ;SEE IF RTCC = 0
0201 0743        BTFSS    STATUS,Z        ;IF 0 THEN SKIP
0202 0A00        GOTO     UPDATE_TIMERS   ;ELSE LOOP
0203 0C60        MOVLW   MSEC5           ;RTCC = 5 mSEC
0204 0021        MOVWF   RTCC            ;
0205 02A8        INCF    MSTMR           ;INC 5 MILLI SEC
0206 06D0        BTFSC   FLAG,KEY_HIT    ;NO KEY HIT THEN SKIP
0207 0A70        GOTO     CHK_DE_BOUNCE   ;ELSE DEBOUNCE

UP_TMR_1
0208 0210        MOVF     FLAG,W         ;ALARM MODE?
0209 0E03        ANDLW   B'00000011'    ;
020A 0F01        XORLW   B'00000001'    ;
020B 0743        BTFSS    STATUS,Z        ;SKIP IF YES
020C 0A14        GOTO     UP_TMR_2       ;DO NEXT
020D 0550        BSF     FLAG,ALRMLED    ;LIGHT LED
020E 0570        BSF     FLAG,COLON      ;
020F 0C64        MOVLW   D'100'         ;IF 1/2 SEC
0210 0088        SUBWF   MSTMR,0        ;BLINK
0211 0703        BTFSS    STATUS,C        ;
0212 0450        BCF     FLAG,ALRMLED    ;ALARM LED
0213 0A19        GOTO     UP_TMR_3       ;SKIP

UP_TMR_2
0214 0570        BSF     FLAG,COLON      ;TURN ON
0215 0C64        MOVLW   D'100'         ;<100 BLINK COLON
0216 0088        SUBWF   MSTMR,0        ;
0217 0703        BTFSS    STATUS,C        ;YES THEN SKIP
0218 0470        BCF     FLAG,COLON      ;ELSE TURN OFF

UP_TMR_3
0219 0208        MOVF     MSTMR,0        ;GET MSTMR IN W
021A 0FC7        XORLW   D'199'         ;= 199 THEN SKIP
021B 0743        BTFSS    STATUS,Z        ;
021C 0800        RETLW   0              ;

;INC SECONDS COUNT
021D 0068        CLRF    MSTMR           ;CLEAR MS_TMR
021E 0216        MOVF     MIN_SEC,W      ;GET MIN_SEC TIMER
021F 0E0F        ANDLW   B'00001111'    ;MASK MINUTES
0220 0743        BTFSS    STATUS,Z        ;ZERO THEN SKIP
0221 00F6        DECF    MIN_SEC        ;REDUCE SECONDS
0222 0C09        MOVLW   STMR           ;LOAD FSR WITH S_TMR
0223 0024        MOVWF   FSR            ;
0224 0955        CALL    INC_60         ;INC SECONDS
0225 0D00        IORLW   0              ;DO AN OPERATION
0226 0743        BTFSS    STATUS,Z        ;IF RETURN = 0 SKIP
0227 0A38        GOTO     CHK_AL_TIM     ;CHK ALRM

;INC MINUTES COUNT
0228 03B6        SWAPF   MIN_SEC        ;SWAP MIN SEC
0229 0216        MOVF     MIN_SEC,W      ;GET MIN_SEC IN W
022A 0E0F        ANDLW   B'00001111'    ;MASK SECONDS
022B 0743        BTFSS    STATUS,Z        ;SKIP IF NOT SET
022C 00F6        DECF    MIN_SEC        ;ELSE DEC
022D 03B6        SWAPF   MIN_SEC        ;SWAP BACK
022E 0966        CALL    CHK_SILNC_TIM   ;SILNCE ON?
022F 0C0A        MOVLW   MTMR           ;INC MINUTES
0230 0024        MOVWF   FSR            ;
0231 0955        CALL    INC_60         ;
0232 0D00        IORLW   0              ;DO AN OPERATION
0233 0743        BTFSS    STATUS,Z        ;IF 0 THEN SKIP
0234 0A38        GOTO     CHK_AL_TIM     ;CHECK ALRAM TIME

;INC HOUR COUNT
0235 0C0B        MOVLW   HTMR           ;GET HTMR IN FSR
0236 0024        MOVWF   FSR            ;
0237 0989        CALL    INC_HR         ;INC HOURS
;

```

Multiplexing LEDs/Keypad

```

CHK_AL_TIM
0238 0718      BTFSS  ALFLAG,ALONOF  ;IF OFF QUIT
0239 0800      RETLW  0              ; /
023A 0658      BTFSS  ALFLAG,SILNC  ;RET IF IN SILENCE
023B 0800      RETLW  0
023C 0638      BTFSS  ALFLAG,INAL  ;ALREADY DONE
023D 0A4D      GOTO   CHK_1_MIN  ;SEE IF 1 MIN UP
;
;          RETLW  0              ;YES THEN QUIT
023E 020D      MOVF   HALARM,W      ;CHK HRS
023F 018B      XORWF  HTMR,W        ;EQUAL?
0240 0743      BTFSS  STATUS,Z      ;YES THEN SKIP
0241 0800      RETLW  0              ;ELSE RET
0242 020C      MOVF   MALARM,W      ;CHK MIN
0243 018A      XORWF  MTMR,W        ;EQUAL?
0244 0743      BTFSS  STATUS,Z      ;YES THEN SKIP
0245 0800      RETLW  0              ;ELSE RET
0246 0209      MOVF   STMR,W        ;SEE IF SEC=0
0247 0743      BTFSS  STATUS,Z      ;YES THEN SKIP
0248 0800      RETLW  0              ;NO THEN RET
0249 0538      BSF    ALFLAG,INAL  ;SET IN ALARM FLAG
024A 0C10      MOVLW  10           ;SET 1 MIN TIMER
024B 0036      MOVWF  MIN_SEC      ; /
024C 0800      RETLW  0
;
;          CHK_1_MIN
024D 0396      SWAPF  MIN_SEC,W      ;SWAP IN W
024E 0E0F      ANDLW  B'00001111'      ;CHK MINUTES
024F 0743      BTFSS  STATUS,Z      ;0 THEN SKIP
0250 0800      RETLW  0              ;ELSE RET
0251 0438      BCF    ALFLAG,INAL  ;CLR IN ALARM
0252 0478      BCF    ALFLAG,INAA  ;CLR IN AA
0253 0505      BSF    PORT_A,BEP    ;STOP BEEPER
0254 0800      RETLW  0
;
;          INC_60
0255 02A0      INCF   F0            ;INC AND GET IN W
0256 0200      MOVF   F0,0          ; /
0257 0E0F      ANDLW  B'00001111'      ;MASK HI BITS
0258 0F0A      XORLW  B'00001010'      ;= 10 THEN MAKE IT 0
0259 0743      BTFSS  STATUS,Z      ; /
025A 0801      RETLW  1              ;ELSE RETURN NON ZERO
025B 0CF0      MOVLW  B'11110000'      ;ZERO LSB
025C 0160      ANDWF  F0            ; /
025D 03A0      SWAPF  F0            ;SWAP INDIRECT
025E 02A0      INCF   F0            ;INC
025F 0200      MOVF   F0,0          ;GET IN W
0260 03A0      SWAPF  F0            ;SWAP F0 BACK
0261 0F06      XORLW  D'6'          ;=6 THEN SKIP
0262 0743      BTFSS  STATUS,Z      ; /
0263 0801      RETLW  1              ;ELSE RETURN NZ
0264 0060      CLRF  F0            ; /
0265 0800      RETLW  0              ;RET 0
;
;
;          CHK_SILNC_TIM
0266 0758      BTFSS  ALFLAG,SILNC  ;CHK IF IN SILENCE
0267 0800      RETLW  0              ;NO THEN SKIP
0268 0396      SWAPF  MIN_SEC,W      ;GET MIN IN W
0269 0E0F      ANDLW  B'00001111'      ;MASK SECS
026A 0743      BTFSS  STATUS,Z      ;ZERO?
026B 0800      RETLW  0              ;NO THEN RET
026C 0458      BCF    ALFLAG,SILNC  ;RESET SILENCE
026D 0C10      MOVLW  10           ;SET 1 MIN TIMER
026E 0036      MOVWF  MIN_SEC      ; /
026F 0800      RETLW  0
;

```

Multiplexing LEDs/Keypad

```

;
CHK_DE_BOUNCE
0270 06B7      BTFSC  ENTFLG,INKEYBEP ;IN KEY BEEP?
0271 0A76      GOTO   CHK_DEB_1      ;YES THEN DEC TIMER
0272 07B0      BTFSS  FLAG,KEY_BEEP ;KEY BEEP SET?
0273 0A7F      GOTO   CHK_SERV     ;NO, SEE IF SERVICED
0274 0678      BTFSC  ALFLAG,INAA   ;IN AA?
0275 0A86      GOTO   CHK_BEP_ON ;YES THEN SEE IF ON

CHK_DEB_1
0276 05B7      BSF    ENTFLG,INKEYBEP ;SET FLAG
0277 0215      MOVF  DEBOUNCE,W    ;GET IN W
0278 0643      BTFSC  STATUS,Z      ;NZ THEN SKIP
0279 0C14      MOVLW D'20'        ;ELSE DB 100 mSEC
027A 0035      MOVWF  DEBOUNCE     ;
027B 0405      BCF    PORT_A,BEP   ;TURN ON BEEPER
027C 02F5      DECFSZ DEBOUNCE ;DEC AND CHK
027D 0A08      GOTO   UP_TMR_1    ;GO BACK
027E 0505      BSF    PORT_A,BEP   ;TURN OFF BEEPER

CHK_SERV
;
;
027F 07F0      BTFSS  FLAG,SERVICED ;SERVICED THEN SKIP
0280 0A08      GOTO   UP_TMR_1      ;GO BACK
0281 04F0      BCF    FLAG,SERVICED ;ELSE CLEAR FLAGS
0282 04D0      BCF    FLAG,KEY_HIT  ;
0283 04B0      BCF    FLAG,KEY_BEEP ;RESET FLAG
0284 04B7      BCF    ENTFLG,INKEYBEP ;
0285 0A08      GOTO   UP_TMR_1    ;GO BACK

;
CHK_BEP_ON
0286 0705      BTFSS  PORT_A,BEP   ;IF OFF THEN SKIP
0287 0A08      GOTO   UP_TMR_1    ;ELSE WAIT
0288 0A76      GOTO   CHK_DEB_1 ;RETURN

;
;
INC_HR
0289 02A0      INCF  F0            ;INC HOUR TIMER
028A 0200      MOVF  F0,W          ;GET HR TMR IN W
028B 0031      MOVWF TEMP         ;SAVE IN TEMP
028C 0E0F      ANDLW B'00001111'  ;CHK LO BYTE = 10
028D 0F0A      XORLW D'10'        ;
028E 0743      BTFSS  STATUS,Z      ;YES THEN SKIP
028F 0A93      GOTO   INC_AM_PM    ;ELSE CHK 12
0290 0C10      MOVLW B'00010000' ;LOAD 1 IN MSB
0291 0020      MOVWF F0           ;
0292 0AA3      GOTO   RESTORE_AM_PM ;RESTORE AM/PM

INC_AM_PM
0293 04E0      BCF    F0,AM_PM     ;CLEAR AM/PM
0294 0200      MOVF  F0,W          ;GET IN W
0295 0F12      XORLW 12H          ;SEE IF 12 HEX
0296 0743      BTFSS  STATUS,Z      ;YES THEN SKIP
0297 0A9D      GOTO   CHK_13       ;ELSE CHK 13
0298 07F1      BTFSS  TEMP,AM_PM    ;IF SET, SKIP
0299 0A9C      GOTO   SET_AM_PM    ;ELSE SET
029A 04E0      BCF    F0,AM_PM     ;CLEAR FLAG
029B 0800      RETLW 0           ;RETURN

SET_AM_PM
029C 05E0      BSF    F0,AM_PM     ;SET FLAG

CHK_13
029D 0200      MOVF  F0,W          ;GET IN W
029E 0F13      XORLW 13H          ;SEE IF 13
029F 0743      BTFSS  STATUS,Z      ;YES THEN SKIP
02A0 0AA3      GOTO   RESTORE_AM_PM

SET_1_HR
02A1 0C01      MOVLW B'00000001'  ;SET TO 1
02A2 0020      MOVWF F0           ;

RESTORE_AM_PM
02A3 06F1      BTFSC  TEMP,AM_PM    ;SKIP IF AM
02A4 05E0      BSF    F0,AM_PM     ;ELSE SET TO PM
02A5 0800      RETLW 0

```



Multiplexing LEDs/Keypad

```

;
;
;
;          ORG      400
;
;KEY DEFINITIONS
000A          ALARM_KEY      EQU      0A
000B          CE_KEY         EQU      0B
000C          SNOOZE_KEY     EQU      0C
000D          AM_PM_KEY      EQU      0D
000E          CLR_ALARM_KEY  EQU      0E
000F          SET_KEY        EQU      0F
;
SERVICE_KEYS
0400 07D0     BTFS    FLAG,KEY_HIT      ;NO KEY HIT THEN ...
0401 0800     RETLW   0                  ;RETURN
0402 06F0     BTFSC   FLAG,SERVICED     ;IF NOT SERVICED SKIP
0403 0800     RETLW   0                  ;ELSE RETURN
0404 05F0     BSF     FLAG,SERVICED     ;SET SERVICED FLAG
0405 0210     MOVF    FLAG,W            ;GET MODE OF OPERATION
0406 0E03     ANDLW   B'00000011'      ;
0407 0643     BTFSC   STATUS,Z          ;00 THEN RTM
0408 0A10     GOTO    RTMKS             ;RTM KEY SERVICE
0409 0031     MOVWF   TEMP              ;SAVE IN TEMP
040A 02F1     DECF    TEMP              ;REDUCE TEMP
040B 0A0D     GOTO    SK1               ;SKIP
040C 0A1D     GOTO    ATMKS             ;01, DO ALARM MODE
;
SK1
040D 02F1     DECF    TEMP              ;REDUCE TEMP
040E 0800     RETLW   0                  ;11 THEN RETURN
040F 0A2A     GOTO    DEMKS             ;10, DATA ENTRY MODE
;
;REAL TIME MODE KEY SERVICE
RTMKS
0410 09BA     CALL    CHK_AL_KEYS       ;CHK ALARM KEYS
0411 0D00     IORLW   0                  ;SEE IF NZ RET
0412 0643     BTFSC   STATUS,Z          ;NZ THEN SKIP
0413 0800     RETLW   0                  ;ELSE RETURN
0414 0C0F     MOVLW   SET_KEY           ;SEE IF SET KEY
0415 0193     XORWF   NEW_KEY,W        ;
0416 0643     BTFSC   STATUS,Z          ;NO THEN SKIP
0417 0A91     GOTO    SERV_SET_RTM     ;SERVICE SET KEY
0418 0C0A     MOVLW   ALARM_KEY        ;ALARM KEY?
0419 0193     XORWF   NEW_KEY,W        ;
041A 0643     BTFSC   STATUS,Z          ;NO THEN SKIP
041B 0AAB     GOTO    SERV_ALARM_RTM   ;ELSE SERVICE ALARM
;
IGNORE_KEY
041C 0800     RETLW   0                  ;ELSE RETURN
;
;ALARM TIME MODE KEY SERVICE
ATMKS
041D 09BA     CALL    CHK_AL_KEYS       ;CHECK ALRM KEYS
041E 0D00     IORLW   0                  ;CHECK IF 0
041F 0643     BTFSC   STATUS,Z          ;NZ THEN SKIP
0420 0800     RETLW   0                  ;ELSE RETURN
0421 0C0F     MOVLW   SET_KEY           ;SEE IF SET KEY
0422 0193     XORWF   NEW_KEY,W        ;
0423 0643     BTFSC   STATUS,Z          ;NO THEN SKIP
0424 0A9C     GOTO    SERV_SET_ATM     ;GET ALARM KEY
0425 0C0A     MOVLW   ALARM_KEY        ;SEE IF HIT
0426 0193     XORWF   NEW_KEY,W        ;SEE IF HIT
0427 0643     BTFSC   STATUS,Z          ;NO THEN SKIP
0428 0AA2     GOTO    SERV_ALARM_ATM   ;ELSE SERVICE
0429 0A1C     GOTO    IGNORE_KEY

```


Multiplexing LEDs/Keypad

```

;
;DATA ENTRY MODE KEY SERVICE
DEMKS
042A 09BA      CALL    CHK_AL_KEYS    ;CHECK ALARM KEYS
042B 0D00      IORLW   0          ;CHK IF 0
042C 0643      BTFSC  STATUS,Z        ;NZ THEN SKIP
042D 0800      RETLW   0          ;ELSE RETURN
042E 0C0F      MOVLW   SET_KEY        ;IF SET KEY THEN END
042F 0193      XORWF   NEW_KEY,W      ;
0430 0643      BTFSC  STATUS,Z        ;NO THEN SKIP
0431 0A3F      GOTO    DEMKS_END    ;GOTO END
0432 0C0B      MOVLW   CE_KEY         ;IF CLEAR ENTRY
0433 0193      XORWF   NEW_KEY,W      ;
0434 0643      BTFSC  STATUS,Z        ;SKIP IF NO
0435 0A48      GOTO    DEMKS_END_1  ;ABANDON ENTRY
0436 0737      BTFSS  ENTFLG,HR10    ;10'S HRS DONE?
0437 0A54      GOTO    ENT_HR_10    ;NO THEN GET
0438 0757      BTFSS  ENTFLG,HR     ;HRS DONE?
0439 0A5F      GOTO    ENT_HRS       ;NO THEN GET
043A 0777      BTFSS  ENTFLG,MIN10   ;10'S MIN. DONE?
043B 0A72      GOTO    ENT_MIN_10   ;NO THEN GET
043C 0797      BTFSS  ENTFLG,MIN     ;MIN DONE?
043D 0A7F      GOTO    ENT_MIN      ;NO THEN GET
043E 0A87      GOTO    ENT_AM_PM    ;NO THEN GET

DEMKS_END
043F 0717      BTFSS  ENTFLG,RTATS   ;GET OLD STATUS
0440 0A4D      GOTO    LD_RTM        ;LOAD IN TIME
0441 020E      MOVF   MENTRY,W       ;LD IN ALARM
0442 002C      MOVWF  MALARM        ;
0443 020F      MOVF   HENTRY,W       ;
0444 002D      MOVWF  HALARM        ;
0445 0450      BCF   FLAG,ALRMLD    ;CLEAR FLAG
0446 0618      BTFSC  ALFLAG,ALONOF  ;SEE IF ON-OFF
0447 0550      BSF   FLAG,ALRMLD    ;ELSE SET

DEMKS_END_1
0448 0410      BCF   FLAG,0          ;RTM MODE
0449 0430      BCF   FLAG,1          ;
044A 0490      BCF   FLAG,FLASH    ;STOP FLASH

SERV_COM_RET
044B 05B0      BSF   FLAG,KEY_BEEP   ;
044C 0800      RETLW   0          ;RETURN

;
LD_RTM
044D 020E      MOVF   MENTRY,W       ;LD IN RTM
044E 002A      MOVWF  MTMR          ;
044F 020F      MOVF   HENTRY,W       ;
0450 002B      MOVWF  HTMR          ;
0451 0068      CLR   MSTMR         ;CLR TIME
0452 0069      CLR   STMR          ;
0453 0A48      GOTO    DEMKS_END_1  ;GO BACK

;
ENT_HR_10
0454 0213      MOVF   NEW_KEY,W       ;SEE IF 0
0455 0643      BTFSC  STATUS,Z        ;NZ THEN SKIP
0456 0A5C      GOTO    LD_HENTRY_0    ;LOAD 0
0457 02D3      DECF   NEW_KEY,0      ;1 THE SKIP
0458 0A1C      GOTO    IGNORE_KEY    ;ELSE IGNORE KEY
0459 058F      BSF   HENTRY,4          ;SET TO 1
045A 0537      BSF   ENTFLG,HR10    ;SET FLAG
045B 0A4B      GOTO    SERV_COM_RET  ;GO GET NEXT

LD_HENTRY_0
045C 048F      BCF   HENTRY,4          ;SET TO 0
045D 0537      BSF   ENTFLG,HR10    ;
045E 0A4B      GOTO    SERV_COM_RET  ;

```

Multiplexing LEDs/Keypad

```

ENT_HRS
045F 0C0F      MOVLW  HENTRY      ;USE INDIRECT ADDR.
0460 0024      MOVWF  FSR          ;
0461 068F      BTFS   HENTRY,4    ;SEE IF 0
0462 0A6D      GOTO  ALLOW0_2    ;YES THEN 0,1&2
0463 0C0A      MOVLW  D'10'       ;SEE IF 0 - 9
0464 0093      SUBWF  NEW_KEY,W   ;
0465 0603      BTFS   STATUS,C    ;IF C THEN SKIP
0466 0A1C      GOTO  IGNORE_KEY  ;ELSE IGNORE

ENT_LO_COM1
0467 0557      BSF   ENTFLG,HR   ;SET FLAG

ENT_LO_COM
0468 0200      MOVF  F0,W         ;LD HRS
0469 0EF0      ANDLW B'11110000' ;MASK LO NIBL
046A 0113      IORWF NEW_KEY,W   ;OR NEW KEY
046B 0020      MOVWF F0         ;SAVE BACK
046C 0A4B      GOTO  SERV_COM_RET ;GET NEXT

ALLOW0_2
046D 0C03      MOVLW  D'3'        ;SEE IF 0 - 2
046E 0093      SUBWF  NEW_KEY,W   ;
046F 0603      BTFS   STATUS,C    ;<3 THEN SKIP
0470 0A1C      GOTO  IGNORE_KEY  ;
0471 0A67      GOTO  ENT_LO_COM1 ;

;
ENT_MIN_10
0472 0C0E      MOVLW  MENTRY      ;DO INDIRECT ADDR.
0473 0024      MOVWF  FSR          ;
0474 0C06      MOVLW  D'6'        ;ALLOW 0 - 5
0475 0093      SUBWF  NEW_KEY,W   ;
0476 0603      BTFS   STATUS,C    ;IF C THEN SKIP
0477 0A1C      GOTO  IGNORE_KEY  ;ELSE IGNORE
0478 0380      SWAPF F0,W        ;SWAP AND GET
0479 0EF0      ANDLW B'11110000' ;MASK LO NIBL
047A 0113      IORWF NEW_KEY,W   ;OR NEW KEY
047B 0020      MOVWF F0         ;SAVE BACK
047C 03A0      SWAPF F0         ;SWAP BACK
047D 0577      BSF   ENTFLG,MIN10 ;
047E 0A4B      GOTO  SERV_COM_RET ;GET NEXT

;
ENT_MIN
047F 0C0E      MOVLW  MENTRY      ;DO INDIRECT
0480 0024      MOVWF  FSR          ;
0481 0C0A      MOVLW  D'10'       ;ALLOW 0 - 9
0482 0093      SUBWF  NEW_KEY,W   ;SEE IF >
0483 0603      BTFS   STATUS,C    ;NO THEN SKIP
0484 0A1C      GOTO  IGNORE_KEY  ;ELSE IGNORE
0485 0597      BSF   ENTFLG,MIN ;SET FLAG
0486 0A68      GOTO  ENT_LO_COM ;

;
ENT_AM_PM
0487 0C0D      MOVLW  AM_PM_KEY   ;AM/PM KEY?
0488 0193      XORWF  NEW_KEY,W   ;
0489 0743      BTFS   STATUS,Z    ;YES THEN SKIP
048A 0A1C      GOTO  IGNORE_KEY  ;
048B 07EF      BTFS   HENTRY,AM_PM ;TEST BIT
048C 0A8F      GOTO  SETAMPM     ;ELSE SET
048D 04EF      BCF   HENTRY,AM_PM ;CLEAR FLAG
048E 0A4B      GOTO  SERV_COM_RET ;GOTO END

SETAMPM
048F 05EF      BSF   HENTRY,AM_PM ;SET FLAG
0490 0A4B      GOTO  SERV_COM_RET ;

;
SERV_SET_RTM
0491 020A      MOVF  MTMR,W       ;TRANSFER TIME
0492 002E      MOVWF MENTRY      ;TO DATA ENTRY
0493 020B      MOVF  HTMR,W       ;
0494 002F      MOVWF HENTRY      ;

```

Multiplexing LEDs/Keypad

```

SERV_COM
0495 0210      MOVF    FLAG,W           ;SAVE IN W
0496 0E01      ANDLW   B'00000001'        ;ATM OR RTM MODE?
0497 0037      MOVWF   ENTFLG          ;SAVE IN ENTFLG
0498 0CF2      MOVLW   B'11110010'    ;FORCE 1S
0499 0130      IORWF   FLAG           ; /
049A 0410      BCF     FLAG,0         ; /
049B 0800      RETLW   0

;
SERV_SET_ATM
049C 020C      MOVF    MALARM,W          ;TRANSFER ALARM
049D 002E      MOVWF   MENTRY          ;TO DATA ENTRY
049E 020D      MOVF    HALARM,W          ; /
049F 002F      MOVWF   HENTRY          ; /
04A0 0518      BSF     ALFLAG,ALONOF      ;SET FLAG
04A1 0A95      GOTO    SERV_COM          ;GOTO COMMON

;
SERV_ALARM_ATM
04A2 0718      BTFSS  ALFLAG,ALONOF      ;TEST ON/OFF
04A3 0AA6      GOTO    SET_ALONOF        ;SET ON/OFF FLG
04A4 0418      BCF     ALFLAG,ALONOF      ;CLEAR FLAG
04A5 0AA7      GOTO    SERV_ATM_COM      ;RET THRO COM

SET_ALONOF
04A6 0518      BSF     ALFLAG,ALONOF      ;SET FLAG

SERV_ATM_COM
04A7 05B0      BSF     FLAG,KEY_BEEP      ;BEEP
04A8 0CF0      MOVLW   B'11110000'        ;CLEAR SEC COUNT
04A9 0176      ANDWF   MIN_SEC           ; /
04AA 0800      RETLW   0                 ;RETURN

;
SERV_ALARM_RTC
04AB 05B0      BSF     FLAG,KEY_BEEP      ;SET BEEP FLAG
04AC 0510      BSF     FLAG,0            ;SET TO ALARM TIME
04AD 0430      BCF     FLAG,1            ; /
04AE 0C05      MOVLW   D'05'             ;SAVE 5 IN MIN_SEC
04AF 0036      MOVWF   MIN_SEC           ; /
04B0 0800      RETLW   0

;
SERV_SNOOZE
04B1 0CA0      MOVLW   0A0               ;SNOOZE FOR 10 MINS
04B2 0036      MOVWF   MIN_SEC           ; /
04B3 0558      BSF     ALFLAG,SILNC      ;SET FLAG

CLR_AL_COM
04B4 05B0      BSF     FLAG,KEY_BEEP      ;SET BEEP FLAG
04B5 007A      CLRWF   AATMR             ;RESET AA TIMER
04B6 0079      CLRWF   AAFLAG           ;CLEAR AA FLAGS
04B7 0478      BCF     ALFLAG,INAA       ;RESET INAA FLAG
04B8 0505      BSF     PORT_A,BEP        ;TURN OFF BEEPER
04B9 0800      RETLW   0                 ;RET

;
CHK_AL_KEYS
04BA 0718      BTFSS  ALFLAG,ALONOF      ;ALARM ON?
04BB 0801      RETLW   1                 ;NO THEN RET
04BC 0738      BTFSS  ALFLAG,INAL       ;IN ALARM?
04BD 0801      RETLW   1                 ;NO THEN SKIP
04BE 0C0E      MOVLW   CLR_ALARM_KEY     ;CHECK IF CLR ALARM
04BF 0193      XORWF   NEW_KEY,W         ; /
04C0 0643      BTFSC  STATUS,Z           ;NO THEN SKIP
04C1 0AC7      GOTO    CLR_ALARM         ;ELSE CLEAR ALARM
04C2 0C0C      MOVLW   SNOOZE_KEY       ;SEE IF SNOOZE HIT
04C3 0193      XORWF   NEW_KEY,W         ; /
04C4 0743      BTFSS  STATUS,Z           ;YES THEN SKIP
04C5 0801      RETLW   1
04C6 0AB1      GOTO    SERV_SNOOZE

```

Multiplexing LEDs/Keypad

```

;
CLR_ALARM
04C7 0438      BCF      ALFLAG,INAL      ;CLEAR ALARM
04C8 0458      BCF      ALFLAG,SILNC     ;CLEAR SILENCE
04C9 0C0F      MOVLW   B'00001111'     ;CLEAR MINS
04CA 0176      ANDWF   MIN_SEC          ; /
04CB 0AB4      GOTO    CLR_AL_COM

;
          ORG      600
;If the AA alarm is set, then this routine takes care of
;the timing in sounding the alarm.
;
SOUND_AA
0600 0738      BTFSS   ALFLAG,INAL      ;SKIP IF IN ALRM
0601 0800      RETLW   0              ;ELSE RETURN
0602 0658      BTFSC   ALFLAG,SILNC     ;SKIP IF NOT IN SIL
0603 0800      RETLW   0              ;ELSE RET
0604 06B7      BTFSC   ENTFLG,INKEYBEP  ;SKIP IF NOT IN KEY BEP
0605 0A55      GOTO    CHK_COLSN      ;CHK COLLISION

SND_AA_0
0606 0778      BTFSS   ALFLAG,INAA     ;SKIP IF IN AA

SND_AA_1
0607 0919      CALL    INIT_AA          ;INIT ALL
0608 0719      BTFSS   AAFLAG,0        ;SKIP IF DONE
0609 0A21      GOTO    DO_CYCL0      ;DO FIRST CYCL
060A 0739      BTFSS   AAFLAG,1        ;SKIP IF DONE
060B 0A29      GOTO    DO_CYCL1      ;ELSE 2ND CYCLE
060C 0759      BTFSS   AAFLAG,2        ;SKIP IF DONE
060D 0A31      GOTO    DO_CYCL2      ;ELSE DO 3RD CYCLE
060E 0779      BTFSS   AAFLAG,3        ;SKIP IF DONE
060F 0A39      GOTO    DO_CYCL3      ;DO CYCLE 4
0610 0799      BTFSS   AAFLAG,4        ;SKIP IF DONE
0611 0A3E      GOTO    DO_CYCL4      ;DO CYCLE 5
0612 07B9      BTFSS   AAFLAG,5        ;SKIP IF DONE
0613 0A43      GOTO    DO_CYCL5      ;DO CYCLE 6
0614 07D9      BTFSS   AAFLAG,6        ;SKIP IF DONE
0615 0A48      GOTO    DO_CYCL6      ;DO CYCLE 6
0616 07F9      BTFSS   AAFLAG,7        ;SKIP IF DONE
0617 0A50      GOTO    DO_CYCL7      ;DO CYCLE 7
0618 0A07      GOTO    SND_AA_1      ;GO BACK

;
INIT_AA
0619 0079      CLRF    AAFLAG          ;CLEAR ALL FLAGS
061A 0578      BSF     ALFLAG,INAA     ;SET IN AA FLAG
061B 0A2D      GOTO    PUT_ON_100    ;ON 100 MSECS

;
DEC_AA_TMR
061C 00FA      DECF    AATMR          ;REDUCE TIMER
061D 021A      MOVF    AATMR,W        ;GET IN W
061E 0743      BTFSS   STATUS,Z        ;CHECK IF Z
061F 0801      RETLW   1              ;NO THEN NZ
0620 0800      RETLW   0              ;ELSE 0

;
DO_CYCL0
0621 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
0622 0743      BTFSS   STATUS,Z        ;IF NZ THEN RET
0623 0800      RETLW   0              ;
0624 0519      BSF     AAFLAG,0        ;SET DONE FLAG

PUT_OFF_100
0625 0505      BSF     PORT_A,BEP        ;TURN OFF BEEPER
0626 0C14      MOVLW   D'20'          ;FOR 100 MSECS
0627 003A      MOVWF   AATMR          ; /
0628 0800      RETLW   0

```

Multiplexing LEDs/Keypad

```
;
DO_CYCL1
0629 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
062A 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
062B 0800      RETLW  0                ;
062C 0539      BSF     AAFLAG,1      ;SET DONE FLAG
                PUT_ON_100
062D 0405      BCF     PORT_A,BEP      ;TURN ON BEEPER
062E 0C14      MOVLW  D'20'          ;FOR 100 MSECS
062F 003A      MOVWF  AATMR          ;
0630 0800      RETLW  0                ;

;
DO_CYCL2
0631 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
0632 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
0633 0800      RETLW  0                ;
0634 0559      BSF     AAFLAG,2      ;SET DONE FLAG
0635 0505      BSF     PORT_A,BEP      ;TURN OFF BEEPER
0636 0C64      MOVLW  D'100'        ;FOR 500 MSECS
0637 003A      MOVWF  AATMR          ;
0638 0800      RETLW  0                ;

;
DO_CYCL3
0639 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
063A 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
063B 0800      RETLW  0                ;
063C 0579      BSF     AAFLAG,3      ;SET DONE FLAG
063D 0A2D      GOTO    PUT_ON_100     ;DO NEXT CYCLE

;
DO_CYCL4
063E 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
063F 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
0640 0800      RETLW  0                ;
0641 0599      BSF     AAFLAG,4      ;SET DONE FLAG
0642 0A25      GOTO    PUT_OFF_100    ;DO NEXT CYCLE

;
DO_CYCL5
0643 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
0644 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
0645 0800      RETLW  0                ;
0646 05B9      BSF     AAFLAG,5      ;SET DONE FLAG
0647 0A2D      GOTO    PUT_ON_100     ;DO NEXT CYCLE

;
DO_CYCL6
0648 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
0649 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
064A 0800      RETLW  0                ;
064B 05D9      BSF     AAFLAG,6      ;SET DONE FLAG
064C 0505      BSF     PORT_A,BEP      ;TURN OFF BEEPER
064D 0CC8      MOVLW  D'200'        ;FOR 1000 MSECS
064E 003A      MOVWF  AATMR          ;
064F 0800      RETLW  0                ;

;
DO_CYCL7
0650 091C      CALL    DEC_AA_TMR      ;REDUCE TIMER
0651 0743      BTFSS  STATUS,Z        ;IF NZ THEN RET
0652 0800      RETLW  0                ;
0653 05F9      BSF     AAFLAG,7      ;SET DONE FLAG
0654 0A2D      GOTO    PUT_ON_100     ;DO NEXT CYCLE
```

Multiplexing LEDs/Keypad

```

;
CHK_COLSN
0655 0605      BTFSC  PORT_A,BEP      ;IF ON THEN SKIP
0656 0A06      GOTO   SND_AA_0      ;ELSE RET
0657 021A      MOVF   AATMR,W        ;GET TIMER
0658 0643      BTFSC  STATUS,Z      ;NZ THEN SKIP
0659 0A5C      GOTO   LD_AAT_1      ;LOAD A 1 IN TMR
065A 00FA      DECF   AATMR         ;REDUCE TIMER
065B 0800      RETLW  0              ;RETURN

LD_AAT_1
065C 02BA      INCF   AATMR         ;INC TIMER
065D 0800      RETLW  0              ;RET

;
ORG    PIC57
SYS_RESET
07FF 0A00      GOTO   START

;
END
```

```
Errors   :    0
Warnings :    0
```

Implementing a Simple Serial Mouse Controller

INTRODUCTION

The mouse is becoming increasingly popular as a standard pointing data entry device. It is no doubt that the demand of the mouse is increasing. Various kinds of mice can be found in the market, including optical mouse, opto-mechanical mouse, and its close relative, trackball. The mouse interfaces to the host via an RS232 port or a dedicated interface card. Their mechanisms are very similar. The major electrical components of a mouse are:

- Microcontroller
- Photo-transistors
- Infrared emitting diode
- Voltage conversion circuit

The intelligence of the mouse is provided by the microcontroller, hence the features and performance of a mouse is greatly related to the microcontroller used.

This application note describes the implementation of a serial mouse using the PIC16C54. The PIC16C54 is a high speed 8-bit CMOS microcontroller offered by Microchip Technology, Inc. It is an ideal candidate for a mouse controller.

THEORY OF OPERATION

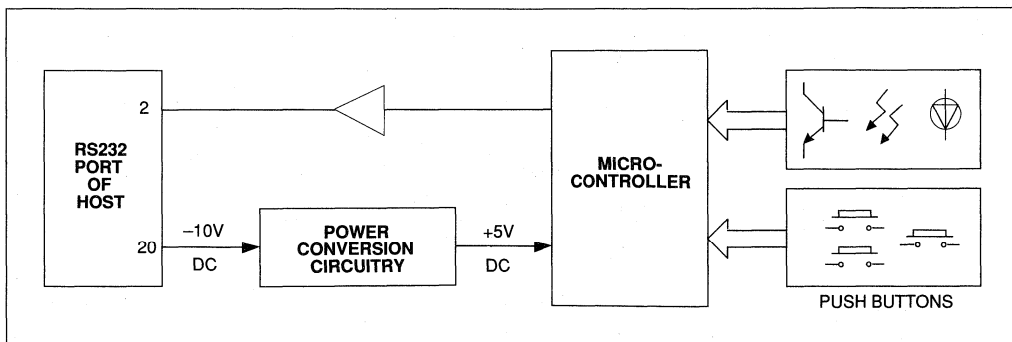
A mouse can be divided into several functional blocks:

- Microcontroller
- Button detection
- Motion detection
- RS232 signal generation
- 5V DC power supply unit

A typical functional block diagram is shown in Figure 1.

In Figure 2, three push buttons are connected to the input ports of the PIC16C54. When a switch opening or closure is detected, a message is formatted and sent to the host. The X and Y movements are measured by counting the pulses generated by the photo-couplers. In the case of an opto-mechanical mouse, the infrared light emitted by the rotating wheel is blocked by the photo-transistor side, so that the pulses are generated on the photo-transistor side. In case of an optical mouse, the infrared light emitted by the infrared diode is reflected off the reflective pad patterned with vertical and horizontal grid lines. It is then received by the photo-transistor in the mouse. When any X or Y movement is detected, a message is formatted and sent to the host.

FIGURE 1 - FUNCTIONAL BLOCKS OF A SERIAL MOUSE



Mouse Controller

The Microsoft® Mouse System and the Mouse Systems® device both use serial input techniques. The Mouse System protocol format contains five bytes of data. One byte describes the status of three push buttons, two bytes for the relative X movements and two bytes for the relative Y movements. The Microsoft protocol format contains three bytes of data describing the status of two push buttons and the relative X and Y movements. The details of these protocols are given in Table 1.

Three lines are connected to the host via the RS232 port:

- Signal Ground
- Received Data
- Request to Send

“Received Data” carries the message sent by the mouse. While “Request to Send” provides a -10V DC for voltage conversion circuitry. A voltage of +5V DC is required for electronic components inside the mouse, however, +5V DC is not part of an RS232 port, so voltage conversion circuitry is required. This circuit is typically composed of a 555 timer, Zener diodes, and capacitors. An example circuit is shown in Figure 3. Since the current supplied through the RS232 port is limited to 10 mA, the mouse cannot be designed to consume more than 10 mA current unless an external power supply is provided. The PIC16C54, running at 4 MHz (1 μ s instruction cycle) can provide a very high tracking speed. An 8 MHz version of PIC16C54 is also available if higher performance is desired.

FIGURE 2 - PIC16C54 PIN ASSIGNMENT

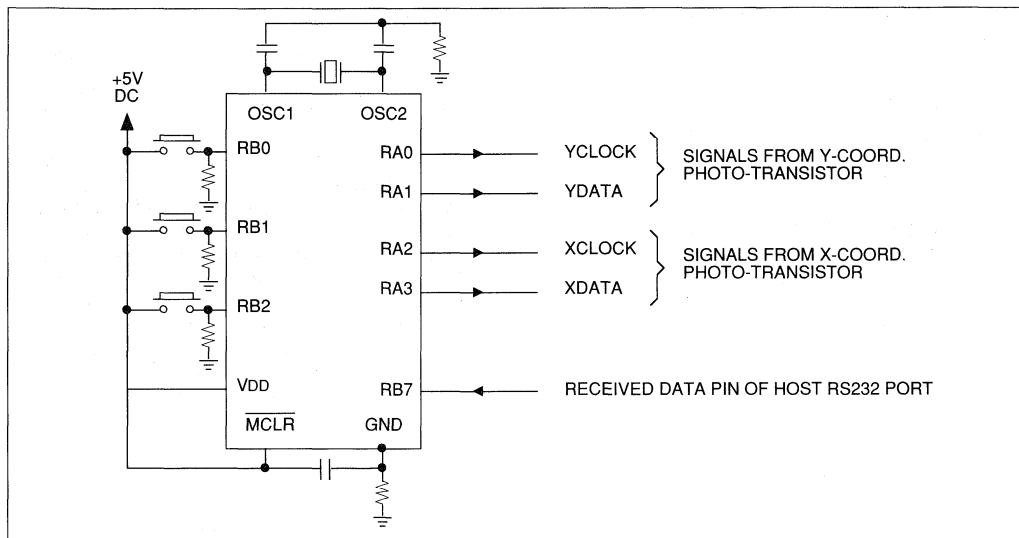
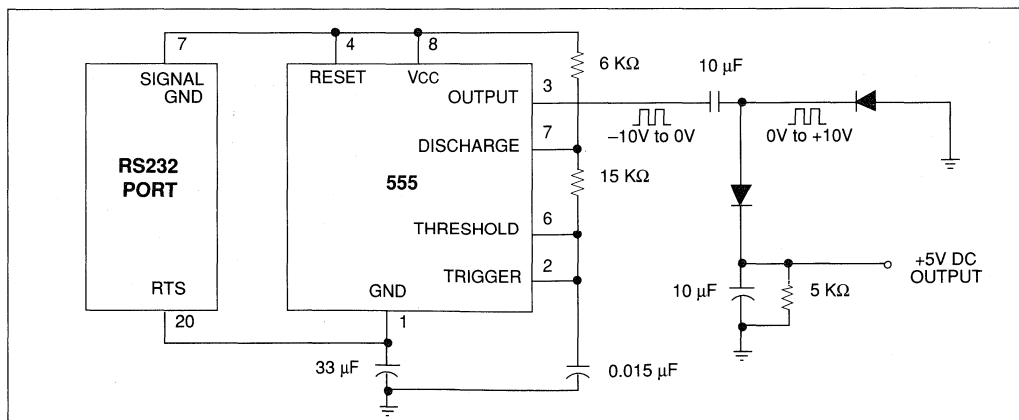


FIGURE 3 - VOLTAGE CONVERSION CIRCUITRY



ABOUT THE SOFTWARE

The major tasks performed by the software are button scanning, X and Y motion scanning, formatting and sending serial data to the host. These tasks need to be performed in parallel in order to gain better tracking speed. The pulses generated by the photo-couplers are counted while transmitting the serial signals to the RS232 port. The number of pulses reflects the speed of the movement. The more number of pulses, the faster the movement is.

The directions of the movement are determined by the last states and the present states of the outputs of the photo-transistors. In Figure 4, XCLOCK and XDATA are outputs from the photo-transistors corresponding to the

X-axis movement. XDATA is read when a rising or a falling edge of XCLOCK is detected. For right movement, XDATA is either LOW at the rising edge of XCLOCK or HIGH at the falling edge of XCLOCK. The up and down movement detections follow the same logic. In Table 1, X7:X0 are data for relative movement. If X is positive, it implies that the mouse is moving to the right. If X is negative, it implies a movement to the left. Similarly, if Y is positive, it indicates that the mouse is moving down and if Y is negative, it indicates that the mouse is moving up. The pulses generated by the photo-couplers are checked before every bit is sent. A bit takes 1/1200 second to send, if the distance between the grid lines is 1 mm, the tracking speed will be up to 1200 mm/second.

FIGURE 4 - VOLTAGE CONVERSION CIRCUITRY

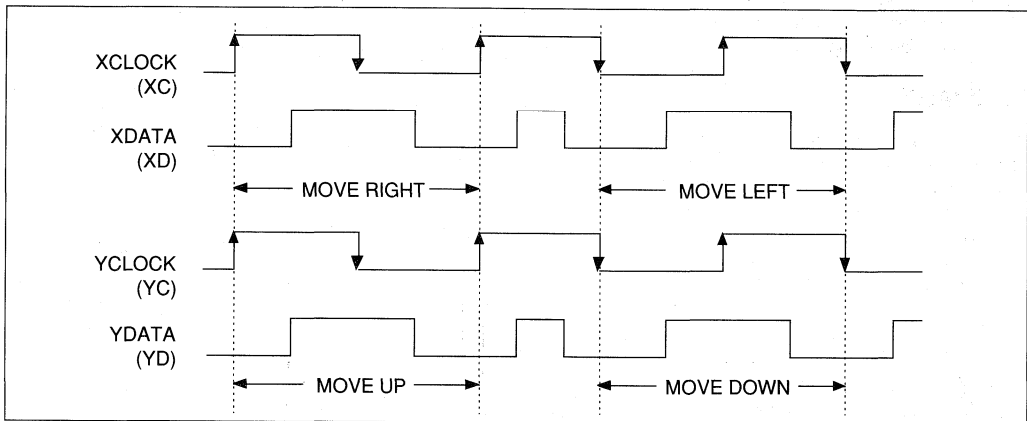


TABLE 1 - MOUSE SYSTEM AND MICROSOFT PROTOCOLS

Bit Position	Mouse System Format*								Microsoft Format*							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R	1	1	L	R	Y7	Y6	X7	X6
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0	0	0	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0	0	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0								
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0								

* L = Left Key Status 1 = Pressed X7-X0 = X-Axis Movement Data
M = Middle Key Status 0 = Released Y7-Y0 = Y-Axis Movement Data
R = Right Key Status



Mouse Controller

The buttons are scanned after a message is sent and the time used to send the message is used as the debouncing time. The message is in an RS232 format with 1200 baud, eight data bits, no parity, and two stop bits.

The flow charts of the main program, subroutine BYTE and subroutine BIT are shown in Figures 5, 6, and 7. Figure 5 shows the Trigger Flag is set when any change of button status or X/Y movement is detected. Subroutine BYTE is called in the main program five times to send five bytes of information. Subroutine BYTE controls the status of the "Received Data" (RD) pin. If Trigger Flag is clear, RD will always be HIGH. Hence, no message will be sent even when subroutine BYTE is called. Figure 7 shows that subroutine BIT counts the number of pulses from outputs of the photo-transistors, determines the directions, and generates 1/1200 second delay to get 1200 baud timing.

The mouse has been tested in Mouse System Mode and is functioning properly. A completed listing of the source program is given in Appendix A.

SUMMARY

The PIC16C54 from Microchip Technology, Inc. provides a very cost-effective, high performance mouse implementation. Its low power (typically < 2 mA at 1 μs instruction cycle), small package (18-pin) and high reliability (on-chip watchdog timer to prevent software hang-ups) are among several reasons why the PIC16C54 is uniquely suitable for mouse applications.

This application note provides the user with a simple, fully functional serial mouse implementation. The user may use this as a starting point for a more comprehensive design. For fully implemented and compliant mouse products see Microchip's ASSP device family (MTA41XXX).

FIGURE 5 - FLOW CHART OF THE MAIN PROGRAM

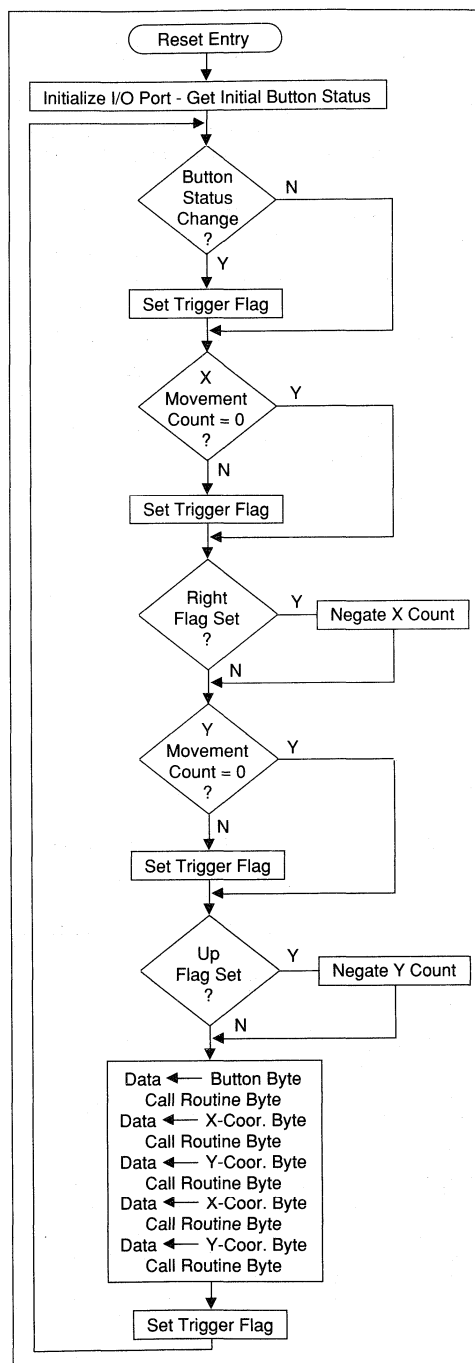
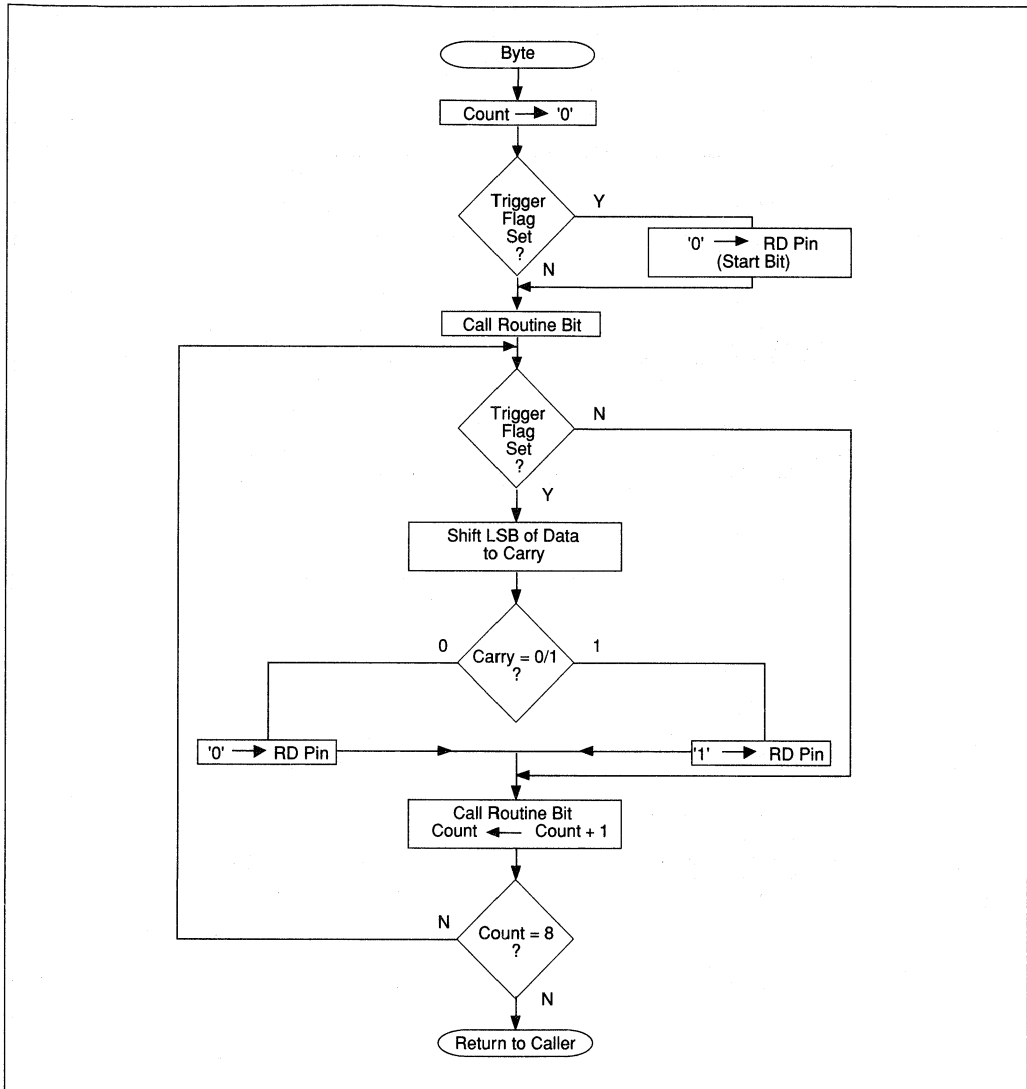
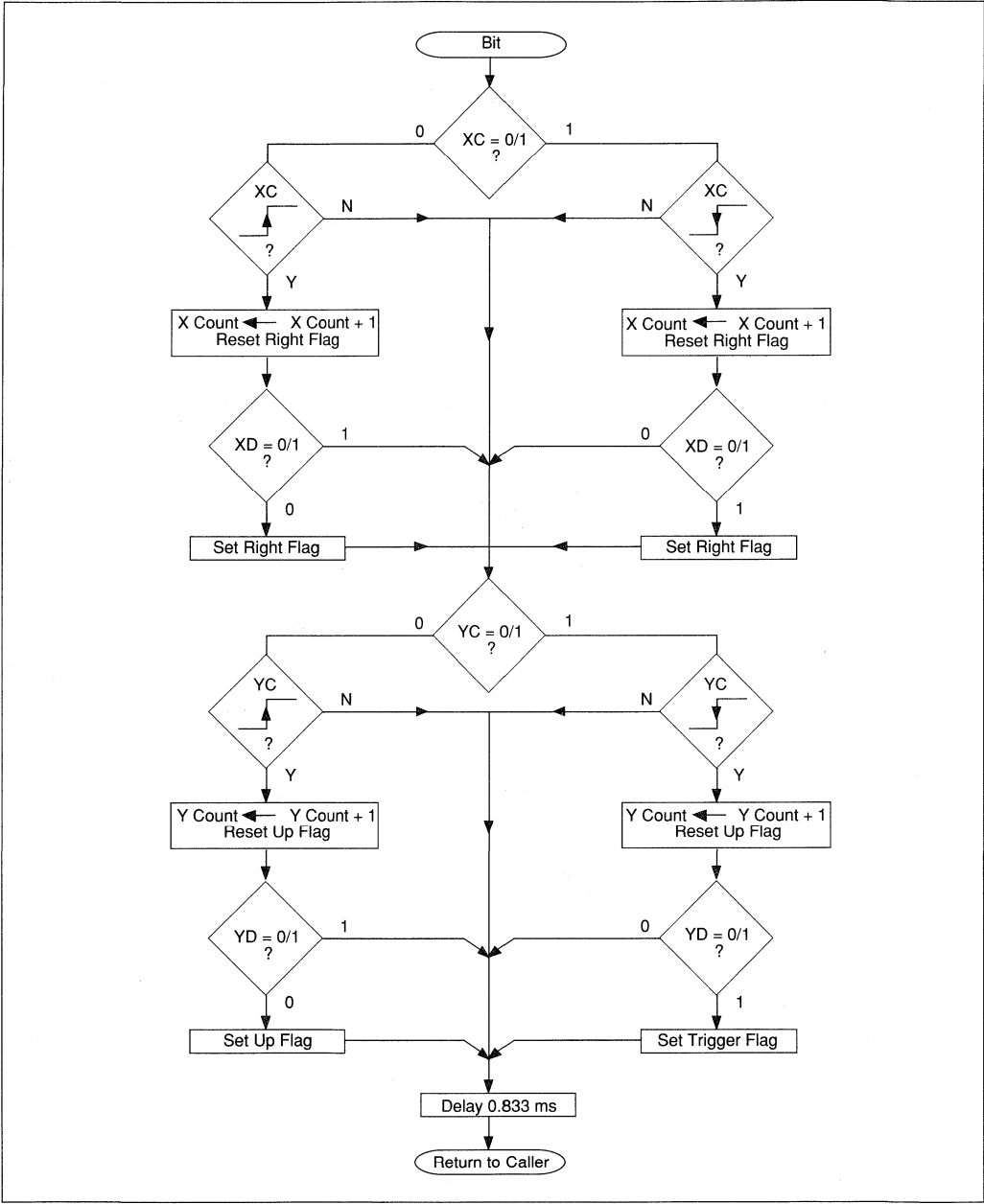


FIGURE 6 - FLOW CHART OF ROUTINE BYTE



Mouse Controller

FIGURE 7 - FLOW CHART OF ROUTINE BIT



APPENDIX A:

MPASM B0.54
MOUSE

PAGE 1

```
TITLE " MOUSE "
LIST P=16C54,R=0
;
;*****
;*
;* MOUSE CONTROLLER
;*
;*
;* VERSION : 25 APRIL, 1990
;*
;*
;* MODE = PIC16C54XT CLK=4.0MHZ
;*
;*****
;
;-----
;
; FILES ASSIGNMENT
;-----
;
0003 STATUS EQU 3 ;STATUS REGISTER
0005 RA EQU 5 ;I/O PORT A
0006 RB EQU 6 ;I/O PORT B
0008 TIMER1 EQU 10 ;COUNTER FOR DELAY
000C CSTAT EQU 14 ;CO-ORDINATE STATUS
000D BSTAT EQU 15 ;BUTTON STATUS
000E DATA0 EQU 16 ;
000F DATA1 EQU 17 ;
0010 DATA2 EQU 20 ;5 BYTE RS232 DATA
0011 DATA3 EQU 21 ;
0012 DATA4 EQU 22 ;
0013 FLAGA EQU 23 ;GENERAL PURPOSE FLAG
0014 XCOUNT EQU 24 ;X-MOVEMENT COUNTER
0015 YCOUNT EQU 25 ;Y-MOVEMENT COUNTER
0016 FLAGB EQU 26 ;GENERAL PURPOSE FLAG
0018 COUNT EQU 30 ;GENERAL PURPOSE COUNTER
0019 DATA_AREA EQU 31 ;FOR TEMP. STORAGE
;
;-----
;
; BIT ASSIGNMENT
;-----
;
0000 YC EQU 0 ;Y-CLOCK PIN
0001 YD EQU 1 ;Y-DATA PIN
0001 UP EQU 1 ;MOVING UP FLAG
0002 XC EQU 2 ;X-CLOCK PIN
0003 XD EQU 3 ;X-DATA PIN
0003 RI EQU 3 ;MOVING RIGHT FLAG
0000 BU1 EQU 0 ;BUTTON #1 PIN
0002 BU2 EQU 2 ;BUTTON #2 PIN
0000 CA EQU 0 ;CARRY FLAG
0007 RD EQU 7 ;RECEIVED DATA PIN TO RS232
0002 ZERO_AREA EQU 2 ;ZERO FLAG
0002 TR EQU 2 ;TIGGER FLAG
;
;=====
;
; SUBROUTINES
;=====
;
;*****
;
; ORG 0
;*****
;
;=====
;
; DELAY A BIT TIME AND CHECK XC & YC STATUS
;=====
```

Mouse Controller

```

BIT
0000 0745      BTFSS   RA,XC          ;XC = 1 ?
0001 0A0A      GOTO    BIT0
0002 064C      BTFSC   CSTAT,XC        ;(XC=1)
0003 0A11      GOTO    BITY          ;(XC ALWAYS = 1)
0004 02B4      INCF    XCOUNT        ;(XC -|__ )
0005 0476      BCF     FLAGB,RI        ;DEFAULT LEFT
0006 0765      BTFSS   RA,XD          ;LEFT / RIGHT ?
0007 0A11      GOTO    BITY
0008 0576      BSF     FLAGB,RI
0009 0A11      GOTO    BITY

BIT0
000A 074C      BTFSS   CSTAT,XC        ;(XC=0)
000B 0A11      GOTO    BITY          ;(XC ALWAYS = 0)
000C 02B4      INCF    XCOUNT        ;(XC __| -)
000D 0476      BCF     FLAGB,RI        ;DEFAULT LEFT
000E 0665      BTFSC   RA,XD          ;LEFT / RIGHT ?
000F 0A11      GOTO    BITY
0010 0576      BSF     FLAGB,RI

BITY
0011 0705      BTFSS   RA,YC          ;YC = 1 ?
0012 0A1B      GOTO    BITY0
0013 060C      BTFSC   CSTAT,YC        ;(YC=1)
0014 0A22      GOTO    BITDY          ;(YC ALWAYS = 1)
0015 02B5      INCF    YCOUNT        ;(YC -|__ )
0016 0436      BCF     FLAGB,UP        ;DEFAULT DOWN
0017 0725      BTFSS   RA,YD          ;DOWN / UP ?
0018 0A22      GOTO    BITDY
0019 0536      BSF     FLAGB,UP
001A 0A22      GOTO    BITDY

BITY0
001B 070C      BTFSS   CSTAT,YC        ;(YC=0)
001C 0A22      GOTO    BITDY          ;(YC ALWAYS = 0)
001D 02B5      INCF   Y   COUNT        ;(YC __| -)
001E 0436      BCF     FLAGB,UP        ;DEFAULT DOWN
001F 0625      BTFSC   RA,YD          ;DOWN / UP ?
0020 0A22      GOTO    BITDY
0021 0536      BSF     FLAGB,UP

BITDY
0022 0205      MOVF    RA,W            ;SAVE COOR. STATUS
0023 002C      MOVWF  CSTAT
0024 0C11      MOVLW  193D            ;0.833 MS DELAY
0025 0028      MOVWF  TIMER1

BITD0
0026 0000      NOP
0027 02E8      DECF   SZ   TIMER1
0028 0A26      GOTO   BITD0
0029 0800      RETLW  0

;
;=====
;
;*****
;*      SUBROUTINE TO SEND A BYTE      *
;*      AS RS232C FORMAT 8,N,1        *
;*****
;
;
BYTE
002A 0078      CLRF   COUNT          ;RESET 8 BIT COUNT
002B 0753      BTFSS  FLAGA,TR        ;ANY TRIGGER
002C 0A2E      GOTO   BYTE0
002D 04E6      BCF    RB,RD          ;LOW RD FOR START BIT

BYTE0
002E 0900      CALL  BIT

BYTE1
002F 0753      BTFSS  FLAGA,TR        ;ANY TRIGGER ?
0030 0A37      GOTO   BYTE3
0031 0339      RRF    DATA_AREA     ;SHIFT DATA TO CARRY
0032 0703      BTFSS  STATUS,CA      ;0 / 1 ?
0033 0A36      GOTO   BYTE2

```

Mouse Controller

```

0034 05E6          BSF    RB,RD          ;SEND A 1
0035 0A37          GOTO   BYTE2
;
0036 04E6          BCF    RB,RD          ;SEND A 0
;
;BYTE2
0037 0900          CALL   BIT
0038 02B8          INCF   COUNT
0039 0778          BTFSS COUNT,3        ;COUNT = 8 ?
003A 0A2F          GOTO   BYTE1
003B 0753          BTFSS FLAGA,TR      ;ANY TRIGGER ?
003C 0A42          GOTO   BYTE4
003D 04E6          BCF    RB,RD          ;SEND SENT BIT
003E 0900          CALL   BIT
003F 05E6          BSF    RB,RD
0040 0900          CALL   BIT
0041 0A44          GOTO   BYTE5
;
;BYTE4
0042 0900          CALL   BIT
0043 0900          CALL   BIT
;
;BYTE5
0044 0800          RETLW 0
;
;=====
;          RESET ENTRY
;=====
;
;
INIT
0045 0CC1          MOVLW B'11000001'    ;DISABLE WATCH DOG
0046 0002          OPTION
0047 0C0F          MOVLW B'00001111'    ;INIT RB0-3 BE INPUTS
0048 0006          TRIS  RB          ;RB4-7 BE OUTPUTS
0049 0CFE          MOVLW B'11111111'    ;INIT RA0-3 BE INPUTS
004A 0005          TRIS  RA
004B 05E6          BSF    RB,RD          ;HIGH RD PIN
004C 0246          COMF  RB,W          ;GET INIT BUTTON INPUTS
004D 0E05          ANDLW B'00000101'
004E 0D80          IORLW B'10000000'
004F 002D          MOVWF BSTAT
0050 002E          MOVWF DATA0
0051 0205          MOVF  RA,W
0052 002C          MOVWF CSTAT
0053 0073          CLRF  FLAGA          ;CLEAR TR FLAG
0054 0074          CLRF  XCOUNT        ;RESET XCOUNT & YCOUNT
0055 0075          CLRF  YCOUNT
;
SCAN
0056 006F          CLRF  DATA1          ;UPDATE X,Y MOVEMENT DATA
0057 0070          CLRF  DATA2
0058 0071          CLRF  DATA3
0059 0072          CLRF  DATA4
005A 0214          MOVF  XCOUNT,W      ;XCOUNT = 0 ?
005B 0743          BTFSS STATUS,ZERO_AREA
005C 0A80          GOTO  WRITX
;
SCANA
005D 0215          MOVF  YCOUNT,W      ;YCOUNT = 0 ?
005E 0743          BTFSS STATUS,ZERO_AREA
005F 0A92          GOTO  WRITY
;
SCANB
0060 0246          COMF  RB,W          ;BUTTON STATUS CHANGE ?
0061 0E05          ANDLW B'00000101'
0062 0D80          IORLW B'10000000'
0063 00AD          SUBWF BSTAT
0064 0643          BTFSC STATUS,ZERO_AREA ;IF CHANGE THEN TRIGGER
0065 0A6B          GOTO  SCANC          ;(NO CHANGE)
0066 0553          BSF    FLAGA,TR      ;(CHANGE) SET TRIGGER FLAG
0067 0246          COMF  RB,W          ;FORMAT BUTTON STATUS DATA
0068 0E05          ANDLW B'00000101'
0069 0D80          IORLW B'10000000'

```


Mouse Controller

```

                                WRITV
009F 0C40                      MOVLW 40H                ;YCOUNT <- 64
00A0 0035                      MOVWF YCOUNT
00A1 0A97                      GOTO WRITW
                                ;
                                WRITD
00A2 0215                      MOVF YCOUNT,W      ; (DOWN)
00A3 0A9B                      GOTO WRITB
                                ;
                                ;=====
                                ;      RESET ENTRY
                                ;=====
                                ;
                                ORG 777
01FF 0A45                      GOTO I NIT          ;JUMP TO PROGRAM STARTING
                                ;
                                END
                                ;
                                ;*****
```

```
Errors : 0
Warnings : 0
```

Mouse Controller

NOTES:

Intelligent Remote Positioner (Motor Control)

INTRODUCTION

The excellent cost/performance ratio of the PIC16C5X are well suited for a low-cost proportional d.c. actuator controller. This application note depicts a design of a remote intelligent positioning system using a d.c. motor (up to 1/3 hp) run from 12 to 24 V. The position accuracy is 1 in 8 bits or 0.4%. The PIC16C5X receives its command and control information via a MICROWIRE™ serial bus. However, any serial communication method is applicable.

IMPLEMENTATION

The PIC16C5X based controller receives commands from a host, compares them to the actual position, calculates the desired motor drive level and then pulses a full H-bridge (Figure 2). In this way it serves as a remote intelligent positioner, driving the load until it has reached the commanded position. It can be used to control any proportional d.c. actuator i.e. d.c. motor or proportional valve.

This system is ideally suited to remotely position valves and machinery. It can be used with d.c. motors to easily automate manual equipment. Because of the 5 wire serial interface, the positioner can be installed near its power supply and load. The remote intelligent positioner can then be linked to the central control processor by a small diameter easily routed cable. Since the positioner is running its own closed-loop PID algorithm (Fig. 3), the host central processor need only send position commands and is therefore free to service the user interface, main application software and command many remote positioners.

The limit switch inputs provide a safety net to keep a system from destroying itself in the event that the feedback device is lost. The optional current sense input can be used to determine if the load has jammed and prevent overheating of the actuator and drive electronics.

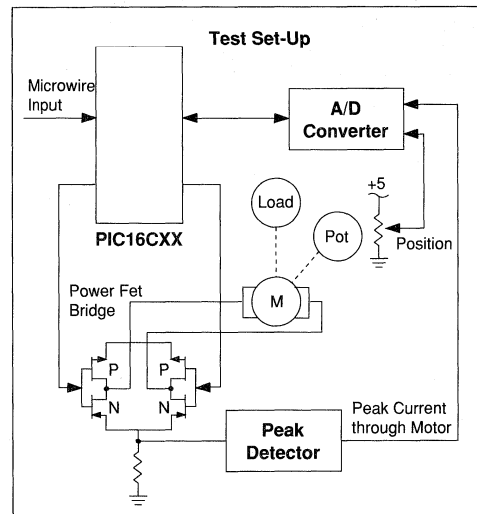
The commanded positions are presented to the PIC16C5X via a microwire type protocol at bit-rates of up to 50kb/s for the 4 MHz part. As currently implemented in this application note, the position request is the only communication. There are several variable locations available and they could be utilized to allow downloading of the loop gain parameters, reading positioner information, or for setting a current limit. The host that is sending the position request must set the chip select low, and wait for the PIC16C5X to raise the "busy" (DO) line high. At this point 8 data bits can be clocked into the PIC16C5X. The requested position is sent most signifi-

cant bit first and can be any 8 bit value. Values 1 through 255 represent valid positions with 0 being reserved for drive disable.

The PIC16C5X acquires data by way of a microwire A/D converter. This part was chosen for low cost yet it provides adequate performance. The second channel of the A/D is shown hooked up to a peak current detector. If the user desired, the PIC16C5X could monitor and protect the motor from overcurrent by monitoring this information.

The H-bridge power amplifier will deliver 10 or more amps at up to 24 volts when properly heat-sinked. It is wired for a modified 4-quadrant mode of operation. One leg of the bridge is used to control direction and the other leg pulses the low FET and the high FET alternately to generate the desired duty-cycle. In this way the system will operate well to produce a desired "speed" without the use of a separate speed control loop. This allows use of the PIC16C5X to control the PID algorithm for position directly while having reasonable speed control. The capacitance at the gates of the FETs combined with the impedance of the drive circuits provides for turn-off of the upper FET before the lower FET turns on... an important criteria.

FIGURE 1 - BLOCK DIAGRAM



MICROWIRE™ is a trademark of National Semiconductor Corporation.

Remote Positioner

The PID algorithm itself is where most of the meat of this application note is located so let's look at it more closely. The Algorithm is formed by summing the contribution of three basic components. The first calculation is the error for that is what the other terms are based on.

The error is the requested position minus the actual position. It is a signed number whose magnitude can be 255. In order not to lose resolution, the error is stored as an 8 bit magnitude with the sign stored separately in the FLAGS register under ER_SGN. This allows us to resolve a full signed 8 bit error with 8 bit math.

The proportional term is merely the algebraic difference of the requested position minus the actual position. It is scaled by a gain term (Kp) called the "proportional gain". The sign of this term is important for it tells the system which direction it must drive to correct the error. The proportional term is limited to +/- 100. Increasing the proportional gain term will improve the dynamic and static accuracy of the system. Increasing it too much will cause oscillations.

The next term that gets calculated is the Integral term. This term is traditionally formed by integrating the error over time. In this application it is done by integrating the Ki term over time. When the error is zero, no integration is performed. This is a more practical way to handle a potentially large number in 8 bit math. By increasing the Ki term the d.c. or static gain of the system is improved. Increasing the integral gain too much can lead to low frequency oscillations.

The differential term (Kd) is a stabilizing term that helps keep the integral and proportional terms from overdriving the system through the desired position and thus creating oscillations. As you use more proportional and integral gain you will need more differential gain as well. The differential gain is calculated by looking at the rate of change of the positional error with respect to time. It is actually formed as "delta error/delta time" with the delta time being a program cycle.

The three terms are summed algebraically and scaled to produce a percentage speed request between 0 and 100%. The sign of the sum is used to control the H-bridge direction. The loop calculations run approximately 20 times per second on a 4 MHz part. This yields sufficient gain-bandwidth for most positioning applications. If higher system performance is desired, the number of pulses can be reduced to 20 and a 16 MHz PIC16C5X can be used. Your Loop gains (Kp, Ki, Kd) will have to be recalculated, but the system sample rate will be increased to 400 Hz. This should be sufficient to control a system that has a response time of 20 milliseconds or more.

The key to using the PIC16C5X series parts for PID control and PWM generation is to separate the two into separate tasks. There is simply not the hardware support or the processing speed to accurately do both concurrently. It is fortunate therefore that it is not necessary to do both concurrently. The systems that are generally controlled can be stabilized with a much lower information update rate than the PWM frequency. This supports the approach of calculating the desired percentage, outputting the PWM for a period of time and then recalculating the new desired percentage. Utilizing this technique the inexpensive PIC16C5X can implement PID control, PWM generation and still have processing time left over for monitor or communication functions.

About the Author:

Steven Frank has been designing analog and digital control systems for 10 years. His background is in medical and consumer electronics. He has received numerous patents in control systems and instrumentation. At Vesta Technology Inc. Mr. Frank works with a number of engineers on custom embedded control systems designs. Vesta Technology Inc. is a provider of embedded control systems from an array of standard products and designs. Vesta offers custom design services and handles projects from concept to manufacturing.

FIGURE 2 - PROGRAM FLOW CHART

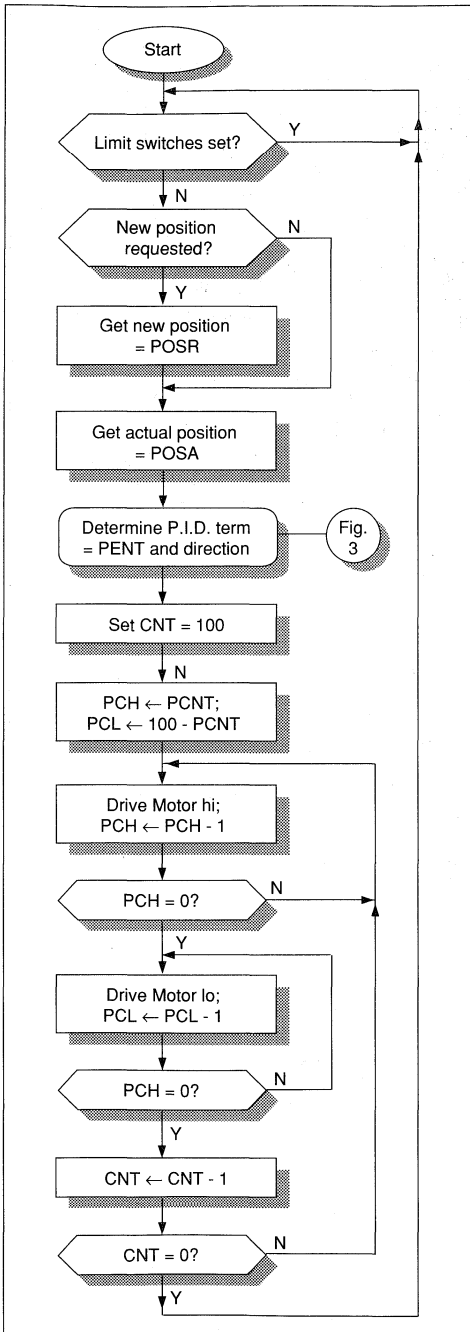
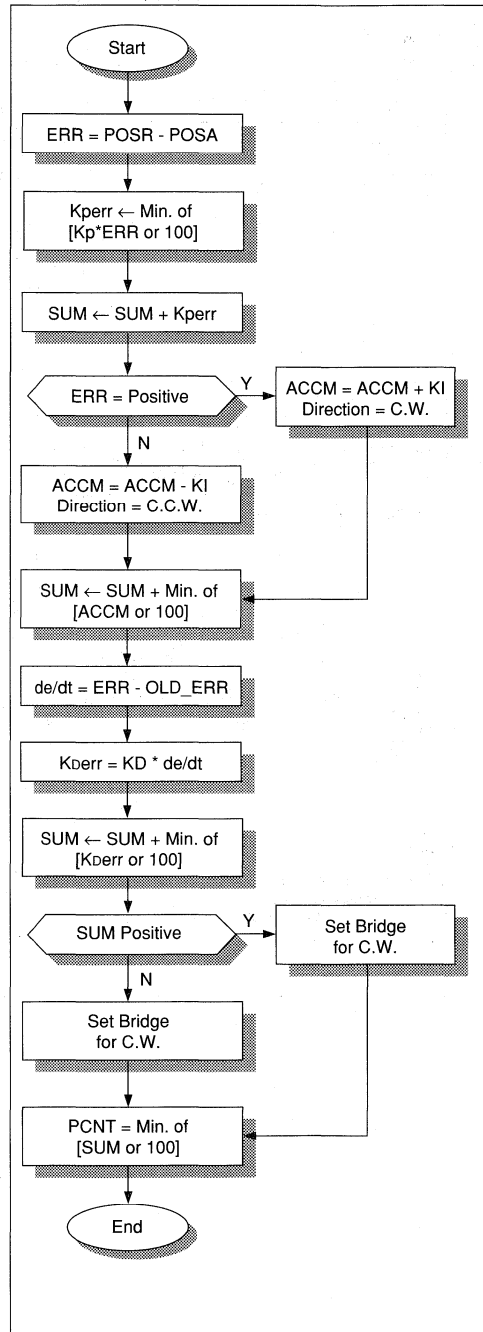


FIGURE 3 - P.I.D. ALGORITHM FLOW CHART



Remote Positioner

```

;
;      mw8pos.asm
;
;      LIST P=16C56
;*****
; REV. A      Original release 1/10/92 srf
;*****
;
; REGISTER EQUATES
;
0000      W      EQU      0
0000      PNTR   EQU      00H      ; CONTENTS OF POINTER
0019      FLAGS  EQU      19H      ; USE THIS VARIABLE LOCATION AS FLAGS
; 0 BIT IS SIGN OF ERROR 1 IS NEGATIVE
; 1 BIT IS SIGN OF ERROR ACCUMULATOR
; 2 BIT IS SIGN OF THE DE/DE TERM
; 3 BIT IS DIRECTION 0 IS CW
; 4 BIT IS SIGN OF THE OLD ERROR

0003      STATUS EQU      03H
0003      SWR    EQU      03H      ; STATUS WORD REGISTER
; 0 = CARRY
; 1 = DC
; 2 = Z, SET IF RESULT IS ZERO

0004      FSR    EQU      04H      ; FILE SELECT REGISTER
0005      PORTA  EQU      05H      ; I/O REG (A0-A3), (A4-A7 DEF=0)
0006      PORTB  EQU      06H      ; I/O REGISTER (B0-B7)
0007      HI     EQU      07H      ; NUMBER OF HIGH MICROSECONDS
0008      LO     EQU      08H      ; NUMBER OF LOW MICROSECONDS
0009      PCNT   EQU      09H      ; PERCENT DUTYCYCLE REQUEST
000A      HI_T   EQU      0AH      ; COUNTER FOR USECONDS LEFT/PULSE HI
000B      LO_T   EQU      0BH      ; COUNTER FOR USECONDS LEFT/PULSE LO
000C      ERROR  EQU      0CH      ; HOLDER FOR THE POSITIONAL ERROR
; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
; KEPT IN THE FLAG REGISTER (9BIT SIGNED)

000D      SUMLO  EQU      0DH      ; PROGRESSIVE SUM OF THE PID TERMS
000E      ACCUM  EQU      0EH      ; ERROR ACCUMULATOR
000F      ERR_O  EQU      0FH      ; ERROR HISTORY USED FOR de/dt
; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
; KEPT IN THE FLAG REGISTER (9BIT SIGNED)

0010      POSR   EQU      10H      ; POSITIONAL REQUEST
0011      POSA   EQU      11H      ; ACTUAL POSITION
0012      CYCLES EQU      12H      ; COUNTER FOR CYCLES OUT

0013      mulcnd equ      13H      ; 8 bit multiplicand
0013      ACCaLO EQU      13H      ; same location used for the add routine
0014      mulplr  equ      14H      ; 8 bit multiplier
0014      ACCbLO EQU      14H      ; same location used for the add routine
0015      H_byte  equ      15H      ; High byte of the 16 bit result
0015      ACCaHI  EQU      15H      ; same location used for the add routine
0016      L_byte  equ      16H      ; Low byte of the 16 bit result
0016      ACCbHI  EQU      16H      ; same location used for the add routine
0017      count  equ      17H      ; loop counter
0018      SUMHI  EQU      18H      ; HIGH BYTE OF THE LOOP SUM

; PORT ASSIGNMENTS AND CONSTANTS

0000      PWMCW  EQU      0      ; CLOCKWISE PWM OUTPUT BIT
0001      PWMCCW EQU      1      ; COUNTERCLOCKWISE PWM OUTPUT BIT
0000      CARRY  EQU      0      ; CARRY BIT IN THE STATUS REGISTER
0002      Z      EQU      2      ; THE ZERO BIT OF THE STATUS REGISTER
0001      Same   equ      1      ;

```



Remote Positioner

```

0000          ER_SGN EQU 0          ; SIGN BIT FOR THE ERROR IN FLAG REGISTER
0001          AC_SGN EQU 1          ; SIGN BIT FOR THE ERROR ACCUMULATOR
0002          DE_SGN EQU 2          ; SIGN BIT FOR DE/DT
0004          OER_SGN EQU 4         ; SIGN BIT FOR THE OLD ERROR
0030          KP EQU 30             ; PROPORTIONAL GAIN
0002          KI EQU 2              ; INTEGRAL GAIN
0020          KD EQU 20             ; DIFFERENTIAL GAIN
0003          DIR EQU 3             ; THE DIRECTION FLAG
0007          CSN EQU 7             ; CHIP SELECT NOT ON A/D
0006          BV EQU 6              ; DATA LINE FOR THE A/D
0005          CK EQU 5              ; CLOCK LINE FOR THE A/D
0002          MWDO EQU 2            ; MICROWIRE DATA OUT FROM POSITIONER
0001          MWDI EQU 1            ; MICROWIRE DATA IN TO POSITIONER
0000          MWCS EQU 0            ; MICROWIRE CHIP SELECT TO POSITIONER
0003          MWCK EQU 3            ; MICROWIRE CLOCK IN TO POSITIONER

```

```

;***** MACROS *****

```

```

;
CLKUP MACRO          ; clock up macro for the microwire
BSF PORTB,CK        ; data acquisition from the a/d
NOP
ENDM

```

```

CLKDN MACRO          ; clock down macro for the microwire
BCF PORTB,CK        ; data acquisition from the a/d
NOP
ENDM

```

```

GET_BIT MACRO          ; ** FOR RECEIVING A/D DATA **
BCF SWR,CARRY
BSF PORTB,CK          ; SET CLOCK BIT HIGH
BTFSC PORTB,BV        ; LOOK AT DATA COMMING IN
BSF SWR,CARRY         ; SET THE CARRY FOR A 1
RLF POSA              ; ROTATE THE W REG LEFT
BCF PORTB,CK          ; SET THE CLOCK LOW
NOP                   ; DELAY
ENDM

```

```

0000 0B88          GOTO CLRREG

```

```

;***** MATH ROUTINES *****

```

```

;
; **** 8 BIT MULTIPLY *****
; ***** Begin Multiplier Routine

```

```

0001 0075          mpy_s  clrf  H_byte
0002 0076          clrf  L_byte
0003 0C08          movlw  8
0004 0037          movwf  count
0005 0213          movf  mulcnd,w
0006 0403          bcf   STATUS,CARRY ; Clear the carry bit in the status Reg.
0007 0334          loop   rrf   mulplr
0008 0603          btfsc STATUS,CARRY
0009 01F5          addwf  H_byte,Same
000A 0335          rrf   H_byte,Same
000B 0336          rrf   L_byte,Same
000C 02F7          decfsz count
000D 0A07          goto  loop
000E 0800          retlw  0

```

```

; *****
; DOUBLE PRECISION ADD AND SUBTRACT ( ACCb-ACCa->ACCb )

```

```

000F 0917          D_sub  call  neg_A          ; At first negate ACCa, then add

```

```

; *****
; Double Precision Addition ( ACCb+ACCa->ACCb )

```

```

0010 0213          D_add  movf  ACCaLO,W
0011 01F4          addwf  ACCbLO          ; add lsb

```


Remote Positioner

```

0012 0603          btfsc STATUS,CARRY    ; add in carry
0013 02B6          incf ACCbHI
0014 0215          movf ACCaHI,W
0015 01F6          addwf ACCbHI          ; add msb
0016 0800          retlw 00
;
;
0017 0273          neg_A   comf ACCaLO          ; negate ACCa
0018 02B3          incf ACCaLO
0019 0643          btfsc STATUS,Z
001A 00F5          decf ACCaHI
001B 0275          comf ACCaHI
001C 0800          retlw 00

; *****
; divide by 16 and limit to 100 Decimal

SHIFT MACRO
BCF   SWR,CARRY
RRF   L_byte
BCF   SWR,CARRY
RRF   H_byte
BTFSC SWR,CARRY
BSF   L_byte,7
ENDM

DIV_LMT
SHIFT
001D 0403          BCF   SWR,CARRY
001E 0336          RRF   L_byte
001F 0403          BCF   SWR,CARRY
0020 0335          RRF   H_byte
0021 0603          BTFSC SWR,CARRY
0022 05F6          BSF   L_byte,7

SHIFT
0023 0403          BCF   SWR,CARRY
0024 0336          RRF   L_byte
0025 0403          BCF   SWR,CARRY
0026 0335          RRF   H_byte
0027 0603          BTFSC SWR,CARRY
0028 05F6          BSF   L_byte,7

SHIFT
0029 0403          BCF   SWR,CARRY
002A 0336          RRF   L_byte
002B 0403          BCF   SWR,CARRY
002C 0335          RRF   H_byte
002D 0603          BTFSC SWR,CARRY
002E 05F6          BSF   L_byte,7

SHIFT
002F 0403          BCF   SWR,CARRY
0030 0336          RRF   L_byte
0031 0403          BCF   SWR,CARRY
0032 0335          RRF   H_byte
0033 0603          BTFSC SWR,CARRY
0034 05F6          BSF   L_byte,7

LMT100
0035 0C01          MOVLW 1H          ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0036 0095          SUBWF H_byte,0      ; IF THERE IS ANYTHING THERE, IF NOT,
0037 0703          BTFSS SWR,CARRY    ; THEN LEAVE THE LOW BYTE ALONE
0038 0A3C          GOTO L8_E          ; OTHERWISE GIVE THE LOW BYTE A FULL
0039 0C64          MOVLW 64H          ; COUNT AND IT WILL HAVE BEEN LIMITED
003A 0036          MOVWF L_byte      ; TO 100
003B 0A42          GOTO LMT_EXIT

L8_E
003C 0C64          MOVLW 64H          ; LIMIT THE MAGNITUDE OF THE VALUE TO

```

Remote Positioner

```
003D 0096          SUBWF  L_byte,0          ; 100 DECIMAL
003E 0703          BTFSS  SWR,CARRY
003F 0A42          GOTO   LMT_EXIT
0040 0C64          MOVLW  64H
0041 0036          MOVWF  L_byte
                                LMT_EXIT
0042 0800          RETLW  00
;
;THE ROUTINE CALCTIMES DOES THE FOLLOWING: PCNT = DUTY CYCLE IN %
; 100 - PCNT -> LO AND PCNT -> HI. ZERO VALUES IN EITHER LO OR HI
;ARE FORCED TO 1.
CALCTIMES
0043 0209          MOVF   PCNT,W          ; PUT REQUESTED % INTO W REGISTER
0044 0027          MOVWF  HI          ; COPY ON MICROSECONDS IN TO HI TIME
0045 0C64          MOVLW  64H
0046 0028          MOVWF  LO
0047 0209          MOVF   PCNT,0
0048 00A8          SUBWF  LO,1          ; LEAVE 100-HI TIME IN LO TIME
0049 0207          MOVF   HI,0          ; INSPECT THE HIGH TIME
004A 0643          BTFSC  SWR,2          ; IF ITS IS ZERO
004B 02A7          INCF   HI,1          ; INCREMENT IT
004C 0208          MOVF   LO,0          ; INSPECT THE LO TIME
004D 0643          BTFSC  SWR,2          ; IF ITS ZERO
004E 02A8          INCF   LO,1          ; INCREMENT IT
004F 0800          RETLW  00

;*****
BEGIN
0050 0000          NOP          ; STUBBED BEGINNING

;****CHECKING THE LIMIT SWITCHES AND CHECKING FOR MW*****
; This will check the switch inputs for closure and will terminate
; pulsing is one is closed. It doesn't distinguish between the switches
; so they are not dedicated to cw end and ccw end.

SW_TRAP
0051 0004          CLRWDT
0052 0746          BTFSS  PORTB,2          ; THIS WILL TEST ALL THREE OF THE
0053 0A51          GOTO   SW_TRAP          ; SWITCH INPUTS. IF ANY ONE IS
0054 0766          BTFSS  PORTB,3          ; SET THEN EXECUTION OF THE CODE
0055 0A51          GOTO   SW_TRAP          ; WILL BE LIMITED TO LOOKING FOR
0056 0786          BTFSS  PORTB,4          ; IT TO BE CLEARED
0057 0A51          GOTO   SW_TRAP

;****RECEIVING THE POSITIONAL REQUEST*****
; The host system that wishes to send positional requests to the positioner
; servo makes its desire known by setting the chip select to the positioner
; low. It then monitors the busy (Data Out) line from the positioner. When
; the positioner sets the busy line high, the host may begin sending its 8
; request. The data bits should be valid on the rising edge of the clock.
; After 8 bits have been received by the positioner it will begin operation
; to send the system to the received position. It can be interrupted at any
; point during the positioning process by the host sending a new command.
; opportunity to update the command is issued every 100 pwm pulses (every 50
; milliseconds).
; If the host sends a zero positional command the positioner will stop the
; system and remain inactive.
; If the host does not successfully complete a microwire transmission of 8
; data bits the watchdog timer will trip and reset the system to an inactive
; "stopped" state.

REC_MW
0058 0C0B          MOVLW  0BH          ; RESET THE PORT FOR THREE INPUTS
```

Remote Positioner

```

0059 0005          TRIS   PORTA          ; AND ONE OUTPUT
005A 0445          BCF    PORTA,MWDO     ; SET THE DATA OUT LOW FOR BUSY
005B 0C20          MOVLW  20H
005C 0037          MOVWF  count

WATCH_CS
005D 0705          BTFSS  PORTA,MWCS     ; CHECK FOR INCOMMING REQUESTS
005E 0A62          GOTO   REC_CMD       ; RECEIVE A NEW POSITION REQUEST
005F 02F7          DECFSZ count,1
0060 0A5D          GOTO   WATCH_CS
0061 0A71          GOTO   REC_EXIT      ; NO REQUEST WAS MADE IN THE TIME ALLOTED

REC_CMD
0062 0545          BSF    PORTA,MWDO     ; SET THE DATA OUT HIGH FOR "OK TO SEND"
0063 0C08          MOVLW  8H            ; SET TO RECEIVE 8 BITS
0064 0037          MOVWF  count

WAIT_UP
0065 0765          BTFSS  PORTA,MWCK     ; WAIT FOR A RISING EDGE
0066 0A65          GOTO   WAIT_UP
0067 0403          BCF    SWR,CARRY      ; RESET THE CARRY TO A DEFAULT ZERO
0068 0625          BTFSC  PORTA,MWDI     ; READ THE DATA IN
0069 0503          BSF    SWR,CARRY      ; SET THE CARRY FOR A ONE
006A 0370          RLF    POSR,1        ; ROTATE THE BIT INTO THE POSITION REQ.
006B 02F7          DECFSZ count,1      ; DECREMENT THE BIT COUNTER
006C 0A6E          GOTO   WAIT_DN       ; WAIT FOR THE FALLING EDGE
006D 0A71          GOTO   REC_EXIT      ; LAST BIT RECEIVED

WAIT_DN
006E 0665          BTFSC  PORTA,MWCK     ; CHECK THE INCOMMING CLOCK
006F 0A6E          GOTO   WAIT_DN       ; IF IT IS STILL HIGH WAIT FOR IT TO GO LOW
0070 0A65          GOTO   WAIT_UP       ; IF IT GOES LOW GO BACK TO RECEIVE NEXT BIT

REC_EXIT
0071 0445          BCF    PORTA,MWDO     ; SET THE BUSY FLAG

;***** CHECK FOR THE DISABLE REQUEST *****
; Position 0 is considered a request to not drive the system. In this way
; the positioner will come up from a reset in a safe state and will not
; try to move the system to some arbitrary location.

MOVE?
0072 0210          MOVF   POSR,W         ; CHECK THE REQUESTED POSTION
0073 0643          BTFSC  SWR,Z         ; IF IT IS ZERO THEN WAIT FOR A NON-ZERO
0074 0A50          GOTO   BEGIN        ; REQUEST BY BRANCHING BACK TO THE BEGINNING

;****READING THE A/D VALUES*****
;
; Read the positional a/d channel (1) and store the value in the actual
; position variable (POSA).
; This is written in line to minimize the use of variables

READ_POS
0075 0071          CLRF   POSA         ; CLEAN THE POSITION ACTUAL HOLDER
0076 04B6          BCF    PORTB,CSN     ; SET THE CHIP SELECT LOW TO A/D
0077 0C1C          MOVLW  1CH          ; SET THE DATA LINE TO OUTPUT
0078 0006          TRIS  PORTB        ; FOR SENDING SET-UP BITS
0079 05C6          BSF   PORTB,BV      ; SET FOR "START" BIT
007A 0000          NOP

CLKUP
007B 05A6          BSF   PORTB,CK       ; CLOCK IN THE START BIT
007C 0000          NOP

CLKDN
007D 04A6          BCF   PORTB,CK       ; "
007E 0000          NOP

CLKUP
007F 05A6          BSF   PORTB,CK       ; CLOCK IN SINGLE-ENDED
0080 0000          NOP

CLKDN
; "

```

Remote Positioner

```

0081 04A6          BCF    PORTB,CK    ; data acquisition from the a/d
0082 0000          NOP

          CLKUP          ; CLOCK IN CHANNEL 1
0083 05A6          BSF    PORTB,CK    ; data acquisition from the a/d
0084 0000          NOP

          CLKDN          ; TO THE MUX
0085 04A6          BCF    PORTB,CK    ; data acquisition from the a/d
0086 0000          NOP

0087 0C5C          MOVLW  5CH          ; SET THE DATA LINE TO INPUT
0088 0006          TRIS   PORTB          ; TO RECEIVE DATA BITS FROM A/D
          CLKUP          ; CLOCK UP TO LET MUX SETTLE
0089 05A6          BSF    PORTB,CK    ; data acquisition from the a/d
008A 0000          NOP

          CLKDN          ; CLOCK DN TO LET MUX SETTLE
008B 04A6          BCF    PORTB,CK    ; data acquisition from the a/d
008C 0000          NOP

          GET_BIT        ; GET BIT 7
008D 0403          BCF    SWR,CARRY
008E 05A6          BSF    PORTB,CK    ; SET CLOCK BIT HIGH
008F 06C6          BTFSC  PORTB,BV    ; LOOK AT DATA COMMING IN
0090 0503          BSF    SWR,CARRY    ; SET THE CARRY FOR A 1
0091 0371          RLF    POSA          ; ROTATE THE W REG LEFT
0092 04A6          BCF    PORTB,CK    ; SET THE CLOCK LOW
0093 0000          NOP          ; DELAY

          GET_BIT        ; BIT 6
0094 0403          BCF    SWR,CARRY
0095 05A6          BSF    PORTB,CK    ; SET CLOCK BIT HIGH
0096 06C6          BTFSC  PORTB,BV    ; LOOK AT DATA COMMING IN
0097 0503          BSF    SWR,CARRY    ; SET THE CARRY FOR A 1
0098 0371          RLF    POSA          ; ROTATE THE W REG LEFT
0099 04A6          BCF    PORTB,CK    ; SET THE CLOCK LOW
009A 0000          NOP          ; DELAY

          GET_BIT        ; BIT 5
009B 0403          BCF    SWR,CARRY
009C 05A6          BSF    PORTB,CK    ; SET CLOCK BIT HIGH
009D 06C6          BTFSC  PORTB,BV    ; LOOK AT DATA COMMING IN
009E 0503          BSF    SWR,CARRY    ; SET THE CARRY FOR A 1
009F 0371          RLF    POSA          ; ROTATE THE W REG LEFT
00A0 04A6          BCF    PORTB,CK    ; SET THE CLOCK LOW
00A1 0000          NOP          ; DELAY

          GET_BIT        ; BIT 4
00A2 0403          BCF    SWR,CARRY
00A3 05A6          BSF    PORTB,CK    ; SET CLOCK BIT HIGH
00A4 06C6          BTFSC  PORTB,BV    ; LOOK AT DATA COMMING IN
00A5 0503          BSF    SWR,CARRY    ; SET THE CARRY FOR A 1
00A6 0371          RLF    POSA          ; ROTATE THE W REG LEFT
00A7 04A6          BCF    PORTB,CK    ; SET THE CLOCK LOW
00A8 0000          NOP          ; DELAY

          GET_BIT        ; BIT 3
00A9 0403          BCF    SWR,CARRY
00AA 05A6          BSF    PORTB,CK    ; SET CLOCK BIT HIGH
00AB 06C6          BTFSC  PORTB,BV    ; LOOK AT DATA COMMING IN
00AC 0503          BSF    SWR,CARRY    ; SET THE CARRY FOR A 1
00AD 0371          RLF    POSA          ; ROTATE THE W REG LEFT
00AE 04A6          BCF    PORTB,CK    ; SET THE CLOCK LOW
00AF 0000          NOP          ; DELAY

          GET_BIT        ; BIT 2
00B0 0403          BCF    SWR,CARRY
00B1 05A6          BSF    PORTB,CK    ; SET CLOCK BIT HIGH
00B2 06C6          BTFSC  PORTB,BV    ; LOOK AT DATA COMMING IN

```

Remote Positioner

```

00B3 0503          BSF     SWR,CARRY      ; SET THE CARRY FOR A 1
00B4 0371          RLF     POSA           ; ROTATE THE W REG LEFT
00B5 04A6          BCF     PORTB,CK       ; SET THE CLOCK LOW
00B6 0000          NOP                     ; DELAY

                GET_BIT          ; BIT 1
00B7 0403          BCF     SWR,CARRY      ; SET CLOCK BIT HIGH
00B8 05A6          BSF     PORTB,CK       ; LOOK AT DATA COMMING IN
00B9 06C6          BTFSC   PORTB,BV       ; SET THE CARRY FOR A 1
00BA 0503          BSF     SWR,CARRY      ; ROTATE THE W REG LEFT
00BB 0371          RLF     POSA           ; SET THE CLOCK LOW
00BC 04A6          BCF     PORTB,CK       ; DELAY
00BD 0000          NOP                     ; DELAY

                GET_BIT          ; BIT 0
00BE 0403          BCF     SWR,CARRY      ; SET CLOCK BIT HIGH
00BF 05A6          BSF     PORTB,CK       ; LOOK AT DATA COMMING IN
00C0 06C6          BTFSC   PORTB,BV       ; SET THE CARRY FOR A 1
00C1 0503          BSF     SWR,CARRY      ; ROTATE THE W REG LEFT
00C2 0371          RLF     POSA           ; SET THE CLOCK LOW
00C3 04A6          BCF     PORTB,CK       ; DELAY
00C4 0000          NOP                     ; DESELECT THE CHIP

00C5 05E6          BSF     PORTB,CSN     ; DESELECT THE CHIP

;***** CALCULATING THE PID TERMS *****

;****CALCULATE THE ERROR*****
; The error is very simply the signed difference between where the
; system is and where it is supposed to be at a particular instant
; in time. It is formed by subtracting the actual position from the
; requested position (Position requested - Position actual). This
; difference is then used to determine the proportional,integral and
; differential term contributions to the output.

C_ERR
00C6 0211          MOVF    POSA,0           ; LOAD THE ACTUAL POSITION INTO W
00C7 0090          SUBWF   POSR,0           ; SUBTRACT IT FROM THE REQUESTED POSITION
00C8 0603          BTFSC   SWR,CARRY      ; CHECK THE CARRY BIT TO DETERMINE THE SIGN
00C9 0ACB          GOTO    PLS_ER          ; ITS POSITIVE (POSR>POSA)
00CA 0ACE          GOTO    MNS_ER          ; ITS NEGATIVE (POSA>POSR)

PLS_ER
00CB 002C          MOVWF   ERROR           ; SAVE THE DIFFERENCE IN "ERROR"
00CC 0419          BCF     FLAGS,ER_SGN     ; SET THE SIGN FLAG TO INDICATE POSITIVE
00CD 0AD2          GOTO    CE_EXIT

MNS_ER
00CE 0210          MOVF    POSR,0           ; RE-DO THE SUBTRACTION
00CF 0091          SUBWF   POSA,0           ; ACTUAL - REQUESTED
00D0 002C          MOVWF   ERROR           ; STORE THE DIFFERENCE IN "ERROR"
00D1 0519          BSF     FLAGS,ER_SGN     ; SET THE SIGN FLAG FOR NEGATIVE

CE_EXIT
00D2 006D          CLRf    SUMLO           ; CLEAN OLD VALUES OUT TO PREPARE
00D3 0078          CLRf    SUMHI           ; FOR THIS CYCLES SUMMATION

;****CALCULATE THE PROPORTIONAL TERM*****
; The proportional term is the error times the proportional gain term.
; This term simply gives you more output drive the farther away you are
; from where you want to be (error)*Kp.
; The proportional gain term is a signed term between -100 and 100 The
; more proportional gain you have the lower your system following error
; will be. The higher your proportional gain, the more integral and
; differential term gains you will have to add to make the system stable.
; The sum is being carried as a 16 bit signed value.

C_PROP

```

Remote Positioner

```

00D4 020C      MOVF     ERROR,0           ; LOAD THE ERROR TERM INTO W
00D5 0033      MOVWF   mulcnd           ; MULTIPLY IT BY THE PROPORTIONAL GAIN
00D6 0C30      MOVLW  KP                ; KP AND THEN SCALE IT DOWN BY DIVIVING
00D7 0034      MOVWF   mulplr          ; IT DOWN BY 16. IF IT IS STILL OVER
00D8 0901      CALL    mpy_S           ; 255 THEN LIMIT IT TO 255
00D9 091D      CALL    DIV_LMT

                RESTORE_SGN
00DA 0719      BTFSF   FLAGS,ER_SGN    ; IF THE ERROR SIGN IS NEGATIVE THEN
00DB 0A0E      GOTO   ADDPROP         ; PUT THE SIGN INTO THE LOW BYTE
00DC 0276      COMF   L_byte,1
00DD 02B6      INCF   L_byte,1

                ADDPROP
00DE 0216      MOVF   L_byte,W         ; SAVE THE PROPORTIONAL PART
00DF 01ED      ADDWF  SUMLO,1         ; IN THE SUM
00E0 0603      BTFSF  SWR,CARRY       ; IF THE ADDITION CARRIED OUT THEN
00E1 02B8      INCF  SUMHI,1         ; INCREMENT THE HIGH BYTE
00E2 0C00      MOVLW  0                ; THEN
00E3 06ED      BTFSF  SUMLO,7         ; SIGN EXTEND TO THE UPPER
00E4 0CFF      MOVLW  0FF             ; BYTE
00E5 01F8      ADDWF  SUMHI,1

                ;****CALCULATE THE INTEGRAL TERM*****
                ; The integral term is an accumulation of the error thus far. Its purpose
                ; is to allow even a small error to effect a large change. It does this
                ; by adding a small number into an accumulator each cycle through the pro
                ; Thusly even a small error that exist for a while will build up to a large
                ; enough number to effect an output sufficient to move the system. The
                ; that this integral accumulator has is modulated by the integral gain term
                ; The integral of the error over time is multiplied be KI and the result is
                ; contribution to the final summation for determining the output value. This
                ; term helps to insure the long-term accuracy of the system is good. A
                ; amount is necessary for this purpose but too much will cause oscillations.
                ; The integral is bounded in magnitude for two purposes. The first is so
                ; it never rolls over and changes sign. The second is that it may saturate
                ; long moves forcing an excessively large overshoot to "de-integrate" the
                ; accumulated during the first of the move

                C_INT
00E6 020C      MOVF   ERROR,W           ; MOVE THE ERROR INTO THE W REG
00E7 0643      BTFSF  SWR,Z           ; AND CHECK TO SEE IF IT IS ZERO
00E8 0AFF      GOTO   ADDINT         ; IF SO THEN DONT CHANGE THE ACCUMULATOR
00E9 0619      BTFSF  FLAGS,ER_SGN    ; TEST THE FLAGS TO FIND THE POLARITY
00EA 0AEE      GOTO   MNS_1         ; OF THE ERROR .. 0 POSITIVE 1 NEGATIVE

                PLS_1
00EB 0C02      MOVLW  KI                ; IF POSATIVE ADD ONE TO
00EC 01EE      ADDWF  ACCUM,1         ; THE ERROR ACCUMULATOR
00ED 0AF0      GOTO   LMTACM        ; THEN LIMIT IT TO +/-100

                MNS_1
00EE 0C02      MOVLW  KI                ; IF NEGATIVE THEN SUBTRACT ONE
00EF 00AE      SUBWF  ACCUM,1         ; FROM THE ERROR ACCUMULATOR

                LMTACM
00F0 06EE      BTFSF  ACCUM,7         ; CHECK THE SIGN BIT OF THE ERROR ACCUMULA-
TOR
00F1 0AF9      GOTO   M_LMT         ; AND DO A POSATIVE OR NEGATIVE LIMIT

                P_LMT
00F2 0C9C      MOVLW  9CH             ; FOR THE POSATIVE LIMIT ADD 156 TO THE
00F3 01CE      ADDWF  ACCUM,0         ; NUMBER AND SEE IF YOU GENERATE A CARRY
00F4 0703      BTFSF  SWR,CARRY       ; BY CHECKING THE CARRY FLAG
00F5 0AFF      GOTO   ADDINT         ; IF NOT THEN ITS O.K.
00F6 0C64      MOVLW  64H             ; IF SO THEN FORCE THE ACCUMULATOR TO
00F7 002E      MOVWF  ACCUM          ; 100 DECIMAL
00F8 0AFF      GOTO   ADDINT

                M_LMT
00F9 0C9C      MOVLW  9CH             ; FOR THE NEGATIVE LIMIT SUBTRACT 156 FROM
00FA 008E      SUBWF  ACCUM,0         ; THE NUMBER AND SEE IF YOU GENERATE A

```


Remote Positioner

```
0126 0214          MOVF   ACCbLO,W      ; AND SAVE THE VALUE
0127 002F          MOVWF  ERR_O

; Then multiply by Kd

MULT_KD
0128 020F          MOVF   ERR_O,W
0129 0033          MOVWF  mulcnd      ; MOVE THE DE/DT TERM INTO THE MULCND REG.
012A 0C20          MOVLW  KD          ; MOVE THE DIFFERENTIAL GAIN TERM INTO
012B 0034          MOVWF  mulplr      ; MULPLR TO MULTIPLY THE DE/DT
012C 0901          CALL   mpy_s       ; DO THE MULTIPLICATION
012D 091D          CALL   DIV_LMT     ; SCALE AND LIMIT TO 100

RE_SGN
012E 0759          BTFSS  FLAGS,DE_SGN  ; IF THE DE SIGN IS NEGATIVE THEN
012F 0B32          GOTO   SAVE_DIFF   ; PUT THE SIGN INTO THE LOW BYTE
0130 0276          COMF   L_byte,1
0131 02B6          INCF   L_byte,1

SAVE_DIFF
0132 0216          MOVF   L_byte,W
0133 0643          BTFSC  SWR,Z
0134 0B45          GOTO   ROLL_ER
0135 002F          MOVWF  ERR_O

; ADD THE DIFF TERM INTO THE SUM *****
ADDDIF
0136 0C00          MOVLW  00
0137 0659          BTFSC  FLAGS,DE_SGN  ; PUT THE KD*(DE/DT) TERM INTO THE
0138 0CEF          MOVLW  0FF        ; REGISTERS TO ADD. AND
0139 0036          MOVWF  ACCbHI      ; SIGN EXTEND THE UPPER BYTE
013A 020F          MOVF   ERR_O,W
013B 0034          MOVWF  ACCbLO
013C 020D          MOVF   SUMLO,W      ; LOAD THE CURRENT SUM INTO THE
013D 0033          MOVWF  ACCaLO     ; REGISTERS TO ADD
013E 0218          MOVF   SUMHI,W
013F 0035          MOVWF  ACCaHI
0140 0910          CALL   D_add        ; ADD IN THE DIFFERENTIAL TERM
0141 0214          MOVF   ACCbLO,W      ; SAVE THE RESULTS BACK
0142 002D          MOVWF  SUMLO      ; INTO SUMLO AND HI
0143 0216          MOVF   ACCbHI,W
0144 0038          MOVWF  SUMHI

ROLL_ER
0145 020C          MOVF   ERROR,W      ; TAKE THE CURRENT ERROR
0146 002F          MOVWF  ERR_O      ; AND PUT IT IN THE ERROR HISTORY
0147 0499          BCF   FLAGS,OER_SGN  ; SAVE THE CURRENT ERROR SIGN
0148 0619          BTFSC  FLAGS,ER_SGN  ; IN THE OLD ERROR SIGN FOR
0149 0599          BSF   FLAGS,OER_SGN  ; NEXT TIME THROUGH

;****SET UP THE DIRECTION FOR THE BRIDGE*****
;
; After the sum of all the components has been made, the sign of the
; sum will determine which way the bridge should be powered.
; If the sum is negative the bridge needs to be set to drive ccw; if the
; sum is positive then the bridge needs to be set to drive cw. This
; is purely a convention and depends upon the polarity the motor and feedback
; element are hooked up in.

SET_DIR
014A 0479          BCF   FLAGS,DIR      ; SET FOR DEFAULT CLOCKWISE
014B 06F8          BTFSC  SUMHI,7      ; LOOK AT THE SIGN BIT, IF IT IS SET
014C 0579          BSF   FLAGS,DIR      ; THEN SET FOR CCW BRIDGE DRIVE

;**** SCALE THE NUMBER TO BETWEEN 0 AND 100% *****

; After the direction is set the request for duty cycle is limited to between
; 0 and 100 percent inclusive. This value is passed to the dutycycle setting
```


Remote Positioner

; routine by loading it in the variable "PCNT".

```
L_SUMM
014D 07F8      BTFSS  SUMHI,7      ; CHECK TO SEE IF IT IS NEGATIVE
014E 0B52      GOTO   POS_LM
014F 0278      COMF   SUMHI,1
0150 026D      COMF   SUMLO,1
0151 02AD      INCF   SUMLO,1

POS_LM
0152 0C01      MOVLW  1H          ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0153 0098      SUBWF  SUMHI,0     ; IF THERE IS ANYTHING THERE, IF NOT,
0154 0703      BTFSS  SWR,CARRY   ; THEN LEAVE THE LOW BYTE ALONE
0155 0B59      GOTO   LB_L       ; OTHERWISE GIVE THE LOW BYTE A FULL
0156 0C64      MOVLW  64H        ; COUNT AND IT WILL HAVE BEEN LIMITED
0157 002D      MOVWF  SUMLO      ; TO 100
0158 0B5F      GOTO   LP_EXIT    ; GOTO LIMIT PERCENT EXIT

LB_L
0159 0C64      MOVLW  64H        ; LIMIT THE MAGNITUDE OF THE VALUE TO
015A 008D      SUBWF  SUMLO,0     ; 100 DECIMAL
015B 0703      BTFSS  SWR,CARRY
015C 0B5F      GOTO   LP_EXIT
015D 0C64      MOVLW  64H
015E 002D      MOVWF  SUMLO

LP_EXIT
015F 020D      MOVF   SUMLO,W     ; STORE THE LIMITED VALUE IN
0160 0029      MOVWF  PCNT       ; THE PERCENT DUTYCYCLE REQUEST

;*****
; PWM GENERATING ROUTINE
;
; The important thing here is not to have to do too many decisions or
; calculations while you are generating the 100 or so pulses. These will
; take time and limit the minimum or maximum duty cycle.

WHICH_DIR
0161 0679      BTFSC  FLAGS,DIR   ; CHECK THE DIRECTION FLAG
0162 0B76      GOTO   GOCCW     ; DO CCW PULSES FOR 1
0163 0B64      GOTO   GOCW      ; DO CW PULSES FOR 0

GOCW
0164 0426      BCF    PORTB,PWMCCW ; SET THE BRIDGE FOR CW MOVE
0165 0C64      MOVLW  64H        ;
0166 0032      MOVWF  CYCLES     ; SET UP CYCLES COUNTER FOR 100 PULSES
0167 0943      CALL  CALCTIMES  ; CALCULATE THE HI AND LO TIMES

RLDCW
0168 0207      MOVF   HI,0        ; RE LOAD THE HI TIMER
0169 002A      MOVWF  HI_T       ; WITH THE CALCULATED TIME
016A 0208      MOVF   LO,0       ; RE LOAD THE LO TIMER
016B 002B      MOVWF  LO_T       ; WITH THE CALCULATED TIME
016C 0004      CLRWDT            ; TAG THE WATCHDOG TIMER

CWHI
016D 0506      BSF    PORTB,PWMCW ; SET THE CLOCKWISE PWMBIT HIGH
016E 02EA      DECFSZ HI_T,1   ; DECREMENT THE HI USEC. COUNTER
016F 0B6D      GOTO   CWHI ; DO ANOTHER LOOP

CWLO
0170 0406      BCF    PORTB,PWMCW ; SET THE CLOCKWISE PWM BIT LOW
0171 02EB      DECFSZ LO_T,1   ; DECREMENT THE LO USEC. COUNTER
0172 0B70      GOTO   CWLO ; DO ANOTHER LOOP
0173 02F2      DECFSZ CYCLES,1 ; DECREMENT THE NUMBER OF CYCLES LEFT
0174 0B68      GOTO   RLDCW ; DO ANOTHER PULSE
0175 0A50      GOTO   BEGIN  ; DO ANOTHER MAIN SYSTEM CYCLE
```

Remote Positioner

```

GOCCW
0176 0406      BCF     PORTB,PWMCW      ; SET THE BRIDGE FOR CCW MOVE
0177 0C64      MOVLW   64H              ;
0178 0032      MOVWF   CYCLES          ; SET UP CYCLE COUNTER FOR 100 PULSES
0179 0943      CALL    CALCTIMES      ; CALCULATE THE HI AND LO TIMES

RLDCCW
017A 0207      MOVF    HI,0           ; RE LOAD THE HI TIMER
017B 002A      MOVWF   HI_T          ; WITH THE CALCULATED TIME
017C 0208      MOVF    LO,0           ; RE LOAD THE LO TIMER
017D 002B      MOVWF   LO_T          ; WITH THE CALCULATED TIME
017E 0004      CLRWDT                ; TAG THE WATCHDOG

CCWHI
017F 0526      BSF     PORTB,PWMCW      ; SET THE COUNTERCLOCKWISE PWM BIT HIGH
0180 02EA      DECFSZ  HI_T,1         ; DECREMENT THE HI USEC. COUNTER
0181 0B7F      GOTO    CCWHI          ; DO ANOTHER LOOP

CCWLO
0182 0426      BCF     PORTB,PWMCW      ; SET THE COUNTERCLOCKWISE PWM BIT LOW
0183 02EB      DECFSZ  LO_T,1         ; DECREMENT THE LO USEC. COUNTER
0184 0B82      GOTO    CCWLO          ; DO ANOTHER LOOP
0185 02F2      DECFSZ  CYCLES,1       ; DECREMENT THE NUMBER OF CYCLES LEFT
0186 0B7A      GOTO    RLDCCW        ; DO ANOTHER PULSE
0187 0A50      GOTO    BEGIN          ; DO ANOTHER MAIN SYSTEM CYCLE

```

,***** START VECTOR *****

```

CLRREG                ;INITIALIZE REGISTERS

0188 0C0B      MOVLW   0BH           ; SET PORT A FOR 3 INPUTS AND
0189 0005      TRIS   PORTA          ; AN OUTPUT
018A 0C1C      MOVLW   1CH           ; SET PORT B FOR INPUTS AND OUTPUTS
018B 0006      TRIS   PORTB          ; THIS SETTING FOR SENDING TO A/D
018C 0040      CLRW                ; CLEAR THE W REGISTER
018D 0002      OPTION                ; STORE THE W REG IN THE OPTION REG
018E 0C08      MOVLW   08H           ; STARTING REGISTER TO ZERO
018F 0024      MOVWF   FSR           ;

GCLR
0190 0060      CLRF    00             ;
0191 03E4      INCFSZ  FSR           ; SKIP AFTER ALL REGISTERS
0192 0B90      GOTO    GCLR          ; HAVE BEEN INITIALIZED
0193 0A50      GOTO    BEGIN          ; START AT THE BEGINING OF THE PROGRAM

ORG 01FF
01FF 0B88      GOTO    CLRREG        ; START VECTOR

```

END

```

Errors : 0
Warnings : 0

```



Programming PIC16C5X Devices on Data I/O's Unisite™ Universal Programmer

INTRODUCTION

UNISITE, from Data I/O Corporation, is a universal programming system that supports a wide variety of programmable IC on the market, including Microchip's PIC16C5X series of EPROM-based single-chip micro-controllers.

The UNISITE is available in various versions and options for:

- single unit programming
- gang/set programming
- IC-handler interfacing
- support for devices in surface mountable packages

This application note describes three ways of programming PIC16C5X devices on this programmer for prototyping during development and production programming in medium and high volumes. Only PIC16C5X specific topics will be covered. For more information on using the UNISITE in general, please refer to Data I/O's literature.

REQUIRED EQUIPMENT OPTIONS

The basic version of the UNISITE with 28-pin support (SITE 48™) and software/firmware release V3.0 is the base requirement to program PIC16C5X devices (see Note 1). In addition, a "dumb" terminal with RS232 interface, or a personal computer with Data I/O's proprietary "PROMLINK" software, any standard communication software package (e.g., PROCOMM™) is required to control UNISITE and/or to download object files.

Note 1: Software release V2.8 supports the PIC16C5X series except for the code protect function. Do not use this function with V2.8 as the devices may not be functional in the application afterwards.

Note 2: The "SETSITE™" option does not support microcontrollers.

Note 3: The "CHIPSITE™" option is not supported by the software release V3.0 regarding PIC16C5X programming. To program PIC16C5Xs in surface mount packages (such as PLCC and SOIC) use adapter sockets that plug into a DIP socket.

THREE WAYS TO PROGRAM PIC16C5X DEVICES ON THE UNISITE

Duplication of a "Master" Device

This is the easiest way to program PIC16C5Xs on the UNISITE. Only a "dumb" terminal is required and an already programmed "Master" PIC16C5X. The "Master" can be generated by either using Microchip's PICPRO programmer or by using the method as described in the section titled "Prototype Programming" of this application note.

Step 1: Select the appropriate PIC16C5X device from UNISITE software menu.

Step 2: Load the "Master" PIC16C5X into the UNISITE's RAM using the corresponding menu function. All program memory locations of the PIC16C5X, including the configuration EPROM (watchdog timer disable fuse and oscillator selection) and the customer ID bits will be loaded and stored. Note that the "Master" PIC16C5X must not be code protected.

Step 3: Remove the "Master" PIC16C5X and insert a blank PIC into the programming socket.

When programming OTP devices, make sure that the oscillator type of the OTP part matches the oscillator selection of your "Master" PIC16C5X as the oscillator type of an OTP part cannot be changed by the programmer.

When programming windowed devices, the oscillator selection of the "Master" PIC16C5X will be duplicated as well.

Step 4: Activate the "Program" menu function of the UNISITE.

A. If no code protection is desired:

Hit the <CR> key.

The UNISITE will produce a 1:1 copy of the "Master", including customer ID code and configuration EPROM. After successful programming a checksum is being displayed in the message line. Note that this checksum is different from the one generated by Microchip's PICPRO due to different ways of computation.

B. If code protection is desired:

Select "security fuse data" to be "1" and enable "security fuse option" ("Y").

Then hit <CR>.

Programming PIC16C5X on Data I/O's Unisite

The "Master" unit will be copied 1:1 into the blank device, will be verified, and then the code protection fuse will be blown. After successful programming a checksum is being displayed in the message line.

Note that this checksum is generated before the code protection has been activated. Thus, it is the same as in Option A.

Steps 3 and 4 can be executed as often as required to produce any amount of "duplicates".

As the programming time for PIC16C54 or PIC16C55, respectively, is less than five seconds, this method is very effective and fast for any kind of production programming of PIC16C5X devices.

An alternative way to duplicate a PIC16C5X device, if no code protection is desired, is given with the "Quick Copy" command of the UNISITE.

PROTOTYPE PROGRAMMING

This method allows the generation of a prototype or "Master" PIC16C5X during the product development cycle. A personal computer is required to:

- A. generate a formatted PIC16C5X object file using MPALC and,
- B. download the contents of this file to the UNISITE.

Besides the normal object code, the configuration EPROM of the PIC16C5X device and, optionally, the customer ID bits have to be specified before programming a PIC16C5X device on the UNISITE.

Step 1: Write the application source code with the processor selection of your choice specified in the LIST assembler directive.

Example: LIST P=16C55

Step 2: Assemble your source code and debug with PICMASTER or MPSIM as required.

Step 3: Modify your debugged source file:

- A. Change the processor selection in the LIST directive to:

P=16C62

This enables MPALC to accept addresses above the normal program memory space.

- B. Define customer ID code by adding before the END statement:

```
ORG    N           ;see Table 1 for value of N
DATA   ID3,ID2,ID1,ID0 ;four hex digits representing
                        ;the ID code
DATA   B'CONFIG'   ;three bits - see Table 2
```

TABLE 1 - PIC16C5X ID CODE START ADDRESS

Device	"N"	Block Size
PIC16C54	200 (hex)	40A (hex)
PIC16C55	200 (hex)	40A (hex)
PIC16C56	400 (hex)	80A (hex)
PIC16C57	800 (hex)	100A (hex)

Note: The block size in UNISITE is the total number of bytes. For PIC16C54, for example, total number of words = 1FFh + 4 (ID locations) + 1 (configuration word) = 205h. The total number of bytes, is therefore, 40Ah.

TABLE 2 - PIC16C5X CONFIGURATION EPROM FUSES

Configuration			Function
Bit 2	Bit 1	Bit 0	
1	X	X	WDT Enabled (Default)
0	X	X	WDT Disabled
X	0	0	LP Oscillator
X	0	1	XT Oscillator
X	1	0	HS Oscillator
X	1	1	RC Oscillator

Example:

```
ORG    200H           ;PIC16C54 or PIC16C55
DATA   0,0A,3,0F     ;ID code = 0A3F (hex)
DATA   B'001'        ;WDT disabled, XT oscillator
```

Step 4: Re-assemble the modified source code with MPALC, specifying the "merged hex" output file format (INHX8M) in the command line.

Example: MPALC -F INHX8M <filename>

or

Example: LIST P=16C54, F=INHX8M

Step 5: Establish communication with UNISITE and select the desired PIC16C5X device from the UNISITE menu.

Step 6: Select the file transfer menu of the UNISITE software. Specify data translation format as "83". (Note that the file size and addresses relate to byte counts - not 12-bit words; one PIC16C5X location corresponds to two bytes.) You may want to "fill" the memory with "FFh" before downloading.

Programming PIC16C5X on Data I/O's Unisite

- Step 7: Download the object file into the UNISITE RAM using the ASCII file transfer option of the serial communication software or the corresponding PROMLINK function.
- Step 8: Select "program" menu. Handle the code protect option the same as in Step 4 of the section titled Duplication of a "Master" Device.

Alternative to Define Customer ID code and Configuration EPROM

If you are familiar with the UNISITE, you may skip Step 3 and edit the customer ID code and configuration EPROM directly in the UNISITE RAM in Step 6.

When doing so, you have to reduce the file size and address limits of the UNISITE default values (Step 6) by ten before downloading the object file (Step 7).

After downloading, edit the data of the customer ID location (see Table 1) one hex digit per location and the location immediately after ID locations per Table 2 (Step 7A). Note that you have to convert the binary CONFIG value to a hex value first. The most significant bits are "don't cares" in both cases.

Then continue with Step 8.

High Volume Programming with IC-Handler

For the fastest and most efficient programming of PIC16C5X devices in high volumes, an IC-handler should be employed. The UNISITE provides with its "HANDLERSITE" option, support for this application.

Data I/O and the manufacturer of the IC-handler should be contacted for further setup information.

This is the only recommended way for high volume programming the PIC16C5X devices in surface mountable packages (e.g., SOIC packages) at the customer's facilities.

The object data can be loaded into the UNISITE's RAM with any of the methods described above.

*Author: Sumit Mitra
Logic Products Division*

Programming PIC16C5X on Data I/O's Unisite

NOTES:



Programming PIC16C5X Devices on Logical Devices' ALLPRO™ Programmer

INTRODUCTION

The ALLPRO, from Logical Devices, is a universal programming system supporting a wide range of programmable devices, including Microchip's PIC16C5X series of EPROM based single-chip microcontrollers.

This application note describes two ways of programming PIC16C5X devices on this programmer for prototyping during development and production programming. Only PIC16C5X specific topics will be covered. For more information on using the ALLPRO in general, please refer to Logical Devices' literature.

REQUIRED EQUIPMENT OPTIONS

The basic version of the ALLPRO with 40-pin support and software/firmware release V1.49C is sufficient to program PIC16C5X devices. In addition, a personal computer with ALLPRO's proprietary software and interface card is required to control the ALLPRO and to download object files.

TWO WAYS TO PROGRAM PIC16C5X DEVICES ON THE ALLPRO

Duplication of a "Master" Device

This is the easiest way to program PICs on the ALLPRO. Only an already programmed "Master" PIC16C5X is required. The "Master" can be generated by either using Microchip's PICPRO programmer or by using the method as described in the section titled "Prototype Programming" of this application note.

Step 1: Run the ALLPRO program on the PC and select the "DEVICE SELECT" menu. Type "MICMIC" where the first MIC is for microcontrollers and the second MIC is for Microchip. Then select the appropriate device..

Step 2: Place the "Master" device into the ALLPRO socket and select "LOAD DEVICE" from the menu to load the "Master" PIC16C5X into the ALLPRO's RAM. All program memory locations of the PIC, including the configuration EPROM (watchdog timer disable fuse and oscillator selection) and the customer ID bits will be loaded and stored. Note that the "Master" PIC16C5X must not be code protected.

Step 3: Remove the "Master" PIC16C5X and insert a blank PIC16C5X into the programming socket.

When programming OTP devices, make sure that the oscillator type of the OTP part matches the oscillator selection of your "Master" PIC16C5X as the oscillator type of an OTP part cannot be changed by the programmer.

When programming windowed devices, the oscillator selection of the "Master" PIC16C5X will be duplicated as well.

Step 4: Select the "PROGRAM DEVICE" menu function of the ALLPRO and hit enter.

The ALLPRO will produce a 1:1 copy of the "Master", including customer ID code and configuration EPROM. After successful programming a checksum will be displayed in the message line. Note that this checksum is different from the one generated by Microchip's PICPRO due to different ways of computation.

If code protection is desired, select "SECURE DEVICE" on the menu, then hit enter. This will blow the code protection fuse.

Steps 3 and 4 can be executed as often as required to produce any amount of "duplicates". The "Master" device only needs to be downloaded the first time.

As the programming time for a PIC16C54 or PIC16C55 is less than ten seconds, this method is very effective and fast for any kind of production programming of PIC16C5X devices.

Prototype Programming

This method allows the generation of a prototype or "Master" PIC16C5X during the product development cycle. A personal computer is required to:

- A. generate a formatted PIC16C5X object file using MPALC and,
- B. download the contents of this file to the ALLPRO.

Besides the normal object code, the configuration EPROM of the PIC16C5X device and, optionally, the customer ID bits have to be specified before programming a PIC16C5X device on the ALLPRO.

Step 1: Write the application source code with the processor selection of your choice specified in the LIST assembler directive.

Example: LIST P=16C55

Step 2: Assemble your source code and debug with PICMASTER or MPSIM as required.

Programming PIC16C5X on Logical Devices' ALLPRO

Step 3: Modify your debugged source file:

- A. Change the processor selection in the LIST directive to:

```
P=16C62
```

This enables MPALC to accept addresses above the normal program memory space.

- B. Define customer ID code by adding before the END statement:

```
ORG    N           ;see Table 1 for value of N
DATA   ID3,ID2,ID1,ID0 ;four hex digits representing
                        ;the ID code

ORG    N+40
DATA   CONFIG      ;last three bits = config - see
                        ;Table 2
```

TABLE 1 - PIC16C5X ID CODE START ADDRESS

Device	"N"
PIC16C54	200 (hex)
PIC16C55	200 (hex)
PIC16C56	400 (hex)
PIC16C57	800 (hex)

TABLE 2 - PIC16C5X CONFIGURATION EPROM FUSES

Configuration*			Function
Bit 2	Bit 1	Bit 0	
1	X	X	WDT Enabled (Default)
0	X	X	WDT Disabled
X	0	0	LP Oscillator
X	0	1	XT Oscillator
X	1	0	HS Oscillator
X	1	1	RC Oscillator (Default)

* Bits 3-7 must always be 1.

Example:

```
ORG    200(HEX)    ;PIC16C54 or PIC16C55
DATA   0,0A,3,0F  ;ID code = 0A3F (hex)

ORG    240
DATA   0F9        ;WDT disabled, XT oscillator
```

- Step 4: Re-assemble the modified source code with MPALC, specifying the "merged hex" output file format (INHX8M) in the command line

Example: MPALC-F INHX8M <filename>
or the LIST directive of your source file.

Example: LIST P=16C62, F=INHX8M

- Step 5: Upload the data file into the ALLPRO RAM using the "READ DATA FILE" menu command..

- Step 6: Select "PROGRAM DEVICE" menu and handle code protection option as described in Step 4 from the section titled "Duplication of Master Device".

Alternative Method to Load ID Code and Configuration EPROM

You may skip Step 3 and edit the customer ID code and configuration EPROM directly in the ALLPRO RAM.

After downloading the file (without ID and configuration data) to the ALLPRO's RAM, use the "EDIT DATA" menu command to view the ALLPRO RAM. Push F8 to see and edit the ID locations. The ID can be entered directly on line 0. For example, the ID code ABCD would be entered as:

```
0 - 0a 00 0b 00 0c 00 0d 00...
```

where the first zero is the line number. Note that the INHX8M format has LSB first and MSB second so the code 0a 00 is actually 000a (the first ID code).

To edit the configuration EPROM push F8 again. Enter the configuration data into the data location per Table 2 above. The example (WDT disabled, crystal oscillator) would be:

```
0 - f9 0f
```

Note that the MSB is a "don't care" and will show a different value when data is loaded from the PC (00) than from a "master" device (0f).

*Author: Sumit Mitra
Logic Products Division*

Utility Math Routines

INTRODUCTION

This application note provides some utility math routines for Microchip's PIC16C5X series of 8-bit microcontrollers. The following math routines are provided:

- 8 x 8 unsigned multiply
- 16 x 16 double precision multiply
- 16 x 16 double precision divide
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- floating point multiplication
- floating point addition
- floating point subtraction
- BCD to binary conversion routines
- binary to BCD conversion routines
- BCD addition
- BCD subtraction
- square root

These are written in native assembly language and the listing files are provided. They are also available on a disk (MS-DOS®). All the routines provided can be called as subroutines. Most of the routines have two different versions: one optimized for speed and the other optimized for code size. The calling sequence of each routine is explained at the beginning of each listing file.

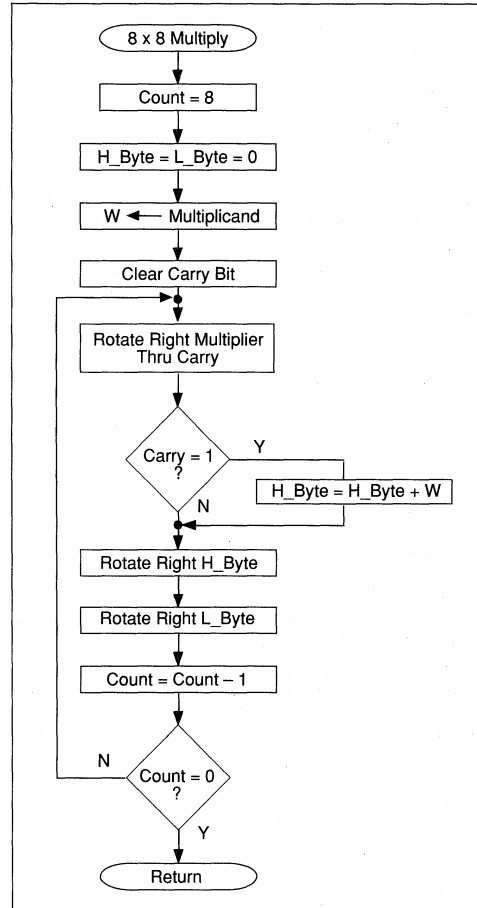
SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices A and B. The performance specs for the routines are shown in Table 1.

TABLE 1 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	35	37
Code Efficient	16	71

FIGURE 1 - FLOW CHART FOR UN SIGNED 8 X 8 MULTIPLY



PIC16C5X Math Routines

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of two 16-bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic are supported. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices C and D. The performance specs for the routines are shown in Table 2.

TABLE 2 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	240	233
Code Efficient	33	333

The code in Appendices C and D has been setup for unsigned arithmetic and the performance specs in the table above is for unsigned arithmetic. If signed arithmetic is desired, edit the line with "Signed equ FALSE" to "Signed equ TRUE" then re-assemble the code.

In case of signed multiply, both operands are assumed to be 16-bit 2's complement numbers.

Conditional assembly is supported by MPALC (PIC16C5X cross assembler) V2.1 or higher. If you have an older version, please contact the Microchip Technology sales office nearest you.

DOUBLE PRECISION DIVISION

This routine divides two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic are supported. Two routines are provided. One routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices E and F. The performance specs for the routines are shown in Table 3.

TABLE 3 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	370	263
Code Efficient	37	310

As can be seen from the above table, writing a speed efficient (straight line code) for this routine does not produce an efficient code. In this case, the looped code has a better overall performance (code size versus the speed). The code in Appendices E and F has been setup for unsigned arithmetic and the performance specs in the table above is for unsigned arithmetic. If signed arithmetic is desired, edit the line with "Signed equ FALSE" to "Signed equ TRUE" and then re-assemble the code. Signed division assumes two 16-bit 2's complement numbers.

DOUBLE PRECISION ADDITION & SUBTRACTION

This routine adds or subtracts two 16-bit numbers and produces a 16-bit result. This routine is used by other double precision routines. The listing of these routines is given in Appendix G. The performance specs for the routines are shown below:

TABLE 4 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Addition	7	8
Subtraction	14	17

FLOATING POINT ROUTINES

Four routines are implemented: addition, subtraction, multiplication and division. A floating point number is implemented as a number with 16-bit signed Mantissa and an 8-bit signed binary exponent, i.e., a number "Num" is represented as:

$$\text{Num} = \pm 16\text{-bit Mantissa} * (2^{**} \pm 8\text{-bit Exponent})$$

Also, a general purpose normalization routine is provided and it is recommended that the user call this routine as often as possible to avoid loss of precision. The normalization routine is written to normalize the value in locations "ACCbHI and ACCbLO". With minor modifications, the user may change this routine to normalize any two consecutive File registers (16 bits) by using indirect addressing scheme.

In case of multiplication, if a 32-bit result (with 8-bit exponent) is desired, then the user should edit the line "Mode16 equ TRUE" and change to "Mode16 equ FALSE".

No attempt has been made to optimize the code for speed. Only a looped version of the code is provided (see Appendix H). Also, only the code size (memory requirement) is shown in the table below and not the execution time because, the execution time varies significantly depending upon the value of the operands (and how they are normalized).

TABLE 5 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Addition	55	-
Subtraction	61	-
Multiplication	102	-

The flow charts of the algorithm for addition/subtraction and multiplication are shown in Figures 2, 3 and 4, respectively.

Please contact Microchip office for any code update.

FIGURE 2 - FLOW CHART FOR FLOATING POINT ADDITION AND SUBTRACTION

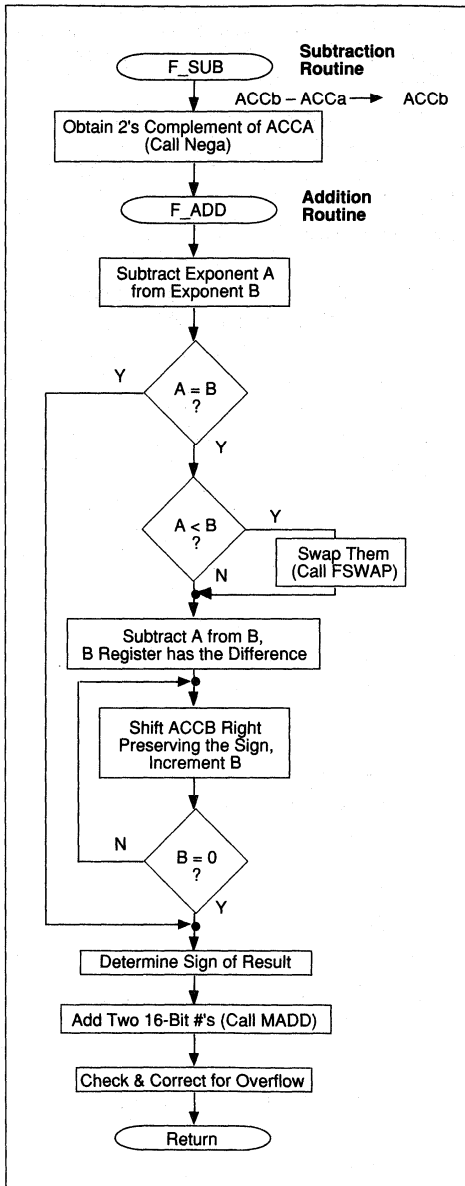
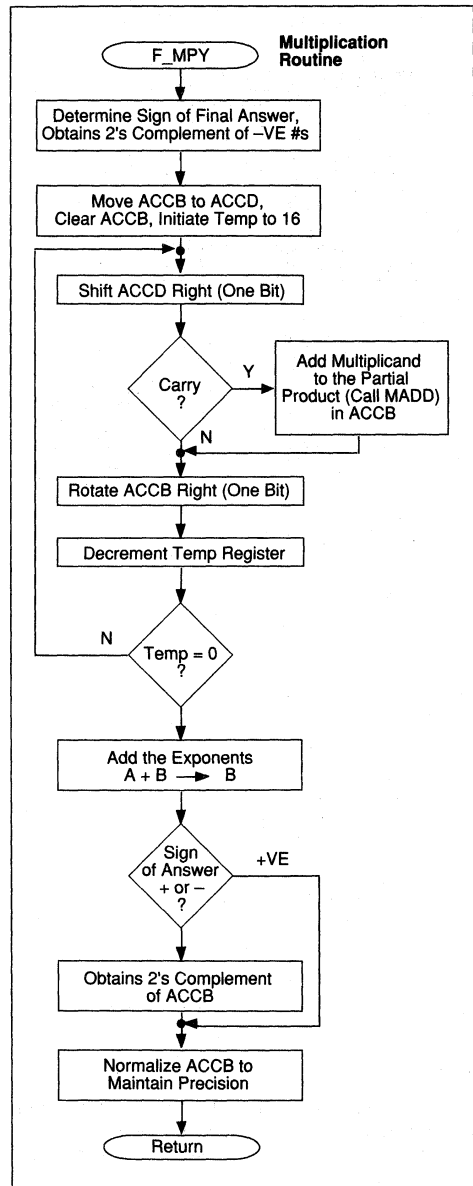
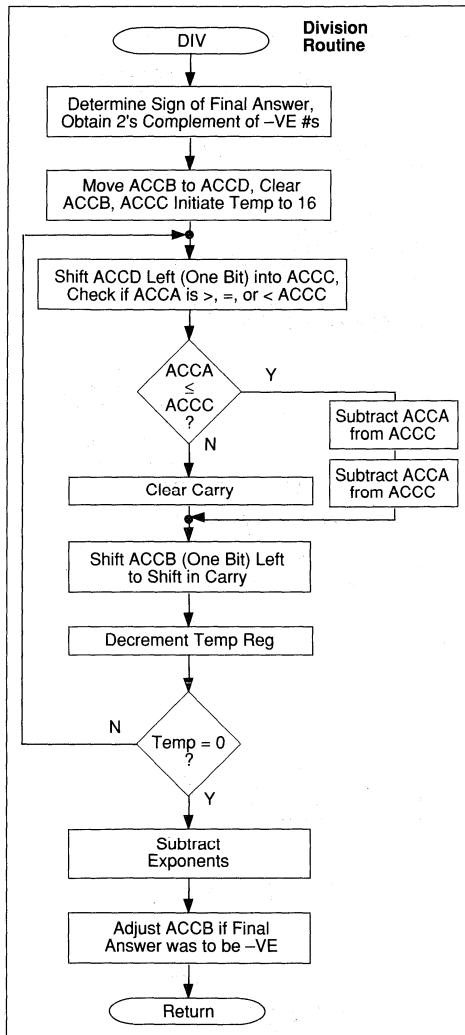


FIGURE 3 - FLOW CHART FOR FLOATING POINT MULTIPLICATION



PIC16C5X Math Routines

FIGURE 4 - FLOW CHART FOR FLOATING POINT DIVISION



BCD TO BINARY CONVERSION

This routine converts a five digit BCD number to a 16-bit binary number. The listing of this routine is given in Appendix I. The performance spec for the routine is shown below:

TABLE 6 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
BCD to Binary	30	121

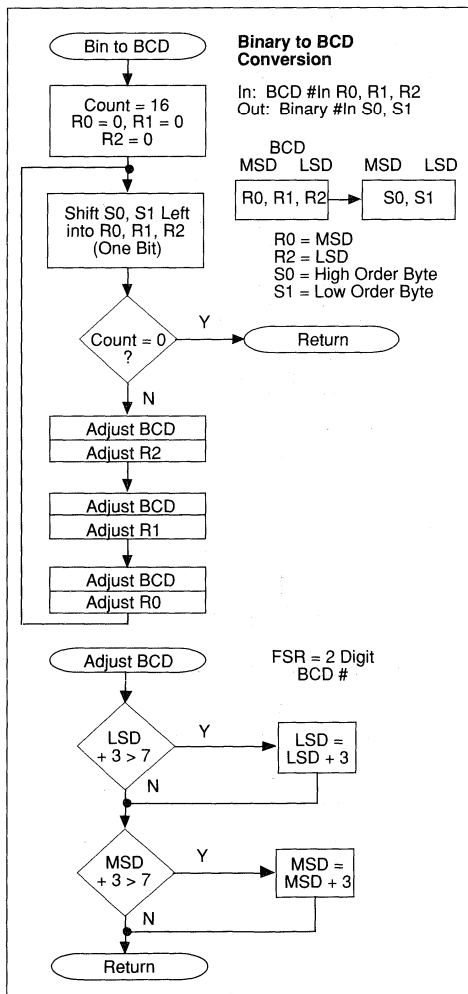
BINARY TO BCD CONVERSION

Two routines are provided: one routine converts a 16-bit binary number to a five digit BCD number and the other routine converts an 8-bit binary number to a two digit BCD number. The listing of these routines are given in Appendices J and K. The performance specs for the routines are shown below:

TABLE 7 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Binary (8-Bit) to BCD	10	81 (Worst Case)
Binary (16-Bit) to BCD	30	719

FIGURE 5 - FLOW CHART FOR BINARY TO BCD CONVERSION



BCD ADDITION & SUBTRACTION

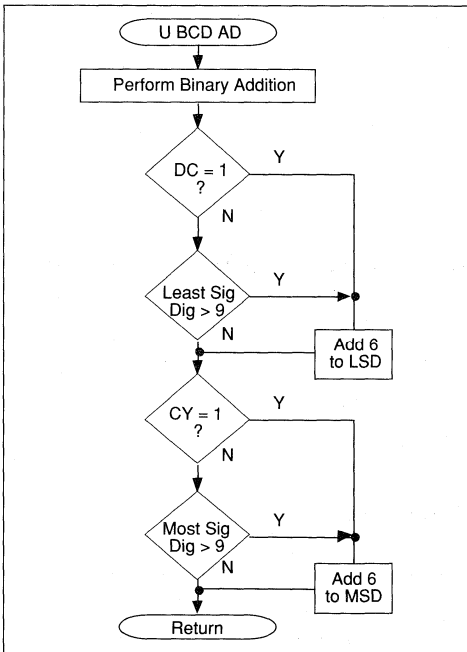
These two routines perform a two digit unsigned BCD addition and subtraction. The results are the sum (or difference) in one File register and with a overflow carry-bit in another File register. The performance specs for the routines are shown below:

TABLE 8 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
BCD Addition	29	23 (Worst Case)
BCD Subtraction	31	21 (Worst Case)

The listing files for the above two routines are given in Appendices L and M. The flow charts for BCD addition and BCD subtraction are given in Figures 6 and 7, respectively.

FIGURE 6 - FLOW CHART FOR BCD ADDITION

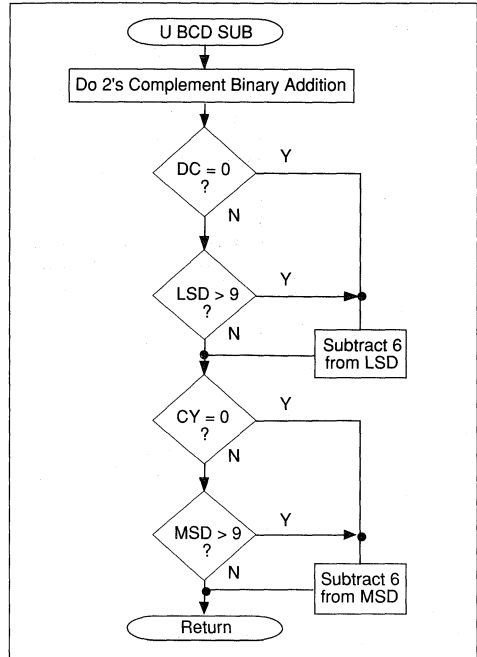


SQUARE ROOT

Often in many applications one needs to find the square root of a number. Of many numerical methods to find the square root of a number, the Newton-Raphson method is very attractive because of its fast convergence rate. In this method the square root of a number, "N", is obtained from the approximate solution of:

$$f(Y) = Y^2 - N = 0$$

FIGURE 7 - FLOW CHART FOR BCD SUBTRACTION



The function "f(Y)" can be expanded about Y_0 using first order Taylor polynomial expansion as:

$$\text{Equation 1: } f(Y) = f(Y_0) + (Y - Y_0) f'(Y_0) + (Y - Y_0)^2 f''(Y_0) / 2! + \dots$$

If X is a root of f(Y), then f(X) = 0:

$$f(X) = f(Y_0) + (X - Y_0) f'(Y_0) + (X - Y_0)^2 f''(Y_0) / 2! + \dots = 0$$

If Y_0 is an approximate root of f(Y), then higher order terms are negligible. Therefore:

$$\text{Equation 2: } f(Y_0) + (X - Y_0) f'(Y_0)$$

$$\text{i.e., } X = Y_0 - f(Y_0) / f'(Y_0)$$

Thus, X is a better approximation for Y_0 . From this, the sequence $\{X_n\}$ can be generated:

$$\text{Equation 3: } X_n = X_{n-1} - f(X_{n-1}) / f'(X_{n-1}), n \geq 1$$

From equation 1 and equation 3 we get,

$$\text{Equation 4: } X_n = 0.5 * (X_{n-1} + N/X_{n-1})$$

An assembly language subroutine implementing the above algorithm is given in Figure 1.8. The initial approximate root of N is taken to be $N/2$. If the approximate range of N is known a priori then the total number of iterations may be cut down by starting with a better approximate root than $N/2$.

This program, as listed in Appendix N, computes the square root of a 16-bit number. This routine uses a double precision math routine (division and addition) as described in the previous pages of this application note.



PIC16C5X Math Routines

APPENDIX A

MPASM B0.54

PAGE 1

```
*****
;
;           8x8 Software Multiplier
;           ( Code Efficient : Looped Code )
*****
;
;   The 16 bit result is stored in 2 bytes
;
;   Before calling the subroutine " mpy ", the multiplier should
;   be loaded in location " mulplr ", and the multiplicand in
;   " mulcnd ". The 16 bit result is stored in locations
;   H_byte & L_byte.
;
;   Performance :
;
;           Program Memory : 15 locations
;           # of cycles    : 71
;           Scratch RAM    : 0 locations
;
;   This routine is optimized for code efficiency ( looped code )
;   For time efficiency code refer to "mult8x8F.asm" ( straight line code )
*****
;
;   LIST      P=16C54
0009      mulcnd equ 09      ; 8 bit multiplicand
0010      mulplr equ 10      ; 8 bit multiplier
0012      H_byte equ 12      ; High byte of the 16 bit result
0013      L_byte equ 13      ; Low byte of the 16 bit result
0014      count equ 14      ; loop counter
;
;
;   include      "mpreg.h"
;   ***** PIC16C5X Header *****
01FF      PIC54 equ 1FFh    ; Define Reset Vectors
01FF      PIC55 equ 1FFh
03FF      PIC56 equ 3FFh
07FF      PIC57 equ 7FFh
;
0001      RTCC equ 1h
0002      PC equ 2h
0003      STATUS equ 3h    ; F3 Reg is STATUS Reg.
0004      FSR equ 4h
;
0005      Port_A equ 5h
0006      Port_B equ 6h    ; I/O Port Assignments
0007      Port_C equ 7h
;
;   *****
;
;   ; STATUS REG. Bits
0000      CARRY equ 0h    ; Carry Bit is Bit.0 of F3
0000      C equ 0h
0001      DCARRY equ 1h
0001      DC equ 1h
0002      Z_bit equ 2h    ; Bit 2 of F3 is Zero Bit
0002      Z equ 2h
0003      P_DOWN equ 3h
0003      PD equ 3h
0004      T_OUT equ 4h
0004      TO equ 4h
0005      PA0 equ 5h
0006      PA1 equ 6h
0007      PA2 equ 7h
```

PIC16C5X Math Routines

```
0001      ; Same equ 1h
0000      ;
0007      ; LSB equ 0h
0007      ; MSB equ 7h
0001      ;
0001      ; TRUE equ 1h
0001      ; YES equ 1h
0000      ; FALSE equ 0h
0000      ; NO equ 0h
0000      ;
;*****

; ***** Begin Multiplier Routine
0000 0072 mpy_S  clrf  H_byte
0001 0073      clrf  L_byte
0002 0C08      movlw 8
0003 0034      movwf count
0004 0209      movf  mulcnd,w
0005 0403      bcf  STATUS,CARRY ; Clear the carry bit in the status Reg.
0006 0330      loop  rrf  mulplr
0007 0603      btfs  STATUS,CARRY
0008 01F2      addwf H_byte,Same
0009 0332      rrf  H_byte,Same
000A 0333      rrf  L_byte,Same
000B 02F4      decfsz count
000C 0A06      goto  loop
000D 0800      ; retlw 0
; *****
; ***** Test Program *****
000E 0CFF      main  movlw 0FF
000F 0030      movwf mulplr ; multiplier (in mulplr) = 0FF
0010 0CFF      movlw 0FF ; multiplicand(W Reg) = 0FF
0011 0029      movwf mulcnd
;
0012 0900      call  mpy_S ; The result 0FF*0FF = FE01 is in locations
; ; H_byte & L_byte
0013 0A13      self  goto  self
;
01FF 0A0E      org  01FF
; goto  main
;
; END

Errors : 0
Warnings : 0
```

PIC16C5X Math Routines

APPENDIX B

MPASM B0.54

PAGE 1

```
*****
;
;           8x8 Software Multiplier
;           ( Fast Version : Straight Line Code )
*****
;
;   The 16 bit result is stored in 2 bytes
;
;   Before calling the subroutine " mpy ", the multiplier should
;   be loaded in location " mulplr ", and the multiplicand in
;   " mulcnd ". The 16 bit result is stored in locations
;   H_byte & L_byte.
;
;   Performance :
;               Program Memory : 35 locations
;               # of cycles    : 37
;               Scratch RAM   : 0 locations
;
;   This routine is optimized for speed efficiency ( straight line code )
;   For code efficiency, refer to "mult8x8S.asm" ( looped code )
*****
;
;   LIST      P=16C54
0009      mulcnd equ 09      ; 8 bit multiplicand
0010      mulplr equ 10      ; 8 bit multiplier
0012      H_byte equ 12      ; High byte of the 16 bit result
0013      L_byte equ 13      ; Low byte of the 16 bit result
;
;
;   include   "picreg.h"
;*****
01FF      PIC54 equ 1FFH    ; Define Reset Vectors
01FF      PIC55 equ 1FFH
03FF      PIC56 equ 3FFH
07FF      PIC57 equ 7FFH
;
0001      RTCC equ 1h
0002      PC equ 2h
0003      STATUS equ 3h    ; F3 Reg is STATUS Reg.
0004      FSR equ 4h
;
0005      Port_A equ 5h
0006      Port_B equ 6h    ; I/O Port Assignments
0007      Port_C equ 7h
;
;*****
;
;           ; STATUS REG. Bits
0000      CARRY equ 0h    ; Carry Bit is Bit.0 of F3
0000      C equ 0h
0001      DCARRY equ 1h
0001      DC equ 1h
0002      Z_bit equ 2h    ; Bit 2 of F3 is Zero Bit
0002      Z equ 2h
0003      P_DOWN equ 3h
0003      PD equ 3h
0004      T_OUT equ 4h
0004      TO equ 4h
0005      PA0 equ 5h
0006      PA1 equ 6h
0007      PA2 equ 7h
;
0001      Same equ 1h
;
;*****
```


PIC16C5X Math Routines

```

0000      LSB      equ    0h
0007      MSB      equ    7h
;
0001      TRUE     equ    1h
0001      YES      equ    1h
0000      FALSE   equ    0h
0000      NO       equ    0h
;
;*****
;
;**** Define a macro for adding & right shifting **
;
mult      MACRO    bit                ; Begin macro
      btfsc    mulplr,bit
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same
      ENDM
; End of macro
;
; ***** Begin Multiplier Routine
0000 0072  mpy_F   clrf    H_byte
0001 0073      clrf    L_byte
0002 0209      movf   mulcnd,w        ; move the multiplicand to W reg.
0003 0403      bcf    STATUS,CARRY   ; Clear the carry bit in the status Reg.
;
mult      0
      btfsc    mulplr,0
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

mult      1
      btfsc    mulplr,1
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

mult      2
      btfsc    mulplr,2
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

mult      3
      btfsc    mulplr,3
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

mult      4
      btfsc    mulplr,4
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

mult      5
      btfsc    mulplr,5
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

mult      6
      btfsc    mulplr,6
      addwf   H_byte,Same
      rrf     H_byte,Same
      rrf     L_byte,Same

```

PIC16C5X Math Routines

```

                                mult    7
0020 06F0                btfsc    mulplr,7
0021 01F2                addwf    H_byte,Same
0022 0332                rrf      H_byte,Same
0023 0333                rrf      L_byte,Same

                                ;
0024 0800                retlw   0
                                ;
                                ;*****
                                ;          Test Program
                                ;*****
0025 0CFF                main     movlw   0FF
0026 0030                movwf  mulplr      ; multiplier (in mulplr)   = 0FF
0027 0CFF                movlw   0FF
0028 0029                movwf  mulcnd     ; multiplicand(in mulcnd ) = 0FF
                                ;
0029 0900                call   mpy_F      ; The result 0FF*0FF = FE01 is in locations
                                ;          ; H_byte & L_byte

002A 0A2A                self    goto   self
                                ;
                                org    01FF
01FF 0A25                goto   main
                                ;
                                END

Errors   :    0
Warnings :    0
```

PIC16C5X Math Routines

APPENDIX C: DOUBLE PRECISION MULTIPLICATION LISTING (LOOPED)

MPASM B0.54

PAGE 1

```

;*****
;
;           Double Precision Multiplication
;
;           ( Optimized for Code Size : Looped Code )
;
;*****;
; Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_mpy
; (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
;
; Performance :
; Program Memory :      033
; Clock Cycles   :      333
;
; Note : The above timing is the worst case timing, when the
;        register ACCb = FFFF. The speed may be improved if
;        the register ACCb contains a number ( out of the two
;        numbers ) with less number of 1s.
;        The performance specs are for Unsigned arithmetic ( i.e.,
;        with "SIGNED equ FALSE ").
;
;        The performance specs are for Unsigned arithmetic ( i.e.,
;        with "SIGNED equ FALSE ").
;*****;
;
; LIST      P=16C54
0010      ACCaLO equ    10
0011      ACCaHI equ    11
0012      ACCbLO equ    12
0013      ACCbHI equ    13
0014      ACCcLO equ    14
0015      ACCcHI equ    15
0016      ACCdLO equ    16
0017      ACCdHI equ    17
0018      temp  equ    18
0019      sign  equ    19
;
; include "picreg.h"
;*****; PIC16C5X Header *****
01FF      PIC54 equ    1FFh ; Define Reset Vectors
01FF      PIC55 equ    1FFh
03FF      PIC56 equ    3FFh
07FF      PIC57 equ    7FFh
;
0001      RTCC  equ    1h
0002      PC   equ    2h
0003      STATUS equ    3h ; F3 Reg is STATUS Reg.
0004      FSR  equ    4h
;
0005      Port_A equ    5h
0006      Port_B equ    6h ; I/O Port Assignments
0007      Port_C equ    7h
;
;*****;
;
; ; STATUS REG. Bits
0000      CARRY equ    0h ; Carry Bit is Bit.0 of F3
0000      C     equ    0h
0001      DCARRY equ    1h
0001      DC     equ    1h
0002      Z_bit equ    2h ; Bit 2 of F3 is Zero Bit
0002      Z     equ    2h

```

PIC16C5X Math Routines

```

0003      P_DOWN equ    3h
0003      PD      equ    3h
0004      T_OUT   equ    4h
0004      TO      equ    4h
0005      PA0     equ    5h
0006      PA1     equ    6h
0007      PA2     equ    7h
;
0001      Same    equ    1h
;
0000      LSB     equ    0h
0007      MSB     equ    7h
;
0001      TRUE    equ    1h
0001      YES     equ    1h
0000      FALSE   equ    0h
0000      NO      equ    0h
;
;*****

                org    0
;*****
0001      SIGNED equ    TRUE        ; Set This To 'TRUE' if the routines
;                                     ; for Multiplication & Division needs
;                                     ; to be assembled as Signed Integer
;                                     ; Routines. If 'FALSE' the above two
;                                     ; routines ( D_mpy & D_div ) use
;                                     ; unsigned arithmetic.
;*****
;          Double Precision Subtraction ( ACCb - ACCa -> ACCb )
;
0000 0210      D_add movf   ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
0001 01F2      addwf  ACCbLO    ;add lsb
0002 0603      btfsc  STATUS,CARRY ;add in carry
0003 02B3      incf   ACCbHI
0004 0211      movf   ACCaHI,w
0005 01F3      addwf  ACCbHI    ;add msb
0006 0800      retlw  0
;*****
;          Double Precision Multiply ( 16x16 -> 32 )
;          ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
;          in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
;
D_mpyS      ;results in ACCb(16 msb's) and ACCc(16
;
                IF SIGNED
0007 0930      CALL   S_SIGN
                ENDIF
;
0008 0921      call   setup
0009 0337      rrf    ACCdHI    ;rotate d right
000A 0336      rrf    ACCdLO
000B 0603      btfsc  STATUS,CARRY ;need to add?
000C 0900      call   D_add
000D 0333      rrf    ACCbHI
000E 0332      rrf    ACCbLO
000F 0335      rrf    ACCcHI
0010 0334      rrf    ACCcLO
0011 02F8      decfsz temp      ;loop until all bits checked
0012 0A09      goto   mloop

```

PIC16C5X Math Routines

```

;
; IF SIGNED
0013 07F9      btfss    sign,MSB
0014 0800      retlw    0
0015 0274      comf    ACCcLO      ; negate ACCa ( -ACCa -> ACCa )
0016 02B4      incf    ACCcLO
0017 0643      btfsc    STATUS,Z_bit
0018 00F5      decf    ACCcHI
0019 0275      comf    ACCcHI
001A 0643      btfsc    STATUS,Z_bit
001B 0272      neg_B    comf    ACCbLO      ; negate ACCb
001C 02B2      incf    ACCbLO
001D 0643      btfsc    STATUS,Z_bit
001E 00F3      decf    ACCbHI
001F 0273      comf    ACCbHI
0020 0800      retlw    0
;
; ELSE
; retlw    0
; ENDIF
;
;*****
;
0021 0C10      setup    movlw    .16      ; for 16 shifts
0022 0038      movwf    temp
0023 0213      movf    ACCbHI,w      ;move ACCb to ACCd
0024 0037      movwf    ACCdHI
0025 0212      movf    ACCbLO,w
0026 0036      movwf    ACCdLO
0027 0073      clrf    ACCbHI
0028 0072      clrf    ACCbLO
0029 0800      retlw    0
;
;*****
;
002A 0270      neg_A    comf    ACCaLO      ; negate ACCa ( -ACCa -> ACCa )
002B 02B0      incf    ACCaLO
002C 0643      btfsc    STATUS,Z_bit
002D 00F1      decf    ACCaHI
002E 0271      comf    ACCaHI
002F 0800      retlw    0
;
;*****
; Assemble this section only if Signed Arithmetic Needed
;
; IF SIGNED
;
; S_SIGN
0030 0211      S_SIGN  movf    ACCaHI,W
0031 0193      xorwf    ACCbHI,W
0032 0039      movwf    sign
0033 07F3      btfss    ACCbHI,MSB      ; if MSB set go & negate ACCb
0034 0A3A      goto    chek_A
;
; comf    ACCbLO      ; negate ACCb
0035 0272      incf    ACCbLO
0036 02B2      btfsc    STATUS,Z_bit
0037 0643      decf    ACCbHI
0038 00F3      comf    ACCbHI
0039 0273
;
; chek_A
003A 07F1      chek_A  btfss    ACCaHI,MSB      ; if MSB set go & negate ACCa
003B 0800      retlw    0
003C 0A2A      goto    neg_A
;
; ENDIF

```

PIC16C5X Math Routines

```

;
;*****
;                                     Test Program
;*****
;   Load constant values to ACCa & ACCb for testing
;
003D 0C01      main    movlw   1
003E 0031      movwf  ACCaHI
003F 0CFF      movlw   0FF          ; loads ACCa = 01FF
0040 0030      movwf  ACCaLO
;
0041 0C7F      movlw   07F
0042 0033      movwf  ACCbHI
0043 0CFF      movlw   0FF          ; loads ACCb = 7FFF
0044 0032      movwf  ACCbLO
;
0045 0907      call   D_mpyS        ; Here (ACCb,ACCc) = 00FF 7E01
;
0046 0A46      self   goto   self
;
;               org   PIC54
01FF 0A3D      goto   main
;               END
;*****
```

```
Errors   :    0
Warnings :    0
```

PIC16C5X Math Routines

APPENDIX D: DOUBLE PRECISION MULTIPLICATION LISTINGS (FAST)

MPASM B0.54

PAGE 1

```
*****
;
;           Double Precision Multiplication
;
;           ( Optimized for Speed : straight Line Code )
;
;*****;
; Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_mpy
; (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
;
; Performance :
;           Program Memory :      240
;           Clock Cycles   :      233
;
; Note : The above timing is the worst case timing, when the
;        register ACCb = FFFF. The speed may be improved if
;        the register ACCb contains a number ( out of the two
;        numbers ) with less number of 1s.
;
;        The performance specs are for Unsigned arithmetic ( i.e.,
;        with "SIGNED equ FALSE ").
;*****;
;
; LIST      p=16c54
0010      ACCaLO equ    10
0011      ACCaHI equ    11
0012      ACCbLO equ    12
0013      ACCbHI equ    13
0014      ACCcLO equ    14
0015      ACCcHI equ    15
0016      ACCdLO equ    16
0017      ACCdHI equ    17
0018      temp  equ    18
0019      sign  equ    19
;
; include "mpreg.h"
;*****;
; PIC16C5X Header *****
01FF      PIC54 equ    1FFh ; Define Reset Vectors
01FF      PIC55 equ    1FFh
03FF      PIC56 equ    3FFh
07FF      PIC57 equ    7FFh
;
0001      RTCC  equ    1h
0002      PC    equ    2h
0003      STATUS equ    3h ; F3 Reg is STATUS Reg.
0004      FSR   equ    4h
;
0005      Port_A equ    5h
0006      Port_B equ    6h ; I/O Port Assignments
0007      Port_C equ    7h
;
;*****;
;
;           ; STATUS REG. Bits
0000      CARRY equ    0h ; Carry Bit is Bit.0 of F3
0000      C      equ    0h
0001      DCARRY equ    1h
0001      DC     equ    1h
0002      Z_bit  equ    2h ; Bit 2 of F3 is Zero Bit
0002      Z      equ    2h
0003      P_DOWN equ    3h
```



PIC16C5X Math Routines

```

0003      PD      equ      3h
0004      T_OUT   equ      4h
0004      TO      equ      4h
0005      PA0     equ      5h
0006      PA1     equ      6h
0007      PA2     equ      7h
;
0001      Same    equ      1h
;
0000      LSB     equ      0h
0007      MSB     equ      7h
;
0001      TRUE    equ      1h
0001      YES     equ      1h
0000      FALSE   equ      0h
0000      NO      equ      0h
;
;*****
;
;          org      0
;*****
0000      SIGNED  equ      FALSE      ; Set This To 'TRUE' if the routines
;                                     ; for Multiplication & Division needs
;                                     ; to be assembled as Signed Integer
;                                     ; Routines. If 'FALSE' the above two
;                                     ; routines ( D_mpy & D_div ) use
;                                     ; unsigned arithmetic.
;*****
;          multiplication macro
;
mulMac    MACRO
LOCAL    NO_ADD
;
;          rrf      ACCdHI      ;rotate d right
;          rrf      ACCdLO
;          btffs    STATUS,CARRY ;need to add?
;          goto     NO_ADD      ; no addition necessary
;          movf     ACCaLO,w     ; Addition ( ACCb + ACCa -> ACCb )
;          addwf    ACCbLO      ;add lsb
;          btfs    STATUS,CARRY ;add in carry
;          incf     ACCbHI
;          movf     ACCaHI,w
;          addwf    ACCbHI      ;add msb
NO_ADD    rrf      ACCbHI
;          rrf      ACCbLO
;          rrf      ACCcHI
;          rrf      ACCcLO
;
;          ENDM
;
;*****
;          Double Precision Multiply ( 16x16 -> 32 )
;          ( ACCb*ACCa -> ACCb,ACCc ) ; 32 bit output with high word
;          in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
;
;          D_mpyF      ;results in ACCb(16 msb's) and ACCc(16 lsb's)
;
;          IF      SIGNED
;          CALL     S_SIGN
;          ENDF
;
0000 09E2      call    setup
;

```


PIC16C5X Math Routines

```

; use the mulMac macro 16 times
;
mulMac
    LOCAL    NO_ADD
;
0001 0337        rrf    ACCdHI        ;rotate d right
0002 0336        rrf    ACCdLO
0003 0703        btfss  STATUS,CARRY  ;need to add?
0004 0A0B        goto   NO_ADD        ; no addition necessary
0005 0210        movf   ACCaLO,w      ; Addition ( ACCb + ACCa -> ACCb )
0006 01F2        addwf  ACCbLO        ;add lsb
0007 0603        btfsc  STATUS,CARRY  ;add in carry
0008 02B3        incf   ACCbHI
0009 0211        movf   ACCaHI,w
000A 01F3        addwf  ACCbHI        ;add msb
000B 0333        NO_ADD rrf    ACCbHI
000C 0332        rrf    ACCbLO
000D 0335        rrf    ACCcHI
000E 0334        rrf    ACCcLO
;

mulMac
    LOCAL    NO_ADD
;
000F 0337        rrf    ACCdHI        ;rotate d right
0010 0336        rrf    ACCdLO
0011 0703        btfss  STATUS,CARRY  ;need to add?
0012 0A19        goto   NO_ADD        ; no addition necessary
0013 0210        movf   ACCaLO,w      ; Addition ( ACCb + ACCa -> ACCb )
0014 01F2        addwf  ACCbLO        ;add lsb
0015 0603        btfsc  STATUS,CARRY  ;add in carry
0016 02B3        incf   ACCbHI
0017 0211        movf   ACCaHI,w
0018 01F3        addwf  ACCbHI        ;add msb
0019 0333        NO_ADD rrf    ACCbHI
001A 0332        rrf    ACCbLO
001B 0335        rrf    ACCcHI
001C 0334        rrf    ACCcLO
;

mulMac
    LOCAL    NO_ADD
;
001D 0337        rrf    ACCdHI        ;rotate d right
001E 0336        rrf    ACCdLO
001F 0703        btfss  STATUS,CARRY  ;need to add?
0020 0A27        goto   NO_ADD        ; no addition necessary
0021 0210        movf   ACCaLO,w      ; Addition ( ACCb + ACCa -> ACCb )
0022 01F2        addwf  ACCbLO        ;add lsb
0023 0603        btfsc  STATUS,CARRY  ;add in carry
0024 02B3        incf   ACCbHI
0025 0211        movf   ACCaHI,w
0026 01F3        addwf  ACCbHI        ;add msb
0027 0333        NO_ADD rrf    ACCbHI
0028 0332        rrf    ACCbLO
0029 0335        rrf    ACCcHI
002A 0334        rrf    ACCcLO
;

```



PIC16C5X Math Routines

```

mulMac
LOCAL NO_ADD
;
002B 0337      rrf      ACCdHI      ;rotate d right
002C 0336      rrf      ACCdLO
002D 0703      btfs    STATUS,CARRY ;need to add?
002E 0A35      goto    NO_ADD      ; no addition necessary
002F 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
0030 01F2      addwf   ACCbLO          ;add lsb
0031 0603      btfs    STATUS,CARRY ;add in carry
0032 02B3      incf    ACCbHI
0033 0211      movf    ACCaHI,w
0034 01F3      addwf   ACCbHI          ;add msb
NO_ADD rrf      ACCbHI
0036 0332      rrf      ACCbLO
0037 0335      rrf      ACCcHI
0038 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
0039 0337      rrf      ACCdHI      ;rotate d right
003A 0336      rrf      ACCdLO
003B 0703      btfs    STATUS,CARRY ;need to add?
003C 0A43      goto    NO_ADD      ; no addition necessary
003D 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
003E 01F2      addwf   ACCbLO          ;add lsb
003F 0603      btfs    STATUS,CARRY ;add in carry
0040 02B3      incf    ACCbHI
0041 0211      movf    ACCaHI,w
0042 01F3      addwf   ACCbHI          ;add msb
NO_ADD rrf      ACCbHI
0043 0333      rrf      ACCbLO
0044 0332      rrf      ACCcHI
0045 0335      rrf      ACCcLO
0046 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
0047 0337      rrf      ACCdHI      ;rotate d right
0048 0336      rrf      ACCdLO
0049 0703      btfs    STATUS,CARRY ;need to add?
004A 0A51      goto    NO_ADD      ; no addition necessary
004B 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
004C 01F2      addwf   ACCbLO          ;add lsb
004D 0603      btfs    STATUS,CARRY ;add in carry
004E 02B3      incf    ACCbHI
004F 0211      movf    ACCaHI,w
0050 01F3      addwf   ACCbHI          ;add msb
NO_ADD rrf      ACCbHI
0051 0333      rrf      ACCbLO
0052 0332      rrf      ACCcHI
0053 0335      rrf      ACCcLO
0054 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
0055 0337      rrf      ACCdHI      ;rotate d right
0056 0336      rrf      ACCdLO
0057 0703      btfs    STATUS,CARRY ;need to add?
0058 0A5F      goto    NO_ADD      ; no addition necessary
0059 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
005A 01F2      addwf   ACCbLO          ;add lsb
005B 0603      btfs    STATUS,CARRY ;add in carry
005C 02B3      incf    ACCbHI
005D 0211      movf    ACCaHI,w
005E 01F3      addwf   ACCbHI          ;add msb

```

PIC16C5X Math Routines

```

005F 0333      NO_ADD rrf      ACCbHI
0060 0332      rrf      ACCbLO
0061 0335      rrf      ACCcHI
0062 0334      rrf      ACCcLO
;
      mulMac
      LOCAL NO_ADD
;
0063 0337      rrf      ACCdHI      ;rotate d right
0064 0336      rrf      ACCdLO
0065 0703      btfs    STATUS,CARRY ;need to add?
0066 0A6D      goto    NO_ADD      ; no addition necessary
0067 0210      movf    ACCaLO,w     ; Addition ( ACCb + ACCa -> ACCb )
0068 01F2      addwf   ACCbLO      ;add lsb
0069 0603      btfs    STATUS,CARRY ;add in carry
006A 02B3      incf    ACCbHI
006B 0211      movf    ACCaHI,w
006C 01F3      addwf   ACCbHI      ;add msb
006D 0333      NO_ADD rrf      ACCbHI
006E 0332      rrf      ACCbLO
006F 0335      rrf      ACCcHI
0070 0334      rrf      ACCcLO
;
      mulMac
      LOCAL NO_ADD
;
0071 0337      rrf      ACCdHI      ;rotate d right
0072 0336      rrf      ACCdLO
0073 0703      btfs    STATUS,CARRY ;need to add?
0074 0A7B      goto    NO_ADD      ; no addition necessary
0075 0210      movf    ACCaLO,w     ; Addition ( ACCb + ACCa -> ACCb )
0076 01F2      addwf   ACCbLO      ;add lsb
0077 0603      btfs    STATUS,CARRY ;add in carry
0078 02B3      incf    ACCbHI
0079 0211      movf    ACCaHI,w
007A 01F3      addwf   ACCbHI      ;add msb
007B 0333      NO_ADD rrf      ACCbHI
007C 0332      rrf      ACCbLO
007D 0335      rrf      ACCcHI
007E 0334      rrf      ACCcLO
;
      mulMac
      LOCAL NO_ADD
;
007F 0337      rrf      ACCdHI      ;rotate d right
0080 0336      rrf      ACCdLO
0081 0703      btfs    STATUS,CARRY ;need to add?
0082 0A89      goto    NO_ADD      ; no addition necessary
0083 0210      movf    ACCaLO,w     ; Addition ( ACCb + ACCa -> ACCb )
0084 01F2      addwf   ACCbLO      ;add lsb
0085 0603      btfs    STATUS,CARRY ;add in carry
0086 02B3      incf    ACCbHI
0087 0211      movf    ACCaHI,w
0088 01F3      addwf   ACCbHI      ;add msb
0089 0333      NO_ADD rrf      ACCbHI
008A 0332      rrf      ACCbLO
008B 0335      rrf      ACCcHI
008C 0334      rrf      ACCcLO
;

```

PIC16C5X Math Routines

```
mulMac
LOCAL NO_ADD
;
008D 0337      rrf      ACCdHI      ;rotate d right
008E 0336      rrf      ACCdLO
008F 0703      btfs    STATUS,CARRY    ;need to add?
0090 0A97      goto    NO_ADD      ; no addition necessary
0091 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
0092 01F2      addwf   ACCbLO      ;add lsb
0093 0603      btfs    STATUS,CARRY    ;add in carry
0094 02B3      incf    ACCbHI
0095 0211      movf    ACCaHI,w
0096 01F3      addwf   ACCbHI      ;add msb
NO_ADD rrf      ACCbHI
0097 0333      rrf      ACCbLO
0098 0332      rrf      ACCcHI
0099 0335      rrf      ACCcLO
009A 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
009B 0337      rrf      ACCdHI      ;rotate d right
009C 0336      rrf      ACCdLO
009D 0703      btfs    STATUS,CARRY    ;need to add?
009E 0AA5      goto    NO_ADD      ; no addition necessary
009F 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
00A0 01F2      addwf   ACCbLO      ;add lsb
00A1 0603      btfs    STATUS,CARRY    ;add in carry
00A2 02B3      incf    ACCbHI
00A3 0211      movf    ACCaHI,w
00A4 01F3      addwf   ACCbHI      ;add msb
NO_ADD rrf      ACCbHI
00A5 0333      rrf      ACCbLO
00A6 0332      rrf      ACCcHI
00A7 0335      rrf      ACCcLO
00A8 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
00A9 0337      rrf      ACCdHI      ;rotate d right
00AA 0336      rrf      ACCdLO
00AB 0703      btfs    STATUS,CARRY    ;need to add?
00AC 0AB3      goto    NO_ADD      ; no addition necessary
00AD 0210      movf    ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
00AE 01F2      addwf   ACCbLO      ;add lsb
00AF 0603      btfs    STATUS,CARRY    ;add in carry
00B0 02B3      incf    ACCbHI
00B1 0211      movf    ACCaHI,w
00B2 01F3      addwf   ACCbHI      ;add msb
NO_ADD rrf      ACCbHI
00B3 0333      rrf      ACCbLO
00B4 0332      rrf      ACCcHI
00B5 0335      rrf      ACCcLO
00B6 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
00B7 0337      rrf      ACCdHI      ;rotate d right
00B8 0336      rrf      ACCdLO
00B9 0703      btfs    STATUS,CARRY    ;need to add?
00BA 0AC1      goto    NO_ADD      ;no addition necessary
00BB 0210      movf    ACCaLO,w    ;Addition ( ACCb + ACCa -> ACCb )
00BC 01F2      addwf   ACCbLO      ;add lsb
00BD 0603      btfs    STATUS,CARRY    ;add in carry
00BE 02B3      incf    ACCbHI
00BF 0211      movf    ACCaHI,w
00C0 01F3      addwf   ACCbHI      ;add msb
```

PIC16C5X Math Routines

```

00C1 0333      NO_ADD rrf      ACCbHI
00C2 0332      rrf      ACCbLO
00C3 0335      rrf      ACCcHI
00C4 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
00C5 0337      rrf      ACCdHI      ;rotate d right
00C6 0336      rrf      ACCdLO
00C7 0703      btffs     STATUS,CARRY ;need to add?
00C8 0ACF      goto     NO_ADD      ; no addition necessary
00C9 0210      movf     ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
00CA 01F2      addwf    ACCbLO      ;add lsb
00CB 0603      btfs    STATUS,CARRY ;add in carry
00CC 02B3      incf     ACCbHI
00CD 0211      movf     ACCaHI,w
00CE 01F3      addwf    ACCbHI      ;add msb
00CF 0333      NO_ADD rrf      ACCbHI
00D0 0332      rrf      ACCbLO
00D1 0335      rrf      ACCcHI
00D2 0334      rrf      ACCcLO
;
mulMac
LOCAL NO_ADD
;
00D3 0337      rrf      ACCdHI      ;rotate d right
00D4 0336      rrf      ACCdLO
00D5 0703      btffs     STATUS,CARRY ;need to add?
00D6 0ADD      goto     NO_ADD      ; no addition necessary
00D7 0210      movf     ACCaLO,w    ; Addition ( ACCb + ACCa -> ACCb )
00D8 01F2      addwf    ACCbLO      ;add lsb
00D9 0603      btfs    STATUS,CARRY ;add in carry
00DA 02B3      incf     ACCbHI
00DB 0211      movf     ACCaHI,w
00DC 01F3      addwf    ACCbHI      ;add msb
00DD 0333      NO_ADD rrf      ACCbHI
00DE 0332      rrf      ACCbLO
00DF 0335      rrf      ACCcHI
00E0 0334      rrf      ACCcLO
;
;
IF SIGNED
btffs sign,MSB
retlw 0
comf ACCcLO      ; negate ACCa ( -ACCa -> ACCa )
incf ACCcLO
btfs STATUS,Z_bit
decf ACCcHI
comf ACCcHI
btfs STATUS,Z_bit
neg_B comf ACCbLO      ; negate ACCb
incf ACCbLO
btfs STATUS,Z_bit
decf ACCbHI
comf ACCbHI
retlw 0
ELSE
retlw 0
ENDIF
00E1 0800

```

PIC16C5X Math Routines

```

;
;*****
;
00E2 0C10      setup    movlw    .16           ; for 16 shifts
00E3 0038      movwf   temp
00E4 0213      movf    ACCbHI,w          ;move ACCb to ACCd
00E5 0037      movwf   ACCdHI
00E6 0212      movf    ACCbLO,w
00E7 0036      movwf   ACCdLO
00E8 0073      clrf   ACCbHI
00E9 0072      clrf   ACCbLO
00EA 0800      retlw   0
;
;*****
;
00EB 0270      neg_A    comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
00EC 02B0      incf    ACCaLO
00ED 0643      btfsc   STATUS,Z_bit
00EE 00F1      decf    ACCaHI
00EF 0271      comf    ACCaHI
00F0 0800      retlw   0
;
;*****
; Assemble this section only if Signed Arithmetic Needed
;
        IF      SIGNED
;
S_SIGN  movf    ACCaHI,W
        xorwf   ACCbHI,W
        movwf   sign
        btfss  ACCbHI,MSB      ; if MSB set go & negate ACCb
        goto   chek_A
;
        comf    ACCbLO          ; negate ACCb
        incf    ACCbLO
        btfsc  STATUS,Z_bit
        decf    ACCbHI
        comf    ACCbHI
;
chek_A  btfss  ACCaHI,MSB      ; if MSB set go & negate ACCa
        retlw   0
        goto   neg_A
;
        ENDIF
;
;*****
```

PIC16C5X Math Routines

```
*****
;
;                               Test Program
;*****
;   Load constant values to ACCa & ACCb for testing
;
00F1 0C01   loadAB  movlw  1
00F2 0031           movwf  ACCaHI
00F3 0CFF           movlw  0FF          ; loads ACCa = 01FF
00F4 0030           movwf  ACCaLO
;
00F5 0C7F           movlw  07F
00F6 0033           movwf  ACCbHI
00F7 0CFF           movlw  0FF          ; loads ACCb = 7FFF
00F8 0032           movwf  ACCbLO
00F9 0800           retlw  0
;
00FA 0000   main   nop
;
00FB 09F1           call   loadAB          ; result of multiplying ACCb*ACCa-
00FC 0900           call   D_mpyF          ; Here (ACCb,ACCc) = 00FF 7E01
;
00FD 0AFD   self   goto   self
;
;                               org   PIC54
01FF 0AFA           goto   main
;                               END
*****
```

```
Errors   : 0
Warnings : 0
```

PIC16C5X Math Routines

APPENDIX E: DOUBLE PRECISION DIVISION LISTING (LOOPED)

MPASM B0.54

PAGE 1

```

;*****
;                               Double Precision Division
;
;                               ( Optimized for Code Size : Looped Code )
;
;*****
;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
;                               Remainder in ACCc (16 bits)
;   (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
;   (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
;   (c) CALL D_div
;   (d) The 16 bit result is in location ACCbHI & ACCbLO
;   (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;
; Performance :
;   Program Memory :      037
;   Clock Cycles   :      310
;
; NOTE :
;   The performance specs are for Unsigned arithmetic ( i.e.,
;   with "SIGNED equ FALSE ").
;*****
;
; LIST      P=16C54
0010      ACCaLO equ    10
0011      ACCaHI equ    11
0012      ACCbLO equ    12
0013      ACCbHI equ    13
0014      ACCcLO equ    14
0015      ACCcHI equ    15
0016      ACCdLO equ    16
0017      ACCdHI equ    17
0018      temp  equ    18
0019      sign  equ    19
;
;   include "mpreg.h"
;***** PIC16C5X Header *****
01FF      PIC54  equ    1FFH   ; Define Reset Vectors
01FF      PIC55  equ    1FFH
03FF      PIC56  equ    3FFH
07FF      PIC57  equ    7FFH
;
0001      RTCC   equ    1h
0002      PC     equ    2h
0003      STATUS equ    3h   ; F3 Reg is STATUS Reg.
0004      FSR    equ    4h
;
0005      Port_A equ    5h
0006      Port_B equ    6h   ; I/O Port Assignments
0007      Port_C equ    7h
;*****

```


PIC16C5X Math Routines

MPASM B0.54

PAGE 3

```

;
;
0000 CARRY equ 0h ; STATUS REG. Bits
0000 C equ 0h ; Carry Bit is Bit.0 of F3
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h
;
0001 Same equ 1h
;
0000 LSB equ 0h
0007 MSB equ 7h
;
0001 TRUE equ 1h
0001 YES equ 1h
0000 FALSE equ 0h
0000 NO equ 0h
;
;*****
;
; org 0
;*****
0000 SIGNED equ FALSE ; Set This To 'TRUE' if the routines
; ; for Multiplication & Division needs
; ; to be assembled as Signed Integer
; ; Routines. If 'FALSE' the above two
; ; routines ( D_mpy & D_div ) use
; ; unsigned arithmetic.
;*****
; Double Precision Divide ( 16/16 -> 16 )
;
; ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
; with Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
(ACCcHI,ACCcLO).
;
; NOTE : Before calling this routine, the user should make sure that
; the Numerator(ACCb) is greater than Denominator(ACCa). If
; the case is not true, the user should scale either Numerator
; or Denominator or both such that Numerator is greater than
; the Denominator.
;
;
; D_divs
;
; IF SIGNED
CALL S_SIGN
ENDIF
;
0000 091C call setup
0001 0075 clrf ACCcHI
0002 0074 clrf ACCcLO
0003 0403 dloop bcf STATUS,CARRY
0004 0376 rlf ACCdLO
0005 0377 rlf ACCdHI
0006 0374 rlf ACCcLO
0007 0375 rlf ACCcHI

```



PIC16C5X Math Routines

```

0008 0211      movf   ACCaHI,w
0009 0095      subwfb ACCcHI,w      ;check if a>c
000A 0743      btfsz  STATUS,Z_bit
000B 0A0E      goto   nochk
000C 0210      movf   ACCaLO,w
000D 0094      subwfb ACCcLO,w      ;if msb equal then check lsb
000E 0703      nochk  btfsz  STATUS,CARRY  ;carry set if c>a
000F 0A17      goto   nogo
0010 0210      movf   ACCaLO,w      ;c-a into c
0011 00B4      subwfb ACCcLO
0012 0703      btfsz  STATUS,CARRY
0013 00F5      decf   ACCcHI
0014 0211      movf   ACCaHI,w
0015 00B5      subwfb ACCcHI
0016 0503      bsf    STATUS,CARRY  ;shift a 1 into b (result)
0017 0372      nogo  rlf    ACCbLO
0018 0373      rlf    ACCbHI
0019 02F8      decfsz temp      ;loop untill all bits checked
001A 0A03      goto   dloop
;
; IF SIGNED
; btfsz sign,MSB      ; check sign if negative
; retlw 0
; goto neg_B          ; negate ACCa ( -ACCa -> ACCa )
ELSE
; retlw 0
ENDIF
001B 0800
;
;*****
;
001C 0C10      setup  movlw  .16      ; for 16 shifts
001D 0038      movwfb temp
001E 0213      movf   ACCbHI,w      ;move ACCb to ACCd
001F 0037      movwfb ACCdHI
0020 0212      movf   ACCbLO,w
0021 0036      movwfb ACCdLO
0022 0073      clrf  ACCbHI
0023 0072      clrf  ACCbLO
0024 0800      retlw 0
;
;*****
;
0025 0270      neg_A  comf   ACCaLO      ; negate ACCa ( -ACCa -> ACCa )
0026 02B0      incf   ACCaLO
0027 0643      btfsz  STATUS,Z_bit
0028 00F1      decf   ACCaHI
0029 0271      comf   ACCaHI
002A 0800      retlw 0
;
;*****
; Assemble this section only if Signed Arithmetic Needed
;
; IF SIGNED
;
; S_SIGN movf   ACCaHI,W
; xorwfb ACCbHI,W
; movwfb sign
; btfsz  ACCbHI,MSB      ; if MSB set go & negate ACCb
; goto  chek_A
;
; comf   ACCbLO          ; negate ACCb
; incf   ACCbLO
; btfsz  STATUS,Z_bit
; decf   ACCbHI
; comf   ACCbHI
;
;

```

PIC16C5X Math Routines

```

      chek_A  btfss  ACCaHI,MSB      ; if MSB set go & negate ACCa
              retlw  0
              goto  neg_A
;
      ENDIF
;
;*****
;                               Test Program
;*****
;   Load constant values to ACCa & ACCb for testing
;
002B 0C01   main  movlw  1
002C 0031           movwf  ACCaHI
002D 0CFF           movlw  0FF      ; loads ACCa = 01FF
002E 0030           movwf  ACCaLO
;
002F 0C7F           movlw  07F
0030 0033           movwf  ACCbHI
0031 0CFF           movlw  0FF      ; loads ACCb = 7FFF
0032 0032           movwf  ACCbLO
;
0033 0900           call  D_divS     ; remainder in ACCc. Here ACCb =0040 &
ACCC=003F
;
0034 0A34   self  goto  self
;
              org   PIC54
01FF 0A2B           goto  main
              END
;*****

Errors   :    0
Warnings :    0
```

PIC16C5X Math Routines

APPENDIX F: DOUBLE PRECISION DIVISION LISTING (FAST)

MPASM B0.54

PAGE 1

```
;*****  
; Double Precision Division  
;  
; ( Optimized for Speed : straight Line Code )  
;  
;*****  
; Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with  
; Remainder in ACCc (16 bits)  
; (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )  
; (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )  
; (c) CALL D_div  
; (d) The 16 bit result is in location ACCbHI & ACCbLO  
; (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO  
;  
; Performance :  
; Program Memory : 370  
; Clock Cycles : 263  
;  
; NOTE :  
; The performance specs are for Unsigned arithmetic ( i.e.,  
; with "SIGNED equ FALSE ").  
;*****  
;  
LIST P=16C54  
0010 ACCaLO equ 10  
0011 ACCaHI equ 11  
0012 ACCbLO equ 12  
0013 ACCbHI equ 13  
0014 ACCcLO equ 14  
0015 ACCcHI equ 15  
0016 ACCdLO equ 16  
0017 ACCdHI equ 17  
0018 temp equ 18  
0019 sign equ 19  
;  
include "picreg.h"  
;***** PIC16C5X Header *****  
01FF PIC54 equ 1FFH ; Define Reset Vectors  
01FF PIC55 equ 1FFH  
03FF PIC56 equ 3FFH  
07FF PIC57 equ 7FFH  
;  
0001 RTCC equ 1h  
0002 PC equ 2h  
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.  
0004 FSR equ 4h  
;  
0005 Port_A equ 5h  
0006 Port_B equ 6h ; I/O Port Assignments  
0007 Port_C equ 7h  
;  
;*****  
;  
; ; STATUS REG. Bits  
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3  
0000 C equ 0h  
0001 DCARRY equ 1h  
0001 DC equ 1h
```

PIC16C5X Math Routines

```

0002      Z_bit   equ    2h      ; Bit 2 of F3 is Zero Bit
0002      Z       equ    2h
0003      P_DOWN equ    3h
0003      PD      equ    3h
0004      T_OUT   equ    4h
0004      TO      equ    4h
0005      PA0     equ    5h
0006      PA1     equ    6h
0007      PA2     equ    7h
;
0001      Same    equ    1h
;
0000      LSB     equ    0h
0007      MSB     equ    7h
;
0001      TRUE    equ    1h
0001      YES     equ    1h
0000      FALSE   equ    0h
0000      NO      equ    0h
;
;*****
;
;          org    0
;*****
0000      SIGNED  equ    FALSE    ; Set This To 'TRUE' if the routines
;                               ; for Multiplication & Division needs
;                               ; to be assembled as Signed Integer
;                               ; Routines. If 'FALSE' the above two
;                               ; routines ( D_mpy & D_div ) use
;                               ; unsigned arithmetic.
;*****
;          division macro
;
divMac    MACRO
LOCAL    NOCHK
LOCAL    NOGO
;
    bcf    STATUS,CARRY
    rlf   ACCdLO
    rlf   ACCdHI
    rlf   ACCcLO
    rlf   ACCcHI
    movf  ACCaHI,w
    subwf ACCcHI,w      ;check if a>c
    btfss STATUS,Z_bit
    goto  NOCHK
    movf  ACCaLO,w
    subwf ACCcLO,w      ;if msb equal then check lsb
NOCHK    btfss STATUS,CARRY ;carry set if c>a
    goto  NOGO
    movf  ACCaLO,w      ;c-a into c
    subwf ACCcLO
    btfss STATUS,CARRY
    decf  ACCcHI
    movf  ACCaHI,w
    subwf ACCcHI
    bsf   STATUS,CARRY    ;shift a 1 into b (result)
NOGO     rlf   ACCbLO
         rlf   ACCbHI
;
;          ENDM
;
;*****
;          Double Precision Divide ( 16/16 -> 16 )
;
;          ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
;          with Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
;

```



PIC16C5X Math Routines

```

; NOTE : Before calling this routine, the user should make sure that
; the Numerator(ACCb) is greater than Denominator(ACCa). If
; the case is not true, the user should scale either Numerator
; or Denominator or both such that Numerator is greater than
; the Denominator.
;
;
0000 0C10      setup    movlw    .16            ; for 16 shifts
0001 0038      movwf    temp
0002 0213      movf     ACCbHI,w           ;move ACCb to ACCd
0003 0037      movwf    ACCdHI
0004 0212      movf     ACCbLO,w
0005 0036      movwf    ACCdLO
0006 0073      clrf     ACCbHI
0007 0072      clrf     ACCbLO
0008 0800      retlw    0

;
;*****
;
0009 0270      neg_A    comf     ACCaLO      ; negate ACCa ( -ACCa -> ACCa )
000A 02B0      incf     ACCaLO
000B 0643      btfsc   STATUS,Z_bit
000C 00F1      decf     ACCaHI
000D 0271      comf     ACCaHI
000E 0800      retlw    0

;
;*****
;
; D_divF
;
; IF SIGNED
; CALL S_SIGN
; ENDF
;
000F 0900      call     setup
0010 0075      clrf     ACCcHI
0011 0074      clrf     ACCcLO

;
; use the mulMac macro 16 times
;
; divMac
; LOCAL NOCHK
; LOCAL NOGO
;
0012 0403      bcf     STATUS,CARRY
0013 0376      rlf     ACCdLO
0014 0377      rlf     ACCdHI
0015 0374      rlf     ACCcLO
0016 0375      rlf     ACCcHI
0017 0211      movf     ACCaHI,w
0018 0095      subwf   ACCcHI,w           ;check if a>c
0019 0743      btfss   STATUS,Z_bit
001A 0A1D      goto     NOCHK
001B 0210      movf     ACCaLO,w
001C 0094      subwf   ACCcLO,w           ;if msb equal then check lsb
001D 0703      NOCHK  btfss   STATUS,CARRY ;carry set if c>a
001E 0A26      goto     NOGO
001F 0210      movf     ACCaLO,w           ;c-a into c
0020 00B4      subwf   ACCcLO
0021 0703      btfss   STATUS,CARRY
0022 00F5      decf     ACCcHI
0023 0211      movf     ACCaHI,w
0024 00B5      subwf   ACCcHI
0025 0503      bsf     STATUS,CARRY      ;shift a 1 into b (result)

```

PIC16C5X Math Routines

```

0026 0372      NOGO    rlf    ACCbLO
0027 0373      NOGO    rlf    ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
0028 0403      bcf     STATUS,CARRY
0029 0376      rlf     ACCdLO
002A 0377      rlf     ACCdHI
002B 0374      rlf     ACCcLO
002C 0375      rlf     ACCcHI
002D 0211      movf   ACCaHI,w
002E 0095      subwf  ACCcHI,w      ;check if a>c
002F 0743      btfss  STATUS,Z_bit
0030 0A33      goto   NOCHK
0031 0210      movf   ACCaLO,w
0032 0094      subwf  ACCcLO,w      ;if msb equal then check lsb
0033 0703      NOCHK  btfss  STATUS,CARRY ;carry set if c>a
0034 0A3C      goto   NOGO
0035 0210      movf   ACCaLO,w      ;c-a into c
0036 00B4      subwf  ACCcLO
0037 0703      btfss  STATUS,CARRY
0038 00F5      decf   ACCcHI
0039 0211      movf   ACCaHI,w
003A 00B5      subwf  ACCcHI
003B 0503      bsf    STATUS,CARRY      ;shift a 1 into b (result)
003C 0372      NOGO    rlf    ACCbLO
003D 0373      NOGO    rlf    ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
003E 0403      bcf     STATUS,CARRY
003F 0376      rlf     ACCdLO
0040 0377      rlf     ACCdHI
0041 0374      rlf     ACCcLO
0042 0375      rlf     ACCcHI
0043 0211      movf   ACCaHI,w
0044 0095      subwf  ACCcHI,w      ;check if a>c
0045 0743      btfss  STATUS,Z_bit
0046 0A49      goto   NOCHK
0047 0210      movf   ACCaLO,w
0048 0094      subwf  ACCcLO,w      ;if msb equal then check lsb
0049 0703      NOCHK  btfss  STATUS,CARRY ;carry set if c>a
004A 0A52      goto   NOGO
004B 0210      movf   ACCaLO,w      ;c-a into c
004C 00B4      subwf  ACCcLO
004D 0703      btfss  STATUS,CARRY
004E 00F5      decf   ACCcHI
004F 0211      movf   ACCaHI,w
0050 00B5      subwf  ACCcHI
0051 0503      bsf    STATUS,CARRY      ;shift a 1 into b (result)
0052 0372      NOGO    rlf    ACCbLO
0053 0373      NOGO    rlf    ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
0054 0403      bcf     STATUS,CARRY
0055 0376      rlf     ACCdLO
0056 0377      rlf     ACCdHI
0057 0374      rlf     ACCcLO
0058 0375      rlf     ACCcHI

```

PIC16C5X Math Routines

```
0059 0211          movf   ACCaHI,w
005A 0095          subwf  ACCcHI,w      ;check if a>c
005B 0743          btfsz  STATUS,Z_bit
005C 0A5F          goto   NOCHK
005D 0210          movf   ACCaLO,w
005E 0094          subwf  ACCcLO,w      ;if msb equal then check lsb
005F 0703          NOCHK btfsz  STATUS,CARRY ;carry set if c>a
0060 0A68          goto   NOGO
0061 0210          movf   ACCaLO,w      ;c-a into c
0062 00B4          subwf  ACCcLO
0063 0703          btfsz  STATUS,CARRY
0064 00F5          decf  ACCcHI
0065 0211          movf   ACCaHI,w
0066 00B5          subwf  ACCcHI
0067 0503          bsf   STATUS,CARRY ;shift a 1 into b (result)
0068 0372          NOGO  rlf   ACCbLO
0069 0373          rlf   ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
006A 0403          bcf   STATUS,CARRY
006B 0376          rlf   ACCdLO
006C 0377          rlf   ACCdHI
006D 0374          rlf   ACCcLO
006E 0375          rlf   ACCcHI
006F 0211          movf   ACCaHI,w
0070 0095          subwf  ACCcHI,w      ;check if a>c
0071 0743          btfsz  STATUS,Z_bit
0072 0A75          goto   NOCHK
0073 0210          movf   ACCaLO,w
0074 0094          subwf  ACCcLO,w      ;if msb equal then check lsb
0075 0703          NOCHK btfsz  STATUS,CARRY ;carry set if c>a
0076 0A7E          goto   NOGO
0077 0210          movf   ACCaLO,w      ;c-a into c
0078 00B4          subwf  ACCcLO
0079 0703          btfsz  STATUS,CARRY
007A 00F5          decf  ACCcHI
007B 0211          movf   ACCaHI,w
007C 00B5          subwf  ACCcHI
007D 0503          bsf   STATUS,CARRY ;shift a 1 into b (result)
007E 0372          NOGO  rlf   ACCbLO
007F 0373          rlf   ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
0080 0403          bcf   STATUS,CARRY
0081 0376          rlf   ACCdLO
0082 0377          rlf   ACCdHI
0083 0374          rlf   ACCcLO
0084 0375          rlf   ACCcHI
0085 0211          movf   ACCaHI,w
0086 0095          subwf  ACCcHI,w      ;check if a>c
0087 0743          btfsz  STATUS,Z_bit
0088 0A8B          goto   NOCHK
0089 0210          movf   ACCaLO,w
008A 0094          subwf  ACCcLO,w      ;if msb equal then check lsb
008B 0703          NOCHK btfsz  STATUS,CARRY ;carry set if c>a
008C 0A94          goto   NOGO
008D 0210          movf   ACCaLO,w      ;c-a into c
008E 00B4          subwf  ACCcLO
008F 0703          btfsz  STATUS,CARRY
0090 00F5          decf  ACCcHI
0091 0211          movf   ACCaHI,w
```


PIC16C5X Math Routines

```

0092 00B5          subwf  ACCcHI
0093 0503          bsf    STATUS,CARRY    ;shift a 1 into b (result)
0094 0372          rlf    ACCbLO
0095 0373          rlf    ACCbHI
;
divMac
LOCAL NOCHK
LOCAL NOGO
;
0096 0403          bcf    STATUS,CARRY
0097 0376          rlf    ACCdLO
0098 0377          rlf    ACCdHI
0099 0374          rlf    ACCcLO
009A 0375          rlf    ACCcHI
009B 0211          movf  ACCaHI,w
009C 0095          subwf  ACCcHI,w    ;check if a>c
009D 0743          btfs  STATUS,Z_bit
009E 0AA1          goto  NOCHK
009F 0210          movf  ACCaLO,w
00A0 0094          subwf  ACCcLO,w    ;if msb equal then check lsb
00A1 0703          NOCHK btfs  STATUS,CARRY ;carry set if c>a
00A2 0AAA          goto  NOGO
00A3 0210          movf  ACCaLO,w    ;c-a into c
00A4 00B4          subwf  ACCcLO
00A5 0703          btfs  STATUS,CARRY
00A6 00F5          decf  ACCcHI
00A7 0211          movf  ACCaHI,w
00A8 00B5          subwf  ACCcHI
00A9 0503          bsf    STATUS,CARRY    ;shift a 1 into b (result)
00AA 0372          NOGO  rlf    ACCbLO
00AB 0373          rlf    ACCbHI
;
divMac
LOCAL NOCHK
LOCAL NOGO
;
00AC 0403          bcf    STATUS,CARRY
00AD 0376          rlf    ACCdLO
00AE 0377          rlf    ACCdHI
00AF 0374          rlf    ACCcLO
00B0 0375          rlf    ACCcHI
00B1 0211          movf  ACCaHI,w
00B2 0095          subwf  ACCcHI,w    ;check if a>c
00B3 0743          btfs  STATUS,Z_bit
00B4 0AB7          goto  NOCHK
00B5 0210          movf  ACCaLO,w
00B6 0094          subwf  ACCcLO,w    ;if msb equal then check lsb
00B7 0703          NOCHK btfs  STATUS,CARRY ;carry set if c>a
00B8 0AC0          goto  NOGO
00B9 0210          movf  ACCaLO,w    ;c-a into c
00BA 00B4          subwf  ACCcLO
00BB 0703          btfs  STATUS,CARRY
00BC 00F5          decf  ACCcHI
00BD 0211          movf  ACCaHI,w
00BE 00B5          subwf  ACCcHI
00BF 0503          bsf    STATUS,CARRY    ;shift a 1 into b (result)
00C0 0372          NOGO  rlf    ACCbLO
00C1 0373          rlf    ACCbHI
;
divMac
LOCAL NOCHK
LOCAL NOGO
;
00C2 0403          bcf    STATUS,CARRY
00C3 0376          rlf    ACCdLO
00C4 0377          rlf    ACCdHI

```


PIC16C5X Math Routines

```

00FE 00F5          decf  ACCcHI
00FF 0211          movf  ACCaHI,w
0100 00B5          subwf ACCcHI
0101 0503          bsf   STATUS,CARRY      ;shift a 1 into b (result)
0102 0372          NOGO  rlf   ACCbLO
0103 0373          rlf   ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
0104 0403          bcf   STATUS,CARRY
0105 0376          rlf   ACCdLO
0106 0377          rlf   ACCdHI
0107 0374          rlf   ACCcLO
0108 0375          rlf   ACCcHI
0109 0211          movf  ACCaHI,w
010A 0095          subwf ACCcHI,w      ;check if a>c
010B 0743          btfs  STATUS,Z_bit
010C 0B0F          goto  NOCHK
010D 0210          movf  ACCaLO,w
010E 0094          subwf ACCcLO,w      ;if msb equal then check lsb
010F 0703          NOCHK btfs  STATUS,CARRY      ;carry set if c>a
0110 0B18          goto  NOGO
0111 0210          movf  ACCaLO,w      ;c-a into c
0112 00B4          subwf ACCcLO
0113 0703          btfs  STATUS,CARRY
0114 00F5          decf  ACCcHI
0115 0211          movf  ACCaHI,w
0116 00B5          subwf ACCcHI
0117 0503          bsf   STATUS,CARRY      ;shift a 1 into b (result)
0118 0372          NOGO  rlf   ACCbLO
0119 0373          rlf   ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
011A 0403          bcf   STATUS,CARRY
011B 0376          rlf   ACCdLO
011C 0377          rlf   ACCdHI
011D 0374          rlf   ACCcLO
011E 0375          rlf   ACCcHI
011F 0211          movf  ACCaHI,w
0120 0095          subwf ACCcHI,w      ;check if a>c
0121 0743          btfs  STATUS,Z_bit
0122 0B25          goto  NOCHK
0123 0210          movf  ACCaLO,w
0124 0094          subwf ACCcLO,w      ;if msb equal then check lsb
0125 0703          NOCHK btfs  STATUS,CARRY      ;carry set if c>a
0126 0B2E          goto  NOGO
0127 0210          movf  ACCaLO,w      ;c-a into c
0128 00B4          subwf ACCcLO
0129 0703          btfs  STATUS,CARRY
012A 00F5          decf  ACCcHI
012B 0211          movf  ACCaHI,w
012C 00B5          subwf ACCcHI
012D 0503          bsf   STATUS,CARRY      ;shift a 1 into b (result)
012E 0372          NOGO  rlf   ACCbLO
012F 0373          rlf   ACCbHI
;
      divMac
      LOCAL  NOCHK
      LOCAL  NOGO
;
0130 0403          bcf   STATUS,CARRY

```


PIC16C5X Math Routines

```

016A 00B4          subwf  ACCcLO
016B 0703          btfss STATUS,CARRY
016C 00F5          decf  ACCcHI
016D 0211          movf  ACCaHI,w
016E 00B5          subwf  ACCcHI
016F 0503          bsf   STATUS,CARRY      ;shift a 1 into b (result)
0170 0372          rlf   ACCbLO
0171 0373          rlf   ACCbHI
;
;
;
IF SIGNED
btfss sign,MSB      ; check sign if negative
retlw 0
goto neg_B          ; negate ACCa ( -ACCa -> ACCa )
ELSE
retlw 0
ENDIF

0172 0800
;
;*****
; Assemble this section only if Signed Arithmetic Needed
;
;
IF SIGNED
;
S_SIGN movf  ACCaHI,W
xorwf  ACCbHI,W
movwf  sign
btfss ACCbHI,MSB   ; if MSB set go & negate ACCb
goto  chek_A

;
comf  ACCbLO        ; negate ACCb
incf  ACCbLO
btfsc STATUS,Z_bit
decf  ACCbHI
comf  ACCbHI

;
chek_A btfss ACCaHI,MSB ; if MSB set go & negate ACCa
retlw 0
goto  neg_A

;
ENDIF

;
;*****
; Test Program
;*****
; Load constant values to ACCa & ACCb for testing
;
main   movlw  1
0174 0031      movwf  ACCaHI
0175 0CFF      movlw  0FF      ; loads ACCa = 01FF
0176 0030      movwf  ACCaLO
;
movlw  07F
0178 0033      movwf  ACCbHI
0179 0CFF      movlw  0FF      ; loads ACCb = 7FFF
017A 0032      movwf  ACCbLO

017B 090F      call  D_divF      ; remainder in ACCc. Here ACCb =0040 &
ACCc=003F

;
017C 0B7C      self   goto  self
;
org  PIC54
LIST  p=16c54
01FF 0B73      goto  main
END
;*****

```

```

Errors   :    0
Warnings :    0

```



PIC16C5X Math Routines

APPENDIX G: DOUBLE PRECISION ADDITION AND SUBTRACTION LISTING

MPASM B0.54

PAGE 1

```
;*****  
;                               Double Precision Addition & Subtraction  
;*****  
; Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)  
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )  
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )  
; (c) CALL D_add  
; (d) The result is in location ACCbLO & ACCbHI ( 16 bits )  
;  
; Performance :  
; Program Memory :      07  
; Clock Cycles   :      08  
;*****  
; Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)  
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )  
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )  
; (c) CALL D_sub  
; (d) The result is in location ACCbLO & ACCbHI ( 16 bits )  
;  
; Performance :  
; Program Memory :      14  
; Clock Cycles   :      17  
;*****  
;                               LIST P=16C54  
0010 ACCaLO equ 10  
0011 ACCaHI equ 11  
0012 ACCbLO equ 12  
0013 ACCbHI equ 13  
;  
; include "mpreg.h"  
;***** PIC16C5X Header *****  
01FF PIC54 equ 1FFH ; Define Reset Vectors  
01FF PIC55 equ 1FFH  
03FF PIC56 equ 3FFH  
07FF PIC57 equ 7FFH  
;  
0001 RTCC equ 1h  
0002 PC equ 2h  
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.  
0004 FSR equ 4h  
;  
0005 Port_A equ 5h  
0006 Port_B equ 6h ; I/O Port Assignments  
0007 Port_C equ 7h  
;  
;*****  
;                               ; STATUS REG. Bits  
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3  
0000 C equ 0h  
0001 DCARRY equ 1h  
0001 DC equ 1h  
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit  
0002 Z equ 2h  
0003 P_DOWN equ 3h  
0003 PD equ 3h  
0004 T_OUT equ 4h  
0004 TO equ 4h  
0005 PA0 equ 5h  
0006 PA1 equ 6h  
0007 PA2 equ 7h  
;  
;
```

PIC16C5X Math Routines

```

0001      Same    equ    1h
;
0000      LSB     equ    0h
0007      MSB     equ    7h
;
0001      TRUE    equ    1h
0001      YES     equ    1h
0000      FALSE   equ    0h
0000      NO      equ    0h
;
;*****
;
;          org      0
;*****
;          Double Precision Subtraction ( ACCb - ACCa -> ACCb )
;
0000 0908 D_sub    call    neg_A      ; At first negate ACCa; Then add
;
;          Double Precision Addition ( ACCb + ACCa -> ACCb )
;
0001 0210 D_add    movf    ACCaLO,w
0002 01F2      addwf   ACCbLO      ;add lsb
0003 0603      btfsc  STATUS,CARRY ;add in carry
0004 02B3      incf    ACCbHI
0005 0211      movf    ACCaHI,w
0006 01F3      addwf   ACCbHI      ;add msb
0007 0800      retlw   0
;
;
0008 0270 neg_A    comf    ACCaLO      ; negate ACCa ( -ACCa -> ACCa )
0009 02B0      incf    ACCaLO
000A 0643      btfsc  STATUS,Z_bit
000B 00F1      decf    ACCaHI
000C 0271      comf    ACCaHI
000D 0800      retlw   0
;
;*****
;          Test Program
;*****
;          Load constant values to ACCa & ACCb for testing
;
000E 0C01 loadAB  movlw   1
000F 0031      movwf  ACCaHI
0010 0CFF      movlw   0FF      ; loads ACCa = 01FF
0011 0030      movwf  ACCaLO
;
0012 0C7F      movlw   07F
0013 0033      movwf  ACCbHI
0014 0CFF      movlw   0FF      ; loads ACCb = 7FFF
0015 0032      movwf  ACCbLO
0016 0800      retlw   0
;
0017 0000 main   nop
;
0018 090E      call    loadAB      ; result of adding ACCb+ACCa->ACCb
0019 0901      call    D_add      ; Here Accb = 81FE
;
001A 090E      call    loadAB      ; result of subtracting ACCb - ACCa->ACCb
001B 0900      call    D_sub      ; Here Accb = 7E00
;
001C 0A1C self   goto    self
;
;          org      PIC54
01FF 0A17      goto    main
;          END
;*****
Errors   :    0
Warnings :    0

```

PIC16C5X Math Routines

APPENDIX H: FLOATING POINT ROUTINES LISTING

MPASM B0.54

PAGE 1

```
*****
; Binary Floating Point Addition, Subtraction, and Multiplication
; routines.
;
;*****
;
; Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
; the 8 bit exponent in EXPa.
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
; the 8 bit exponent in EXPb.
; (c) CALL F_add
; (d) The result is in location ACCbLO & ACCbHI ( 16 bits ) with
; the 8 bit exponent in EXPb
;
; Program Memory :      55 locations
;
;*****
;
; Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
; the 8 bit exponent in EXPa .
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
; the 8 bit exponent in EXPb .
; (c) CALL F_sub
; (d) The result is in location ACCbLO & ACCbHI ( 16 bits ) with
; the 8 bit exponent in EXPb.
;
; Program Memory :      61 locations
;
;*****
;
; Multiplication :
; ACCb(16 bits)EXP(b) * ACCa(16 bits)EXPa -> ACCb(16 bits)EXPb
; where, EXP(x) represents an 8 bit exponent.
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
; an 8 bit exponent in location EXPa
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
; an 8 bit exponent in location EXPb
; (c) CALL F_mpy
; (d) The 16 bit result overwrites ACCb(ACCbLO & ACCbHI). The exponent
; is stored in EXPb and the results are normalized.
;
; NOTE : If one needs to get a 32 bit product( & an 8 bit exponent ),
; re assemble the program after changing the line " Model6 equ TRUE"
; to " Model6 equ FALSE ".
; If this option is chosen, then the 32 bit result is returned in
; ( ACCbHI, ACCbLO, ACCcHI, ACCcLO ) and the 8 bit exponent in EXPb.
; This method ( with " Model6 equ FALSE " ) is NOT Recommended.
;
; Program Memory :      102 locations
;
;*****
;
; LIST      n=0,p=16C54
0010      ACCaLO equ 10
0011      ACCaHI equ 11
0012      EXPa  equ 12
0013      ACCbLO equ 13
0014      ACCbHI equ 14
0015      EXPb  equ 15
```


PIC16C5X Math Routines

```

0016          ACCcLO equ    16
0017          ACCcHI equ    17
0018          ACCdLO equ    18
0019          ACCdHI equ    19
001A          temp  equ    1A
001B          sign  equ    1B
;
include "mpreg.h"
;***** PIC16C5X Header *****
01FF          PIC54 equ    1FFH ; Define Reset Vectors
01FF          PIC55 equ    1FFH
03FF          PIC56 equ    3FFH
07FF          PIC57 equ    7FFH
;
0001          RTCC  equ    1
0002          PC    equ    2
0003          STATUS equ    3 ; F3 Reg is STATUS Reg.
0004          FSR   equ    4
;
0005          Port_A equ    5
0006          Port_B equ    6 ; I/O Port Assignments
0007          Port_C equ    7
;
;*****
;
; ; STATUS REG. Bits
0000          CARRY equ    0 ; Carry Bit is Bit.0 of F3
0000          C     equ    0
0001          DCARRY equ    1
0001          DC    equ    1
0002          Z_bit equ    2 ; Bit 2 of F3 is Zero Bit
0002          Z     equ    2
0003          P_DOWN equ    3
0003          PD    equ    3
0004          T_OUT equ    4
0004          TO    equ    4
0005          PA0   equ    5
0006          PA1   equ    6
0007          PA2   equ    7
;
0001          Same  equ    1
0000          W     equ    0
;
0000          LSB   equ    0
0007          MSB   equ    7
;
0001          TRUE  equ    1
0001          YES   equ    1
0000          FALSE equ    0
0000          NO    equ    0
;
;*****
;
0001          Model6 equ    TRUE ; Change this to FALSE if a 32 bit product
; ; is desired for F_mpy routine.
;
; org 0
;*****
; Floating Point Subtraction ( ACCb - ACCa -> ACCb )
;
0000 0968          F_sub call    neg_A ; At first negate ACCa; Then add
;
0001 0212          F_add movf    EXPa,w ; scale mantissas
0002 0095          subwf   EXPb,w ; find the greater exponent
0003 0643          btfsc   STATUS,Z_bit
0004 0A0E          goto    padd ; exponents are equal
0005 0603          btfsc   STATUS,CARRY ;

```

PIC16C5X Math Routines

```

0006 0986          call    F_swap          ; if A > B then swap ( A<->B )
0007 0212          movf    EXPa,w
0008 00B5          subwf   EXPb
0009 0920          scloop  call    shftSR
000A 03F5          incfsz  EXPb
000B 0A09          goto    scloop
000C 0212          movf    EXPa,w
000D 0035          movwf   EXPb
000E 0211          padd   movf    ACCaHI,w
000F 0114          iorwf   ACCbHI,w
0010 003B          movwf   sign
0011 0919          call    D_add          ; compute double precision integer add
0012 07FB          btfsz  sign,MSB
0013 07F4          btfsz  ACCbHI,MSB
0014 0800          retlw   0
0015 0403          bcf    STATUS,CARRY
0016 02B5          incf    EXPb
0017 0A23          goto    shftR
;
;*****
;          Double Precision Subtraction ( ACCb - ACCa -> ACCb )
;
0018 0968          D_sub  call    neg_A          ; At first negate ACCa; Then add
;
0019 0210          D_add  movf    ACCaLO,w          ; Addition ( ACCb + ACCa -> ACCb )
001A 01F3          addwfb ACCbLO          ;add lsb
001B 0603          btfsz  STATUS,CARRY      ;add in carry
001C 02B4          incf    ACCbHI
001D 0211          movf    ACCaHI,w
001E 01F4          addwfb ACCbHI          ;add msb
001F 0800          retlw   0
;*****
;
;
0020 0403          shftSR bcf    STATUS,CARRY
0021 06F4          btfsz  ACCbHI,MSB
0022 0503          bsf    STATUS,CARRY      ; set carry if < 0
0023 0334          shftR  rrf    ACCbHI
0024 0333          rrf    ACCbLO
0025 0800          retlw   0
;
0026 0403          shftSL bcf    STATUS,CARRY
;
;          if Model6
0027 0376          rlf    ACCcLO
0028 0377          rlf    ACCcHI
;          endif
;
0029 0373          rlf    ACCbLO
002A 0374          rlf    ACCbHI
002B 04F4          bcf    ACCbHI,MSB
002C 0603          btfsz  STATUS,CARRY
002D 05F4          bsf    ACCbHI,MSB
002E 0800          retlw   0
;
;*****
;          Binary Floating Point Multiplication :
;          ACCb(16 bits)EXP(b) * ACCa(16 bits)EXPa -> ACCb(16 bits)EXPb
;
F_mpy
002F 096E          call    S_SIGN
0030 095F          call    setup
0031 0403          mloop  bcf    STATUS,CARRY      ; clear carry bit          ??????????
0032 0339          rrf    ACCdHI          ;rotate d right
0033 0338          rrf    ACCdLO
0034 0603          btfsz  STATUS,CARRY      ;need to add?
0035 0919          call    D_add
0036 0334          rrf    ACCbHI
0037 0333          rrf    ACCbLO

```

PIC16C5X Math Routines

```

0038 0337          rrf     ACCcHI
0039 0336          rrf     ACCcLO
003A 02FA          decfsz temp          ;loop until all bits checked
003B 0A31          goto    mloop

;

003C 0212          movf    EXPa,w
003D 01F5          addwf   EXPb

;

IF     Model6
003E 0234          movf    ACCbHI
003F 0743          btfsz  STATUS,Z_bit
0040 0A51          goto    finup          ; if ACCbHI != 0
0041 0233          movf    ACCbLO
0042 0743          btfsz  STATUS,Z_bit
0043 0A4B          goto    Shft08          ; if ACCbLO != 0 && ACCbHI == 0

;

0044 0217          movf    ACCcHI,w
0045 0034          movwf   ACCbHI          ; if ACCb == 0, then move ACCc to ACCb
0046 0216          movf    ACCcLO,w
0047 0033          movwf   ACCbLO
0048 0C10          movlw   .16
0049 01F5          addwf   EXPb
004A 0A51          goto    finup

;
Shft08
004B 0213          movf    ACCbLO,w
004C 0034          movwf   ACCbHI
004D 0217          movf    ACCcHI,w
004E 0033          movwf   ACCbLO
004F 0C08          movlw   .8
0050 01F5          addwf   EXPb

;
ENDIF          ; matching endif for IF Model6

;
finup
0051 07FB          btfsz  sign,MSB
0052 0A7B          goto    F_norm

;

0053 00F6          decf    ACCcLO          ; negate ACCc
0054 0276          comf    ACCcLO
0055 0643          btfsz  STATUS,Z_bit
0056 00F7          decf    ACCcHI
0057 0277          comf    ACCcHI
0058 0643          btfsz  STATUS,Z_bit

;
neg_B
0059 00F3          decf    ACCbLO          ; negate ACCb
005A 0273          comf    ACCbLO
005B 0643          btfsz  STATUS,Z_bit
005C 00F4          decf    ACCbHI
005D 0274          comf    ACCbHI

;

005E 0A7B          goto    F_norm

;
;*****
;
setup
005F 0C10          movlw   .16          ; for 16 shifts
0060 003A          movwf   temp
0061 0214          movf    ACCbHI,w          ;move ACCb to ACCd
0062 0039          movwf   ACCdHI
0063 0213          movf    ACCbLO,w
0064 0038          movwf   ACCdLO
0065 0074          clrf   ACCbHI
0066 0073          clrf   ACCbLO          ; clear ACCb ( ACCbLO & ACCbHI )
0067 0800          retlw  0

;
;*****
;
neg_A
0068 0270          comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
0069 02B0          incf   ACCaLO

```

PIC16C5X Math Routines

```

006A 0643          btfsc  STATUS,Z_bit
006B 00F1          decf   ACCaHI
006C 0271          comf   ACCaHI
006D 0800          retlw  0
;
;*****
;
006E 0211  S_SIGN  movf   ACCaHI,W
006F 0194          xorwf  ACCbHI,W
0070 003B          movwf  sign
0071 07F4          btfss ACCbHI,MSB      ; if MSB set go & negate ACCb
0072 0A78          goto   chek_A
;
0073 0273          comf   ACCbLO      ; negate ACCb
0074 02B3          incf   ACCbLO
0075 0643          btfsc  STATUS,Z_bit
0076 00F4          decf   ACCbHI
0077 0274          comf   ACCbHI
;
0078 07F1  chek_A  btfss  ACCaHI,MSB      ; if MSB set go & negate ACCa
0079 0800          retlw  0
007A 0A68          goto   neg_A
;
;*****
;      Normalize Routine
; Normalizes ACCb for use in floating point calculations.
; Call this routine as often as possible to minimize the loss
; of precision. This routine normalizes ACCb so that the
; mantissa is maximized and the exponent minimized.
;
;
;
007B 0234  E_norm  movf   ACCbHI
007C 0743          btfss  STATUS,Z_bit
007D 0A81          goto   C_norm
007E 0233          movf   ACCbLO
007F 0643          btfsc  STATUS,Z_bit
0080 0800          retlw  0
0081 06D4  C_norm  btfsc  ACCbHI,6
0082 0800          retlw  0
0083 0926          call  shftSL
0084 00F5          decf   EXPb
0085 0A81          goto   C_norm
;
;*****
; Swap ACCa & ACCb  [ (ACCa,EXPa) <-> (ACCb,EXPb) ]
;
;
0086 0211  E_swap  movf   ACCaHI,w
0087 003A          movwf  temp
0088 0214          movf   ACCbHI,w      ;ACCaHI <-> ACCbHI
0089 0031          movwf  ACCaHI
008A 021A          movf   temp,w
008B 0034          movwf  ACCbHI
;
008C 0210          movf   ACCaLO,w
008D 003A          movwf  temp
008E 0213          movf   ACCbLO,w      ;ACCaLO <-> ACCbLO
008F 0030          movwf  ACCaLO
0090 021A          movf   temp,w
0091 0033          movwf  ACCbLO
;
0092 0212          movf   EXPa,w
0093 003A          movwf  temp
0094 0215          movf   EXPb,w      ;EXPa <-> EXPb
0095 0032          movwf  EXPa
0096 021A          movf   temp,w
0097 0035          movwf  EXPb
;

```

PIC16C5X Math Routines

```
0098 0800                retlw  0
;
;*****
;                               Test Program
;*****
;   Load constant values to (ACCa, EXPa) & (ACCb, EXPb) for testing
;
0099 0C01    loadAB    movlw   1
009A 0031                movwf  ACCaHI
009B 0CFF                movlw   0FF          ; loads ACCa = 01FF EXP(4)
009C 0030                movwf  ACCaLO
009D 0C04                movlw   04
009E 0032                movwf  EXPa
;
009F 0C7F                movlw   07F
00A0 0034                movwf  ACCbHI
00A1 0CFF                movlw   0FF          ; loads ACCb = 7FFF EXP(6)
00A2 0033                movwf  ACCbLO
00A3 0C06                movlw   06
00A4 0035                movwf  EXPb
00A5 0800                retlw  0
;
; main    nop
;
00A7 0999                call   loadAB          ; result of adding  b)+ACCa(EXPa)-
00A8 0901                call   F_add           ; Here Accb = 403F, EXPb = 07
;
00A9 0999                call   loadAB          ; result of subtracting  b(EXPb)-
00AA 0900                call   F_sub           ; Here Accb = 7F7F, EXPb = 06
;
00AB 0999                call   loadAB          ; result of multiplying  b(EXPb) *
00AC 092F                call   F_mpy           ; Here ACCb = FF7E, EXPb = 12
;
00AD 0AAD                self   goto   self
;
;                               org   PIC54
;                               goto   main
01FF 0AA6                END
;*****
```

```
Errors   : 0
Warnings : 0
```



PIC16C5X Math Routines

APPENDIX I: BCD TO BINARY CONVERSION LISTING

MPASM B0.54

PAGE 1

```
;
;*****
;          BCD To Binary Conversion
;
;          This routine converts a 5 digit BCD number to a 16 bit binary
; number.
;          The input 5 digit BCD numbers are asumed to be in locations
; R0, R1 & R2 with R0 containing the MSD in its right most nibble.
;
;          The 16 bit binary number is output in registers H_byte & L_byte
; ( high byte & low byte repectively ).
;
;          The method used for conversion is :
;          input number X = abcde ( the 5 digit BCD number )
;          X = abcde = 10[10[10[10a+b]+c]+d]+e
;
; Performance :
;          Program Memory :      30
;          Clock Cycles   :      121
;*****
;
;          LIST      P=16C54
0010      H_byte equ 10
0011      L_byte equ 11
0012      R0     equ 12      ; RAM Assignments
0013      R1     equ 13
0014      R2     equ 14
;
0015      H_temp equ 15      ; temporary register
0016      L_temp equ 16      ; temporary register
;
;          INCLUDE      "mpreg.h"
;*****
;          PIC16C5X Header *****
01FF      PIC54 equ 1FFH    ; Define Reset Vectors
01FF      PIC55 equ 1FFH
03FF      PIC56 equ 3FFH
07FF      PIC57 equ 7FFH
;
0001      RTCC equ 1h
0002      PC   equ 2h
0003      STATUS equ 3h    ; F3 Reg is STATUS Reg.
0004      FSR  equ 4h
;
0005      Port_A equ 5h
0006      Port_B equ 6h    ; I/O Port Assignments
0007      Port_C equ 7h
;
;*****
;
;          ; STATUS REG. Bits
0000      CARRY equ 0h    ; Carry Bit is Bit.0 of F3
0000      C     equ 0h
0001      DCARRY equ 1h
0001      DC    equ 1h
0002      Z_bit equ 2h    ; Bit 2 of F3 is Zero Bit
0002      Z     equ 2h
0003      P_DOWN equ 3h
0003      PD    equ 3h
0004      T_OUT equ 4h
0004      TO    equ 4h
0005      PA0   equ 5h
0006      PA1   equ 6h
```

PIC16C5X Math Routines

```

0007          PA2      equ    7h
;
0001          Same    equ    1h
;
0000          LSB     equ    0h
0007          MSB     equ    7h
;
0001          TRUE    equ    1h
0001          YES     equ    1h
0000          FALSE   equ    0h
0000          NO      equ    0h
;
;*****
;
;
0000 0E0F    mpy10b  andlw   0F
0001 01F1    addwf   L_byte
0002 0603    btfsc  STATUS,CARRY
0003 02B0    incf   H_byte
0004 0403    mpy10a  bcf    STATUS,CARRY    ; multiply by 2
0005 0351    rlf    L_byte,w
0006 0036    movwf  L_temp
0007 0350    rlf    H_byte,w    ; (H_temp,L_temp) = 2*N
0008 0035    movwf  H_temp
;
0009 0403    bcf    STATUS,CARRY    ; multiply by 2
000A 0371    rlf    L_byte
000B 0370    rlf    H_byte
000C 0403    bcf    STATUS,CARRY    ; multiply by 2
000D 0371    rlf    L_byte
000E 0370    rlf    H_byte
000F 0403    bcf    STATUS,CARRY    ; multiply by 2
0010 0371    rlf    L_byte
0011 0370    rlf    H_byte    ; (H_byte,L_byte) = 8*N
;
0012 0216    movf   L_temp,w
0013 01F1    addwf  L_byte
0014 0603    btfsc  STATUS,CARRY
0015 02B0    incf   H_byte
0016 0215    movf   H_temp,w
0017 01F0    addwf  H_byte
0018 0800    retlw  0    ; (H_byte,L_byte) = 10*N
;
;
0019 0070    BCDtoB  clrf   H_byte
001A 0212    movf   R0,w
001B 0E0F    andlw  0F
001C 0031    movwf  L_byte
001D 0904    call  mpy10a    ; result = 10a+b
;
001E 0393    swapf  R1,w
001F 0900    call  mpy10b    ; result = 10[10a+b]
;
0020 0213    movf   R1,w
0021 0900    call  mpy10b    ; result = 10[10[10a+b]+c]
;
0022 0394    swapf  R2,w
0023 0900    call  mpy10b    ; result = 10[10[10[10a+b]+c]+d]
;
0024 0214    movf   R2,w
0025 0E0F    andlw  0F
0026 01F1    addwf  L_byte
0027 0603    btfsc  STATUS,CARRY
0028 02B0    incf   H_byte    ; result = 10[10[10[10a+b]+c]+d]+e
0029 0800    retlw  0    ; BCD to binary conversion done
;
;

```

PIC16C5X Math Routines

```

;*****
;                               Test Program
;*****
002A 0C06      main    movlw  06
002B 0032      movwf  R0      ; Set R0 = 06
002C 0C55      movlw  55
002D 0033      movwf  R1      ; Set R1 = 55
002E 0C35      movlw  35
002F 0034      movwf  R2      ; Set R2 = 35      ( R0, R1, R2 = 6,55,35 )
;
0030 0919      call   BCDtoB ; After conversion H_Byte = FF & L_Byte = FF
;
0031 0A31      self   goto   self
;
01FF 0A2A      org    1FF
               goto   main
;
               END

```

```
Errors   : 0
Warnings : 0

```


PIC16C5X Math Routines

APPENDIX J: BINARY (8-BIT) TO BCD LISTING

MPASM B0.54

PAGE 1

```
LIST      p=16c54,n=0

;
;*****
;
;          Binary To BCD Conversion Routine
;
;          This routine converts the 8 bit binary number in the W Register
; to a 2 digit BCD number.
;          The least significant digit is returned in location LSD and
; the most significant digit is returned in location MSD.
;
; Performance :
; Program Memory :      10
; Clock Cycles   :      81 (worst case when W = 63 Hex )
;                ( i.e max Decimal number 99 )
;*****
;
0010      LSD      equ    10
0011      MSD      equ    11

INCLUDE   "mpreg.h"
;***** PIC16C5X Header *****
01FF      PIC54     equ    1FFH   ; Define Reset Vectors
01FF      PIC55     equ    1FFH
03FF      PIC56     equ    3FFH
07FF      PIC57     equ    7FFH
;
0001      RTCC      equ    1
0002      PC        equ    2
0003      STATUS    equ    3      ; F3 Reg is STATUS Reg.
0004      FSR       equ    4
;
0005      Port_A    equ    5
0006      Port_B    equ    6      ; I/O Port Assignments
0007      Port_C    equ    7
;
;*****
;
;          ; STATUS REG. Bits
0000      CARRY     equ    0      ; Carry Bit is Bit.0 of F3
0000      C         equ    0
0001      DCARRY    equ    1
0001      DC        equ    1
0002      Z_bit     equ    2      ; Bit 2 of F3 is Zero Bit
0002      Z         equ    2
0003      P_DOWN    equ    3
0003      PD        equ    3
0004      T_OUT     equ    4
0004      TO        equ    4
0005      PA0       equ    5
0006      PA1       equ    6
0007      PA2       equ    7
;
0001      Same      equ    1
0000      W         equ    0
;
0000      LSB       equ    0
0007      MSB       equ    7
;
0001      TRUE      equ    1
```

PIC16C5X Math Routines

```
0001          YES      equ    1
0000          FALSE   equ    0
0000          NO      equ    0
;
;*****
;
;
0000 0071      BinBCD  clrf   MSD
0001 0030          movwf  LSD
0002 0C0A      gtenth  movlw  .10
0003 0090          subwf  LSD,W
0004 0703          BTFSS  STATUS,CARRY
0005 0A09          goto   over
0006 0030          movwf  LSD
0007 02B1          incf   MSD
0008 0A02          goto   gtenth
0009 0800      over   retlw  0
;*****
;
000A 0C63      main   movlw  63          ; W reg = 63 Hex
000B 0900          call   BinBCD       ; after conversion, MSD = 9 & LSD = 9
000C 0A0C      self   goto   self      ; ( 63 Hex = 99 Decimal )
;
;          org   1FF
01FF 0A0A      goto   main
;
END

Errors   :   0
Warnings :   0
```

PIC16C5X Math Routines

APPENDIX K: BINARY (16-BIT) TO BCD LISTING

MPASM B0.54

PAGE 1

```
;
;*****
; Binary To BCD Conversion Routine
; This routine converts a 16 Bit binary Number to a 5 Digit
; BCD Number. This routine is useful since PIC16C55 & PIC16C57
; have two 8 bit ports and one 4 bit port ( total of 5 BCD digits)
;
; The 16 bit binary number is input in locations H_byte and
; L_byte with the high byte in H_byte.
; The 5 digit BCD number is returned in R0, R1 and R2 with R0
; containing the MSD in its right most nibble.
;
; Performance :
; Program Memory : 35
; Clock Cycles : 885
;*****
;
; LIST P=16C54
0016 count equ 16
0017 temp equ 17
;
0010 H_byte equ 10
0011 L_byte equ 11
0012 R0 equ 12 ; RAM Assignments
0013 R1 equ 13
0014 R2 equ 14
;
include "mpreg.h"
;***** PIC16C5X Header *****
01FF PIC54 equ 1FFH ; Define Reset Vectors
01FF PIC55 equ 1FFH
03FF PIC56 equ 3FFH
07FF PIC57 equ 7FFH
;
0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h
;
0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h
;
;*****
;
; STATUS REG. Bits
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h
;
0001 Same equ 1h
;
0000 LSB equ 0h
```

PIC16C5X Math Routines

```

0007          MSB      equ      7h
;
0001          TRUE     equ      1h
0001          YES      equ      1h
0000          FALSE    equ      0h
0000          NO       equ      0h
;
;*****

;
0000 0403    B2_BCD   bcf      STATUS,0          ; clear the carry bit
0001 0C10          movlw   .16
0002 0036          movwf   count
0003 0072          clrfsz  R0
0004 0073          clrfsz  R1
0005 0074          clrfsz  R2
0006 0371    loop16  rlf     L_byte
0007 0370          rlf     H_byte
0008 0374          rlf     R2
0009 0373          rlf     R1
000A 0372          rlf     R0
;
000B 02F6          decfsz  count
000C 0A0E          goto    adjDEC
000D 0800          RETLW   0
;
000E 0C14    adjDEC  movlw   R2
000F 0024          movwf   FSR
0010 0918          call    adjBCD
;
0011 0C13          movlw   R1
0012 0024          movwf   FSR
0013 0918          call    adjBCD
;
0014 0C12          movlw   R0
0015 0024          movwf   FSR
0016 0918          call    adjBCD
;
0017 0A06          goto    loop16
;
0018 0C03    adjBCD  movlw   3
0019 01C0          addwf   0,W
001A 0037          movwf   temp
001B 0677          btfsz  temp,3          ; test if result > 7
001C 0020          movwf   0
001D 0C30          movlw   30
001E 01C0          addwf   0,W
001F 0037          movwf   temp
0020 06F7          btfsz  temp,7          ; test if result > 7
0021 0020          movwf   0          ; save as MSD
0022 0800          RETLW   0
;
;*****
;                               Test Program
;*****
0023 0CFF    main   movlw   0FF
0024 0030          movwf   H_byte
0025 0031          movwf   L_byte          ; The 16 bit binary number = FFFF
0026 0900          call    B2_BCD          ; After conversion the Decimal Number
;                               ; in R0,R1,R2 = 06,55,35
;
0027 0A27    self   goto    self
;
;                               org    1FF
01FF 0A23          goto    main
;
END

```

PIC16C5X Math Routines

APPENDIX L: UNSIGNED BCD ADDITION LISTING

MPASM B0.54

PAGE 1

```
;***** Unsigned BCD Addition *****
;
; This routine performs a 2 Digit Unsigned BCD Addition
; It is assumed that the two BCD numbers to be added are in
; locations Num_1 & Num_2. The result is the sum of Num_1+Num_2
; and is stored in location Num_2 and the overflow carry is returned
; in location Num_1
;
; Performance :
; Program Memory : 25
; Clock Cycles : 23 ( worst case )
;
; Rev 2.0 changed on 7/30/92.
;*****;
;
LIST P=16C54
0008 Num_1 equ 8 ; Overflow flow carry overwrites Num_1
0008 result equ 8
;
0009 Num_2 equ 9 ; Num_2 + Num_1 overwrites Num_2
0009 O_flow equ 9
;
; include "mpreg.h"
;***** PIC16C5X Header *****
01FF PIC54 equ 1FFh ; Define Reset Vectors
01FF PIC55 equ 1FFh
03FF PIC56 equ 3FFh
07FF PIC57 equ 7FFh
;
0001 RTCC equ 1h
0002 PC equ 2h
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.
0004 FSR equ 4h
;
0005 Port_A equ 5h
0006 Port_B equ 6h ; I/O Port Assignments
0007 Port_C equ 7h
;
;*****;
; ; STATUS REG. Bits
0000 CARRY equ 0h ; Carry Bit is Bit.0 of F3
0000 C equ 0h
0001 DCARRY equ 1h
0001 DC equ 1h
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit
0002 Z equ 2h
0003 P_DOWN equ 3h
0003 PD equ 3h
0004 T_OUT equ 4h
0004 TO equ 4h
0005 PA0 equ 5h
0006 PA1 equ 6h
0007 PA2 equ 7h
;
0001 Same equ 1h
;
0000 LSB equ 0h
0007 MSB equ 7h
;
0001 TRUE equ 1h
0001 YES equ 1h
0000 FALSE equ 0h
```

2

PIC16C5X Math Routines

```

0000          NO      equ      0h
;
;*****
;
;
0000 0208      BCDAdd  movf     Num_1,w
0001 01E9          addwf   Num_2          ; do binary addition
0002 0068          clrf    Num_1
0003 0368          rlf     Num_1
0004 0623          btfsc  STATUS,DC      ; Is DC = 0 ?
0005 0A0D          goto   adjust        ; adjust LSD
0006 0C06          movlw   6
0007 01E9          addwf   Num_2          ; Test for LSD > 9 ( by adding 6
0008 0603          btfsc  STATUS,CARRY
0009 02A8          incf    Num_1
000A 0723          btfss  STATUS,DC      ; & checking Digit Carry
000B 00A9          subwf   Num_2          ; LSD < 9 , so get back original value.
000C 0A0F          goto   over1
000D 0C06      adjust  movlw   6
000E 01E9          addwf   Num_2
000F 0C60      over1  movlw   60          ; add 6 to MSD
0010 01E9          addwf   Num_2
0011 0603          btfsc  STATUS,CARRY
0012 0A16          goto   over3
0013 0708          btfss  Num_1,0
0014 00A9          subwf   Num_2
0015 0800          RETLW   0
0016 0C01      over3  movlw   1
0017 0028          movwf   Num_1
0018 0800          RETLW   0
;
;*****
;
;          Test Program
;*****
0019 0C99      main   movlw   99
001A 0028          movwf   Num_1          ; Set Num_1 = 99 ( max BCD digit )
001B 0C99          movlw   99
001C 0029          movwf   Num_2          ; Set Num_2 = 99
;
001D 0900          call    BCDAdd        ; After addition, Num_2 = 98
;
;          ; and Num_1 = 01 ( 99+99 = 198 -> max number
)
;
001E 0A1E      self   goto   self
;
;          org     1FF
01FF 0A19          goto   main
;
END

```

```

Errors   : 0
Warnings : 0

```

PIC16C5X Math Routines

APPENDIX M: UNSIGNED BCD SUBTRACTION LISTING

MPASM B0.54

PAGE 1

```
;***** Unsigned BCD Subtraction *****  
;  
; This routine performs a 2 Digit Unsigned BCD Subtraction.  
; It is assumed that the two BCD numbers to be subtracted are in  
; locations Num_1 & Num_2. The result is the difference of Num_1 & Num_2  
; ( Num_2 - Num_1) and is stored in location Num_2 and the overflow carry  
; is returned in location Num_1.  
;  
; Performance :  
; Program Memory : 31  
; Clock Cycles : 21 ( worst case )  
;  
;*****  
;  
LIST P=16C54  
0008 Num_1 equ 8 ; Overflow flow carry overwrites Num_1  
0008 result equ 8  
;  
0009 Num_2 equ 9 ; Num_2 - Num_1 overwrites Num_2  
0009 O_flow equ 9  
;  
include "mpreg.h"  
;***** PIC16C5X Header *****  
01FF PIC54 equ 1FFh ; Define Reset Vectors  
01FF PIC55 equ 1FFh  
03FF PIC56 equ 3FFh  
07FF PIC57 equ 7FFh  
;  
0001 RTCC equ 1h  
0002 PC equ 2h  
0003 STATUS equ 3h ; F3 Reg is STATUS Reg.  
0004 FSR equ 4h  
;  
0005 Port_A equ 5h  
0006 Port_B equ 6h ; I/O Port Assignments  
0007 Port_C equ 7h  
;  
;*****  
;  
0000 CARRY equ 0h ; STATUS REG. Bits  
0000 C equ 0h ; Carry Bit is Bit.0 of F3  
0001 DCARRY equ 1h  
0001 DC equ 1h  
0002 Z_bit equ 2h ; Bit 2 of F3 is Zero Bit  
0002 Z equ 2h  
0003 P_DOWN equ 3h  
0003 PD equ 3h  
0004 T_OUT equ 4h  
0004 TO equ 4h  
0005 PA0 equ 5h  
0006 PA1 equ 6h  
0007 PA2 equ 7h  
;  
0001 Same equ 1h  
;  
0000 LSB equ 0h  
0007 MSB equ 7h  
;  
0001 TRUE equ 1h  
0001 YES equ 1h  
0000 FALSE equ 0h  
0000 NO equ 0h  
;  
;*****
```

PIC16C5X Math Routines

```

;
0000 0208      BCDSub  movf   Num_1,w
0001 00A9          subwf  Num_2
0002 0068          clrf   Num_1
0003 0368          rlf    Num_1
0004 0723          btfss  STATUS,DC
0005 0A0C          goto   adjst1
0006 0769          btfss  Num_2,3      ; Adjust LSD of Result
0007 0A0E          goto   Over_1
0008 0649          btfsc  Num_2,2
0009 0A0C          goto   adjst1      ; Adjust LSD of Result
000A 0729          btfss  Num_2,1
000B 0A0E          goto   Over_1      ; No : Go for MSD
000C 0C06      adjst1  movlw  6
000D 00A9          subwf  Num_2
000E 0708      Over_1  btfss  Num_1,0      ; CY = 0 ?
000F 0A17          goto   adjst2      ; Yes, adjust MSD of result
0010 0068          clrf   Num_1
0011 07E9          btfss  Num_2,7      ; No, test for MSD >9
0012 0800          RETLW  0
0013 06C9          btfsc  Num_2,6
0014 0A17          goto   adjst2
0015 07A9          btfss  Num_2,5
0016 0800          RETLW  0
0017 0C60      adjst2  movlw  60      ; add 6 to MSD
0018 00A9          subwf  Num_2
0019 0068          clrf   Num_1
001A 0703          btfss  STATUS,CARRY  ; test if underflow
001B 0800          RETLW  0
001C 0C01          movlw  1
001D 0028          movwf  Num_1
001E 0800      Over   RETLW  0
;
;*****
;                               Test Program
;*****
001F 0C23      main   movlw  23
0020 0028          movwf  Num_1      ; Set Num_1 = 23
0021 0C99          movlw  99
0022 0029          movwf  Num_2      ; Set Num_2 = 99
0023 0900          call   BCDSub      ; After subtraction, Num_2 = 76 ( 99-23 )
;                               ; and Num_1 = 0 ( indicates positive result
;
;                               movlw  99
0024 0C99          movwf  Num_1      ; Set Num_1 = 99
0025 0028          movlw  0
0026 0C00          movlw  0
0027 0029          movwf  Num_2      ; Set Num_2 = 0
;
;                               call   BCDSub      ; After subtraction, Num_2 = 1
0028 0900          call   BCDSub      ; and Num_1 = 1 ( indicates negative result
;                               ; -1 <- ( -99 )
;
;                               self   goto   self
0029 0A29          self   goto   self
;
;                               org    1FF
01FF 0A1F          goto   main
;
END

```

```

Errors   : 0
Warnings : 0

```


PIC16C5X Math Routines

APPENDIX N: SQUARE ROOT BY NEWTON-RAPHSON METHOD

MPASM B0.54

PAGE 1

```
list p=16c54,f=inhx8m,n=0
;*****
;
;           Square Root By Newton Raphson Method
;
;   This routine computes the square root of a 16 bit number(with
;   low byte in NumLo & high byte in NumHi ). After loading NumLo &
;   NumHi with the desired number whose square root is to be computed,
;   branch to location Sqrt ( by "GOTO Sqrt" ). " CALL Sqrt" cannot
;   be issued because the Sqrt function makes calls to Math routines
;   and the stack is completely used up.
;   The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
;   The total number of iterations is set to ten. If more iterations
;   are desired, change "LupCnt equ .10" to the desired value. Also,
;   the initial guess value of the square root is given set as
;   input/2 ( in subroutine "init" ). The user may modify this scheme
;   if a better initial approximation value is known. A good initial
;   guess will help the algorithm converge at a faster rate and thus
;   less number of iterations required.
;   Two utility math routines are used by this program : D_divs
;   and D_add. These two routines are listed as separate routines
;   under double precision Division and double precision addition
;   respectively.
;
;   Note : If square root of an 8 bit number is desired, it is probably
;   better to have a table look scheme rather than using numerical
;   methods.
;
;
;   Performance :
;
;           Program Memory :      27 (excluding Math Routines
;                                   D_divs & D_add )
;           Clock Cycles   :      3600 ( approximately )
;
;   To assemble this program, two routines, namely "D_add" &
;   "D_divs" must be included into this program. These two routines
;   are listed as separate programs in files "DBL_ADD.ASM" &
;   "DBL_DIVS.ASM" respectively.
;*****
;
0010 ACCaLO equ 10
0011 ACCaHI equ 11
0012 EXPa equ 12
0013 ACCbLO equ 13
0014 ACCbHI equ 14
0015 EXPb equ 15
0016 ACCcLO equ 16
0017 ACCcHI equ 17
0018 ACCdLO equ 18
0019 ACCdHI equ 19
001A temp equ 1A
001B sign equ 1B
;
;
;           org 0
;
000A LupCnt equ .10 ; Number of iterations
```

PIC16C5X Math Routines

```

;
0010      SqrtLo equ   ACCaLO
0011      SqrtHi equ   ACCaHI
;
001D      NumLo  equ   1D
001E      NumHi  equ   1E
001F      count  equ   1F
;
;
init
0000 0C0A      movlw   LupCnt
0001 003F      movwf   count
0002 021E      movf    NumHi,W
0003 0031      movwf   SqrtHi
0004 021D      movf    NumLo,W      ; set initial guess root = NUM/2
0005 0030      movwf   SqrtLo
0006 0403      bcf     STATUS,CARRY
0007 0331      rrf     SqrtHi
0008 0330      rrf     SqrtLo
0009 0800      retlw   0
;
div2
000A 0403      bcf     STATUS,CARRY
000B 0314      rrf     ACCbHI,W
000C 0031      movwf   SqrtHi
000D 0313      rrf     ACCbLO,W
000E 0030      movwf   SqrtLo
000F 0800      retlw   0
;
Sqrt
0010 0900      call    init
sloop
0011 021D      movf    NumLo,W
0012 0033      movwf   ACCbLO
0013 021E      movf    NumHi,W
0014 0034      movwf   ACCbHI
;
0015 091B      call    D_divS      ; double precision division
0016 0947      call    D_add      ; double precision addition
;
; the above 2 routines are listed
; as seperate routines
;
0017 090A      call    div2
0018 02FF      decfsz  count
0019 0A11      goto   sloop
001A 0A53      goto   over      ; all iterations done
;
; branch back to desired location
;*****
; double precision division is placed here
include "d_divS.asm"
;*****
; double precision addition is placed here
include "d_add.asm"
;*****
;
;
;*****
;
; Test Program
;*****
;
004E 0CF3      main   movlw   0F3
004F 003E      movwf  NumHi
0050 0CF6      movlw  0F6      ; Set input test number = 62454
0051 003D      movwf  NumLo    ; = F3F6h
;
0052 0A10      goto   Sqrt     ; cannot use CALL : Math routines
;
; use up all the stack.
0053 0000      over   nop
;
; all iterations done
;
0054 0A54      self   goto   self      ; result = 00F9h = 249

```

PIC16C5X Math Routines

```
                                ; exact sqrt(62454) = 249.9
                                ;
                                ;
01FF 0A4E                        org PIC54
                                goto main
                                ;
                                END
```

```
Errors : 0
Warnings : 0
```

PIC16C5X Math Routines

NOTES:

Using PIC16C5X Microcontrollers as LCD Drivers

INTRODUCTION

This application report describes an LCD controller implementation using a PIC16C55 microcontroller. This technique offers display capabilities for applications that require a small display at a low cost together with the capabilities of the standard PIC15C55 microcontroller. We start by an overview of LCD devices and their theory of operation followed by software implementation issues of the controller. The source code for controlling a multiplexed LCD display is included in Appendix A.

LIQUID CRYSTAL DISPLAYS

The Liquid Crystal Display (LCD) is a thin layer of "Liquid Crystal Material" deposited between two plates of glass. The raw LCD is often referred to as "glass". Electrodes are attached to both sides of the glass. One side is referred to as common or backplane, while the other side is referred to as segment.

An LCD is modelled as a capacitor, with one side connected to the common plane and the other side connected to the segment, as shown in Figure 1. LCDs are sensitive to Root Mean Square Voltage levels. When a V_{RMS} level of zero volts is applied to the LCD, the LCD is practically transparent.

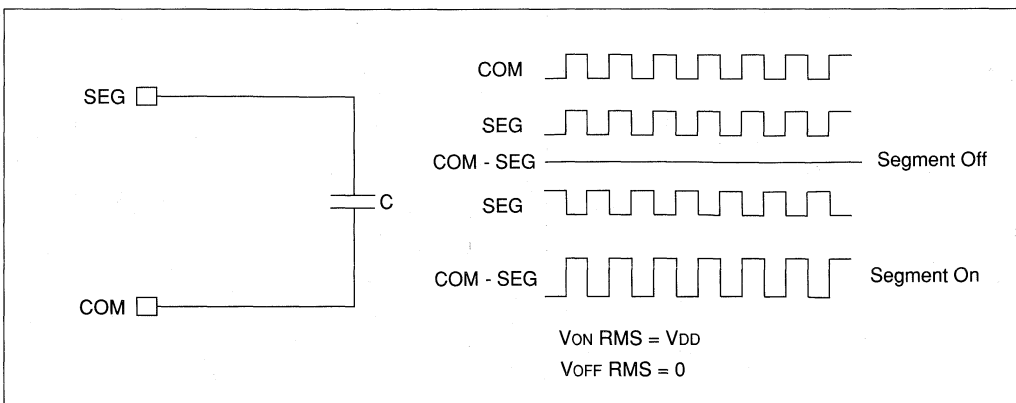
To turn an LCD segment "on", which makes the segment turn dark or opaque, an LCD RMS voltage that is greater than the LCD threshold voltage is applied to the LCD. The RMS LCD voltage is the RMS voltage across the capacitor C in Figure 1, which is equal to the potential difference between SEG and COM values.

Different LCDs have different characteristics; Figure 2 shows typical voltage vs relative contrast characteristics. Notation on curve shows operating points for multiplex operation with the threshold voltage set to 1.7 Vrms. This voltage is often used as the measure of voltage for LCD to be "off" or transparent. The curve is normalized and assumes a viewing angle of 90° to the plane of the LCD.

Contrast control, the process of turning on a segment, is achieved by moving the operating point of the LCD by applying voltage to the LCD that is greater than the LCD threshold voltages. A typical circuit to accomplish this task is shown in Figure 3.

Driving a liquid crystal display at direct current (DC) will cause permanent damage to the display unit. In order to prevent irreversible electrochemical action from destroying the display, the voltage at all segment locations must reverse polarity periodically so that a zero net voltage is applied to the device. This process is referred to as AC voltage application. There are two LCD driving methods available: Static driving method and multiplexed driving method.

FIGURE 1: ELECTRICAL MODEL OF AN LCD SEGMENT WITH DRIVING VOLTAGES



Using PIC16C5X Microcontrollers as LCD Drivers

Conventional LCDs have separate external connections for each and every segment plus a common plane. This is the most basic method that results in good display quality. The main disadvantage of this driving method is that each segment requires one liquid crystal driver. The static driving method uses the frame frequency, defined as a period of the common plane signal, of several tens to several hundred Hz. A lower frequency would result in blinking effects and higher frequencies would increase power requirements. To turn a segment on, a voltage that has an opposite polarity to the common plane signal must be applied resulting in a large RMS voltage across the plates. To turn off a segment a voltage that is of the same polarity to common plane signal is applied. This drive method is universal to driving LCD segments. Figure 1 shows an example of this driving method.

The LCD frequency is defined as the rate of output changes of the common plane and segment signals,

whereas the frame rate is defined as $f_{\text{frame}} = \frac{f_{\text{frame}}}{N}$

where N is the multiplex rate or number of backplane. Typically, f_{frame} ranges from 25HZ to 300HZ. The most commonly used frame frequency is 40-70HZ. A lower frequency would result in flicker effects and higher frequency would increase power requirements.

Multiplexed LCDs maintain their liquid crystal characteristics. These are low power consumption, high contrast ratio under high ambient light levels, and reduce the number of external connections necessary for dot matrix and alphanumeric displays. The multiplex driving method reduces the number of driver circuits, or microcontroller I/O pins if a software method is used. The method of drive for multiplexed displays is Time Division Multiplex (TDM) with the number of time divisions equal to twice the number of common planes used in a given format. In order to prevent permanent damage to the LCD display, the voltage at all segment locations must reverse polarity periodically so that zero net voltage is applied. This is the reason for the doubling in time divisions; each common plane must be alternately driven with a voltage pulse of opposite polarity. The drive frequency should be greater than the flicker rate of 25Hz. Since increasing the drive frequency significantly above this value increases current demand by the CMOS circuitry, an upper drive frequency level of 60Hz is recommended by most LCD manufacturers. We have chosen a drive rate of 50Hz for this application report which results in a frame period of 20 ms. The most commonly available formats are 2x4, 3x3, and 5x7. In this report we use a 2x4 format LCD to display hexadecimal digits.

FIGURE 2: TYPICAL LCD CHARACTERISTICS

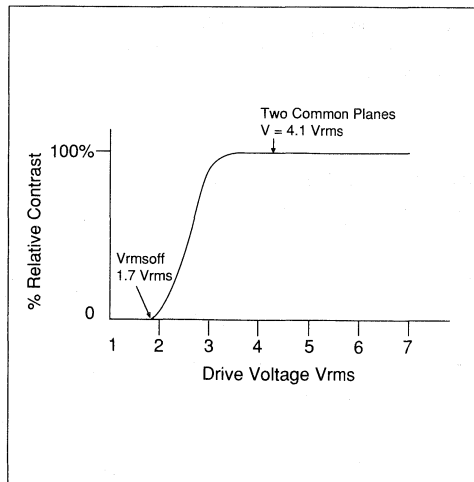
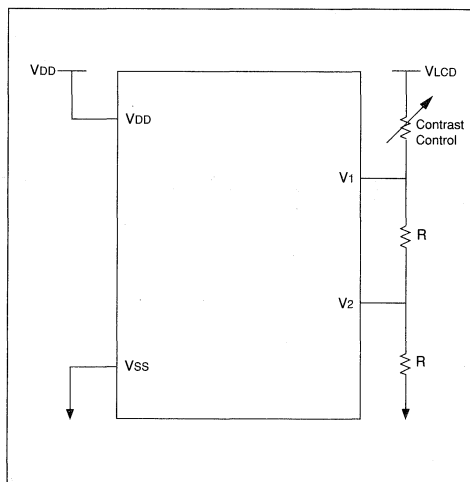


FIGURE 3: CONTRAST CONTROL CIRCUIT



Using PIC16C5X Microcontrollers as LCD Drivers

To better understand multiplexed LCD control it is best to look at the general case. The segments in a multiplexed LCD are arranged in an X-Y grid form as shown in Figure 4. The common plane signals maintain their relative shape at all times, as shown in Figure 5. To turn on segment 1 (SEG1), we need to apply a voltage V_d , such that $V_s + V_d$ turns the segment on and $V_s - V_d$ turns the segment off. Note that the segment signal V_d is symmetrical. This is a consequence of the intervals that the common plane signal is not present at all times. Use of nonsymmetrical waveform will result in a higher V_{rms} present on the unaddressed segments. The symmetri-

cal nature of the waveforms theoretically result in a zero DC voltage levels. CMOS drivers (e.g. microcontrollers) operate at 0 to +5V levels (rail voltage levels). This would require driving voltages beyond the range of operation. This constraint is addressed by a technique referred to as "level shifting" or "biasing". Level shifting allows application of voltages in the range of 0 to +2.5V, which is compatible with these drivers. This would require an additional voltage level of +2.5V, which can be implemented through a simple resistive voltage divider circuit.

FIGURE 4: MULTIPLEXED LCD SEGMENT ARRANGEMENT

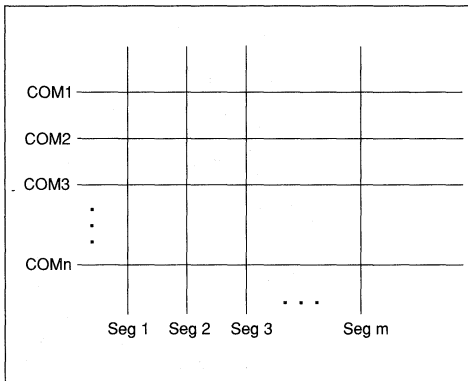
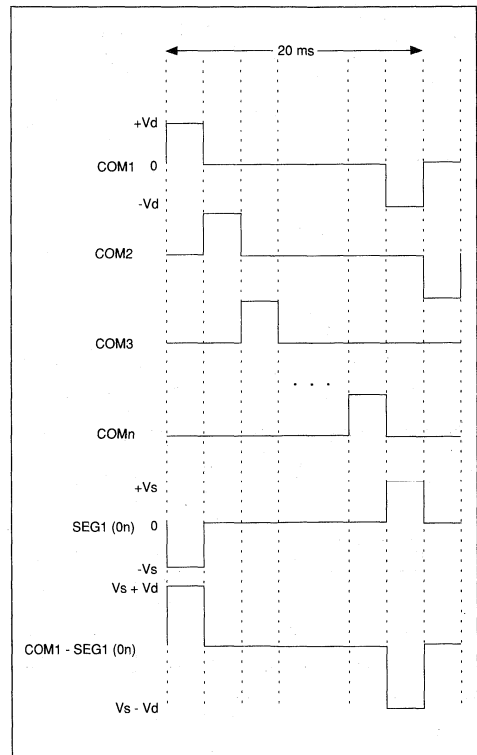


FIGURE 5: MULTIPLEXED LCD DRIVE WAVEFORMS



Using PIC16C5X Microcontrollers as LCD Drivers

IMPLEMENTATION

The ideas presented in the previous section can be applied to any size multiplexed LCD display. In our implementation we used a 4-digit LCD from Ocular Inc. [1]. The circuit diagram used in this application report is shown in Figure 6. Each I/O pin on the PIC16C55 device controls the state of two segments (see Figure 6) which requires a total of 16 I/O pins. The reference voltages are generated through a simple

resistive voltage divider circuit. The voltage levels are generated by taking advantage of PIC16C5X I/O pin set to input, which tristates the voltage level seen on the pin. This method uses 4 I/O pins to generate the proper voltage levels. Figure 7 shows the truth table for generating the voltage levels. Figure 8 shows how to create a bitmap for different digits. Figure 9 shows the waveforms generated for the accompanying software which implements a hexadecimal counter.

FIGURE 6: SYSTEM CONFIGURATION WITH LCD PINOUT

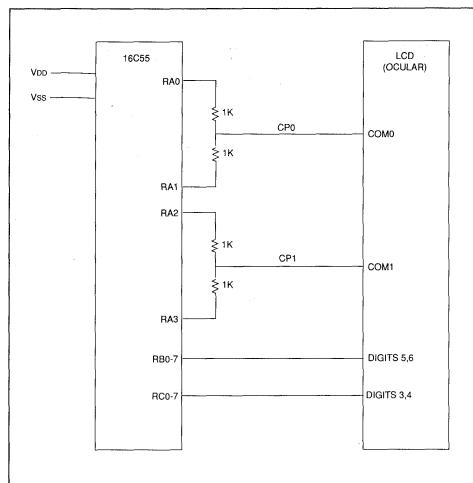


FIGURE 7: COMMON PLANE SIGNAL GENERATION

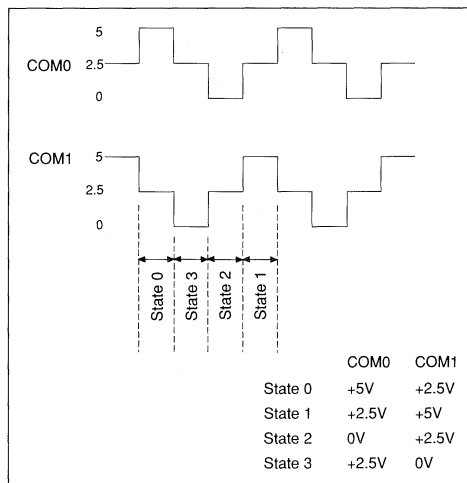
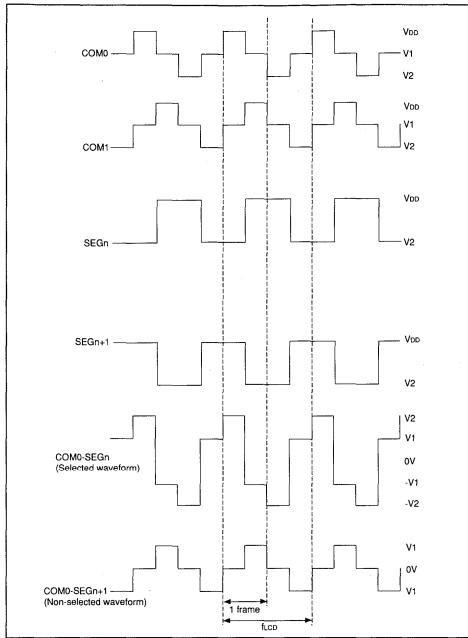


FIGURE 8: LCD CHARACTER BITMAP

Digit	COM0 SEG 0				COM1 SEG 1				COM0 SEG 2				COM1 SEG 3			
	F	E	D	DP	A	G	C	B	F	E	D	DP	A	G	C	B
0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1
2	1	0	0	1	0	0	0	1	0	1	1	0	1	1	1	0
3	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1
4	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1	1
5	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	1
6	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
7	1	1	1	1	0	0	1	0	0	0	0	0	1	1	0	1
8	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1
9	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1
a	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1
b	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1
c	1	0	0	1	1	1	0	1	0	1	1	0	0	0	1	0
d	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1
e	0	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0
f	0	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0

Using PIC16C5X Microcontrollers as LCD Drivers

FIGURE 9: EXAMPLE OF OUTPUT WAVEFORMS FOR DIGIT 4



CONCLUSION

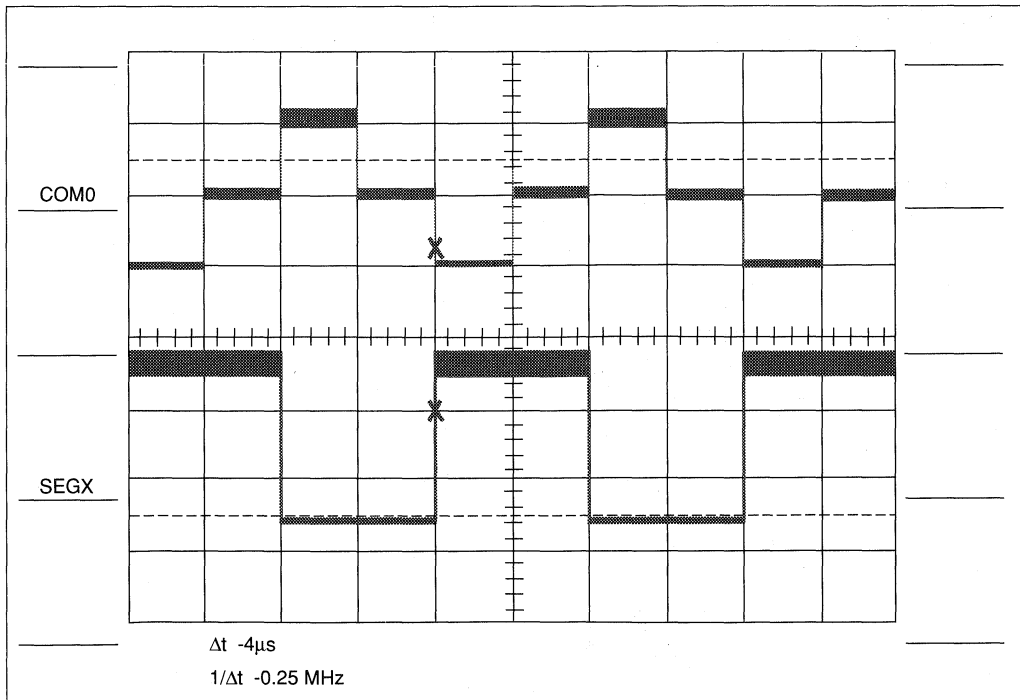
In this application report we have demonstrated the use of PIC16C5X devices to implement a simple LCD controller. As discussed earlier, it is important to keep the generated DC voltage to a minimum to extend the life of the LCD. Ideally one should switch all the I/O lines simultaneously, however, a software implementation of the LCD controller will necessarily introduce a delay which is proportional to the instruction cycle of the microcontroller, as shown in Figure 10. Therefore it is necessary to keep the switching time to a minimum. Our implementation introduced less than 50 mV of DC voltage on the segment lines which is below the manufacturer's recommended DC offset voltage of 60 mV.

REFERENCES

- [1] Ocular Inc., Drawing number JH074.

*AUTHOR: Al Lovrich
Logic Products Division*

FIGURE 10: MICROCONTROLLER GENERATED OUTPUT WAVEFORM



Using PIC16C5X Microcontrollers as LCD Drivers

MPASM B0.54

PAGE 1

```
LIST      C=132,n=0,p=16c55,r=dec

;*****
; Project: PIC16C5X as a multiplexed LCD driver.      *
;
*
; Revision history:
*
;      04/14/93      original
*
;*****

; Equates

01FF  pic54  equ   0x1fff          ; Define Reset Vectors
01FF  pic55  equ   0x1fff
03FF  pic56  equ   0x3fff
07FF  pic57  equ   0x7fff

0001  rtcc   equ   1              ; f1
0002  pc     equ   2              ; f2
0003  status equ   3              ; f3
0004  fsr    equ   4              ; f4

0005  porta  equ   5              ; f5
0006  portb  equ   6              ; f6
0007  portc  equ   7              ; f8

; realtime mode registers

0008  currentState  equ   8
0009  msTimer  equ   currentState+1 ; Millisecond timer
000A  sTimerLow  equ   msTimer+1 ; Lower byte second timer
000B  sTimerHigh equ   sTimerLow+1 ; Upper byte second timer
000C  digit56  equ   sTimerHigh+1
000D  digit34  equ   digit56+1

; Misc definitions

0060  FIVEMSEC  equ   96          ; Assuming 4.096 MHz crystal

0000  w        equ   0
0001  f        equ   1

0002  z        equ   2

; Status register bits
;*****
; Port assignments      *
;      *
; porta - bit0: Common Plane 0 *
;      bit1: Common Plane 0 *
;      bit2: Common Plane 1 *
;      bit3: Common Plane 1 *
;      *
; portb - bit0: 6B/DP      *
;      bit1: 6C/6D      *
;      bit2: 6G/6E      *
;      bit3: 6A/6F      *
;      bit4: 5B/DP      *
;      bit5: 5C/5D      *
;      bit6: 5G/5E      *
```

Using PIC16C5X Microcontrollers as LCD Drivers

```

;          bit7: 5A/5F          *
;          *
; portc - bit0: 4B/DP          *
;          - bit1: 4C/4D          *
;          - bit2: 4G/4E          *
;          - bit3: 4A/4F          *
;          - bit4: 3B/DP          *
;          - bit5: 3C/3D          *
;          - bit6: 3G/3E          *
;          - bit7: 3A/3F          *
;
;          *
;*****
;*****
;          Macro definitions          *
;*****
UpdateState macro State, Table

        swapf   sTimerLow, w
        andlw   0xf                      ; Isolate digit 5 (offset)
        call    Table
        movwf   digit56
        swapf   digit56, f

        movf    sTimerLow, w
        andlw   0xf                      ; Isolate digit 6 (offset)
        call    Table
        iorwf   digit56, f

        swapf   sTimerHigh, w
        andlw   0xf                      ; Isolate digit 5 (offset)
        call    Table
        movwf   digit34
        swapf   digit34, f

        movf    sTimerHigh, w
        andlw   0xf                      ; Isolate digit 6 (offset)
        call    Table
        iorwf   digit34, f

        movf    digit34, w                ; Display digits 3 & 4
        movwf   portc
        movf    digit56, w
        movwf   portb                    ; Display digits 5 & 6
        endm

        org     0

; Initialize ports A, B, and C and RTCC. In case of output data
; values, set the data latch first, then set the port direction.

Initialize
0000 0C01    movlw   00000001b           ; Set data latch
0001 0025    movwf   porta
0002 0C08    movlw   00001000b           ; Set I/O direction
0003 0005    tris    porta

0004 0C00    movlw   00000000b           ; Set levels to low
0005 0026    movwf   portb
0006 0C00    movlw   00000000b           ; Set as outputs
0007 0006    tris    portb

0008 0C00    movlw   00000000b           ; Set levels to low
0009 0027    movwf   portc
000A 0C00    movlw   00000000b           ; Set as outputs
000B 0007    tris    portc

000C 0C04    movlw   0x04                 ; Set prescaler
000D 0002    option

```

Using PIC16C5X Microcontrollers as LCD Drivers

```
000E 0C60      movlw   FIVEMSEC           ; rtcc = 5ms
000F 0021      movwf   rtcc

0010 0C04      movlw   4
0011 0028      movwf   currentState

0012 0C0D      movlw   0xd
0013 0029      movwf   msTimer           ; Initialize millisecond timer

0014 006A      clrfl   sTimerLow        ; Clear second counter
0015 006B      clrfl   sTimerHigh

0016 0800      retlw   0

; Check timer register for timing out (rtcc = 0). Remain in the
; loop until the timer times out.

; Wait for 5ms timer timeout

Timer_Check
0017 0201      movf    rtcc, w
0018 0743      btfsz  status, z
0019 0A17      goto   Timer_Check

001A 0C60      movlw   FIVEMSEC
001B 0021      movwf   rtcc

001C 02E9      decfsz msTimer, f
001D 0A21      goto   Update_Backplane

001E 03EA      incfsz sTimerLow        ; Update second counter
001F 0A21      goto   Update_Backplane
0020 02AB      incf   sTimerHigh, f

; RA0 and RA1 are used to control voltage level for common plane 0.
; RA2 and RA3 are used to control voltage level for common plane 1.
; There are four possible states with different voltage levels as
; follows:
;
; State 0 - cp0 = +5v  ra0=1, ra1=x
;          cp1 = +2.5v ra2=1, ra3=0
; State 1 - cp0 = +2.5v ra0=1, ra1=0
;          cp1 = +5v   ra2=1, ra3=x
; State 2 - cp0 = 0v   ra0=0, ra1=x
;          cp1 = +2.5v ra2=1, ra3=0
; State 3 - cp0 = +2.5v ra0=1, ra1=0
;          cp1 = 0v   ra2=0, ra3=x

Update_Backplane
0021 0004      clrwdt           ; Reset watchdog timer

0022 00C8      decf    currentState, w ; Update w register
0023 0E03      andlw  0x03       ; Use only bit0/1
0024 0028      movwf  currentState ; Update currentState
0025 01E2      addwf  pc, f

0026 0AAD      goto   State3
0027 0A81      goto   State2
0028 0A55      goto   State1
;          goto   State0

;          State 0
;          State 0
UpdateState   State0, S0_Table
```

Using PIC16C5X Microcontrollers as LCD Drivers

```
0029 038A      swapf  sTimerLow, w
002A 0E0F      andlw  0xf           ; Isolate digit 5 (offset)
002B 0944      call   S0_Table
002C 002C      movwf  digit56
002D 03AC      swapf  digit56, f

002E 020A      movf   sTimerLow, w
002F 0E0F      andlw  0xf           ; Isolate digit 6 (offset)
0030 0944      call   S0_Table
0031 012C      iorwf  digit56, f

0032 038B      swapf  sTimerHigh, w
0033 0E0F      andlw  0xf           ; Isolate digit 5 (offset)
0034 0944      call   S0_Table
0035 002D      movwf  digit34
0036 03AD      swapf  digit34, f

0037 020B      movf   sTimerHigh, w
0038 0E0F      andlw  0xf           ; Isolate digit 6 (offset)
0039 0944      call   S0_Table
003A 012D      iorwf  digit34, f

003B 020D      movf   digit34, w     ; Display digits 3 & 4
003C 0027      movwf  portc
003D 020C      movf   digit56, w
003E 0026      movwf  portb         ; Display digits 5 & 6

003F 0C05      movlw  00000101b
0040 0025      movwf  porta
0041 0C02      movlw  00000010b
0042 0005      tris   porta

0043 0800      retlw  0

      S0_Table
0044 01E2      addwf  pc, f         ; Add offset to pc

0045 0804      retlw  0100b        ; 0
0046 080C      retlw  1100b        ; 1
0047 0802      retlw  0010b        ; 2
0048 0800      retlw  0000b        ; 3
0049 0808      retlw  1000b        ; 4
004A 0801      retlw  0001b        ; 5
004B 080F      retlw  1111b        ; 6
004C 0804      retlw  0100b        ; 7
004D 0800      retlw  0000b        ; 8
004E 0800      retlw  0000b        ; 9
004F 0800      retlw  0000b        ; a
0050 0809      retlw  1001b        ; b
0051 080B      retlw  1011b        ; c
0052 0808      retlw  1000b        ; d
0053 0803      retlw  0011b        ; e
0054 0803      retlw  0011b        ; f

      ; State 1

      State1
      UpdateState      State1, S1_Table

0055 038A      swapf  sTimerLow, w
0056 0E0F      andlw  0xf           ; Isolate digit 5 (offset)
0057 0970      call   S1_Table
0058 002C      movwf  digit56
0059 03AC      swapf  digit56, f

005A 020A      movf   sTimerLow, w
005B 0E0F      andlw  0xf           ; Isolate digit 6 (offset)
005C 0970      call   S1_Table
```

Using PIC16C5X Microcontrollers as LCD Drivers

```
005D 012C          iorwf  digit56, f

005E 038B          swapf  sTimerHigh, w
005F 0E0F          andlw  0xf                ; Isolate digit 5 (offset)
0060 0970          call   S1_Table
0061 002D          movwf  digit34
0062 03AD          swapf  digit34, f

0063 020B          movf   sTimerHigh, w
0064 0E0F          andlw  0xf                ; Isolate digit 6 (offset)
0065 0970          call   S1_Table
0066 012D          iorwf  digit34, f

0067 020D          movf   digit34, w        ; Display digits 3 & 4
0068 0027          movwf  portc
0069 020C          movf   digit56, w
006A 0026          movwf  portb            ; Display digits 5 & 6

006B 0C05          movlw  00000101b
006C 0025          movwf  porta
006D 0C08          movlw  00001000b
006E 0005          tris   porta

006F 0800          retlw  0

S1_Table
0070 01E2          addwf  pc, f

0071 0801          retlw  0001b            ; 0
0072 080F          retlw  1111b           ; 1
0073 0809          retlw  1001b           ; 2
0074 080D          retlw  1101b           ; 3
0075 0807          retlw  0111b           ; 4
0076 0805          retlw  0101b           ; 5
0077 080F          retlw  1111b           ; 6
0078 080F          retlw  1111b           ; 7
0079 0801          retlw  0001b           ; 8
007A 0805          retlw  0101b           ; 9
007B 0803          retlw  0011b           ; a
007C 0801          retlw  0001b           ; b
007D 0809          retlw  1001b           ; c
007E 0809          retlw  1001b           ; d
007F 0801          retlw  0001b           ; e
0080 0803          retlw  0011b           ; f

; State 2

State2
UpdateState      State2, S2_Table

0081 038A          swapf  sTimerLow, w
0082 0E0F          andlw  0xf                ; Isolate digit 5 (offset)
0083 099C          call   S2_Table
0084 002C          movwf  digit56
0085 03AC          swapf  digit56, f

0086 020A          movf   sTimerLow, w
0087 0E0F          andlw  0xf                ; Isolate digit 6 (offset)
0088 099C          call   S2_Table
0089 012C          iorwf  digit56, f

008A 038B          swapf  sTimerHigh, w
008B 0E0F          andlw  0xf                ; Isolate digit 5 (offset)
008C 099C          call   S2_Table
008D 002D          movwf  digit34
008E 03AD          swapf  digit34, f
```

Using PIC16C5X Microcontrollers as LCD Drivers

```
008F 020B      movf   sTimerHigh, w
0090 0E0F      andlw  0xf                ; Isolate digit 6 (offset)
0091 099C      call   S2_Table
0092 012D      iorwf  digit34, f

0093 020D      movf   digit34, w        ; Display digits 3 & 4
0094 0027      movwf  portc
0095 020C      movf   digit56, w
0096 0026      movwf  portb            ; Display digits 5 & 6

0097 0C04      movlw  00000100b
0098 0025      movwf  porta
0099 0C02      movlw  00000010b
009A 0005      tris   porta

009B 0800      retlw  0

S2_Table
009C 01E2      addwf  pc, f

009D 080B      retlw  1011b ; 0
009E 0803      retlw  0011b ; 1
009F 080D      retlw  1101b ; 2
00A0 080F      retlw  1111b ; 3
00A1 0807      retlw  0111b ; 4
00A2 080E      retlw  1110b ; 5
00A3 080E      retlw  1110b ; 6
00A4 080B      retlw  1011b ; 7
00A5 080F      retlw  1111b ; 8
00A6 080F      retlw  1111b ; 9
00A7 080F      retlw  1111b ; a
00A8 0806      retlw  0110b ; b
00A9 0804      retlw  0100b ; c
00AA 0807      retlw  0111b ; d
00AB 080C      retlw  1100b ; e
00AC 080C      retlw  1100b ; f

; State 3

State3
UpdateState   State3, S3_Table

00AD 038A      swapf  sTimerLow, w
00AE 0E0F      andlw  0xf                ; Isolate digit 5 (offset)
00AF 09C8      call   S3_Table
00B0 002C      movwf  digit56
00B1 03AC      swapf  digit56, f

00B2 020A      movf   sTimerLow, w
00B3 0E0F      andlw  0xf                ; Isolate digit 6 (offset)
00B4 09C8      call   S3_Table
00B5 012C      iorwf  digit56, f

00B6 038B      swapf  sTimerHigh, w
00B7 0E0F      andlw  0xf                ; Isolate digit 5 (offset)
00B8 09C8      call   S3_Table
00B9 002D      movwf  digit34
00BA 03AD      swapf  digit34, f

00BB 020B      movf   sTimerHigh, w
00BC 0E0F      andlw  0xf                ; Isolate digit 6 (offset)
00BD 09C8      call   S3_Table
00BE 012D      iorwf  digit34, f

00BF 020D      movf   digit34, w        ; Display digits 3 & 4
00C0 0027      movwf  portc
```

Using PIC16C5X Microcontrollers as LCD Drivers

```
00C1 020C          movf    digit56, w
00C2 002E          movwf   portb          ; Display digits 5 & 6

00C3 0C01          movlw   00000001b
00C4 0025          movwf   porta
00C5 0C08          movlw   00001000b
00C6 0005          tris    porta

00C7 0800          retlw   0

S3_Table
00C8 01E2          addwf   pc, f

00C9 080E          retlw   1110b          ; 0
00CA 0800          retlw   0000b          ; 1
00CB 0806          retlw   0110b          ; 2
00CC 0802          retlw   0010b          ; 3
00CD 0808          retlw   1000b          ; 4
00CE 080A          retlw   1010b          ; 5
00CF 080E          retlw   1110b          ; 6
00D0 0800          retlw   0000b          ; 7
00D1 080E          retlw   1110b          ; 8
00D2 080A          retlw   1010b          ; 9
00D3 080C          retlw   1100b          ; a
00D4 080E          retlw   1110b          ; b
00D5 0806          retlw   0110b          ; c
00D6 0806          retlw   0110b          ; d
00D7 080E          retlw   1110b          ; e
00D8 080C          retlw   1100b          ; f

; Main code

Start
00D9 0900          call   Initialize
Repeat
00DA 0917          call   Timer_Check
00DB 0ADA          goto   Repeat

org    pic55

System_Reset
01FF 0AD9          goto   Start

END

Errors : 0
Warnings : 0
```


SECTION 3

PIC16CXX APPLICATION NOTES

Using the Analog to Digital Converter - AN546	3- 1
Implementing Wake Up on Keystroke - AN552	3- 21
PortB as External Interrupt - AN566	3- 25
Table Read Using PIC16CXX - AN556	3- 29
Software Implementation of I ² C Bus Master - AN554	3- 33
Software Implementation of Asynchronous Serial I/O - AN555	3- 121
Four Channel Digital Volt Meter with Display and Keyboard - AN557	3- 157

Using the Analog to Digital Converter

INTRODUCTION

This application note is intended for PIC16C71 users with various degrees of familiarity with analog system design. The various sections discuss the following topics:

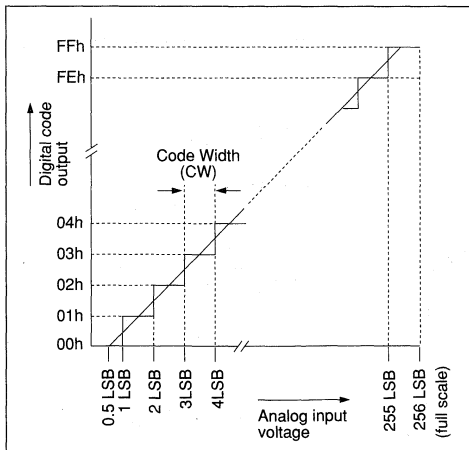
- Commonly used A/D terminology
- How to configure and use the PIC16C71 A/D
- Various ways to generate external reference voltage (VREF)
- Configuring RA0-RA3 pins

COMMONLY USED A/D TERMINOLOGY

The Ideal Transfer Function

In an A/D converter, an analog voltage is mapped into an N-bit digital value. This mapping function is defined as the transfer function. An ideal transfer is one in which there are no errors or non-linearity. It describes the "ideal" or intended behavior of the A/D. Figure 1 shows the ideal transfer function for the PIC16C71 A/D. Note that the digital output value is 00h for analog input voltage range of 0 to 1LSB. In some converters, the first transition point is at 0.5LSB and not at 1LSB as shown in Figure 2. Either way, knowing the transfer function the user can appropriately interpret the data.

FIGURE 1 - PIC16C71 IDEAL TRANSFER FUNCTION



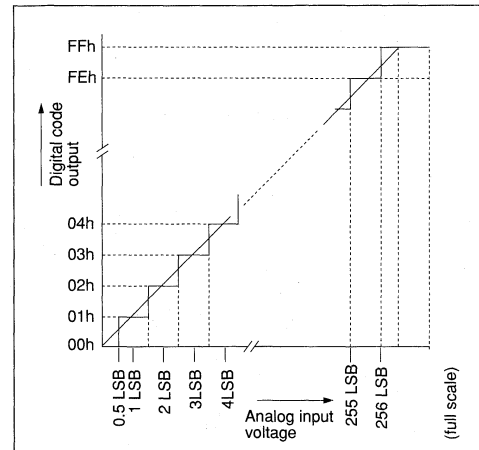
Transition Point

It is the analog input voltage at which the digital output switches from one code to the next. The transition point is typically not a single threshold, rather a small region of uncertainty (see Figure 3) The transition point is therefore defined as the statistical average of many conversions. Stated differently, it is the voltage input at which the uncertainty of the conversion is 50%.

Code Width

It is the distance (voltage differential) between two transition points. Ideally the Code Width should be 1LSB. See Figure 1.

FIGURE 2 - ALTERNATE TRANSFER FUNCTION

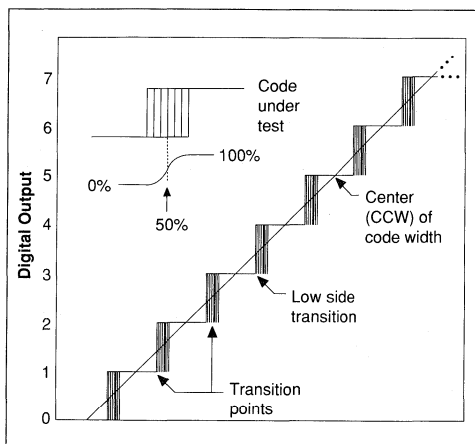


Using the Analog to Digital Converter

Center of Code Width

It is the midpoint between two transition points. See Figure 3.

FIGURE 3 - TRANSITION POINTS



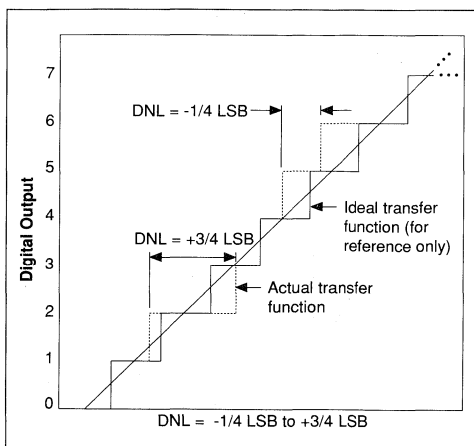
Differential Non-Linearity (DNL)

It is the deviation in code-width from 1LSB (Figure 7). The difference is calculated for each and every transition. The largest difference is reported as DNL.

It is important to note that the DNL is measured after the transfer function is normalized to match offset error and gain error.

Note that the DNL cannot be any less than -1LSB . In the other direction, DNL can be $>1\text{LSB}$.

FIGURE 7 - DIFFERENTIAL NON-LINEARITY



Absolute Error

The maximum deviation between any transition point from the corresponding ideal transfer function is defined as the absolute error. This is how it is measured and reported in the PIC16C71 (Figure 8). The notable difference between absolute error and INL is that the measured data is not normalized for full scale and offset errors.

It is probably the first parameter the user will look at to evaluate an A/D. Sometimes absolute error is reported as the sum of offset, full-scale and integral non linearity errors.

Total Unadjusted Error

It is the same as absolute error. Again, sometimes it is reported as the sum of offset, full-scale and integral non-linearity errors.

No Missing Code

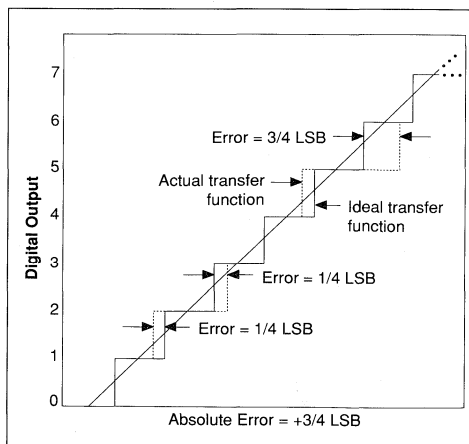
No missing code implies that as the analog input voltage is gradually increased from zero to full scale (or vice versa), all digital codes are produced. Stated otherwise, changing analog input voltage from one quantum of the analog range to the next adjacent range will not produce a change in the digital output by more than one code count.

Monotonic

Monotonicity guarantees that an increase (or decrease) in the analog input value will result in an equal or greater digital code (or less). Monotonicity does not guarantee that there are no missing codes. However, it is an important criterion for feedback control systems. Non monotonicity may cause oscillations in such a system.

The first derivative of a monotonic function always has the same sign.

FIGURE 8 - ABSOLUTE ERROR



Using the Analog to Digital Converter

Ratiometric Conversion

It is the A/D conversion process where the binary result is a ratio of the supply voltage or reference voltage, the latter being equal to full-scale value by default. The PIC16C71 is a ratiometric A/D converter where the result depends on V_{DD} or V_{REF} .

In some A/D's, an absolute reference is provided resulting in "absolute conversion".

Sample and Hold

In sample and hold type A/D converters, the analog input has a switch (typically a FET switch in CMOS) which is opened for a short duration to capture the analog input voltage onto an on-chip capacitor. Conversion is typically started after the sampling switch is closed.

Track and Hold

It is basically the same as sample and hold, except the sampling switch is typically left on. Therefore the voltage on the on-chip holding capacitor "tracks" the analog input voltage. To begin a conversion, the sampling switch is shut off.

The PIC16C71 A/D falls in this category.

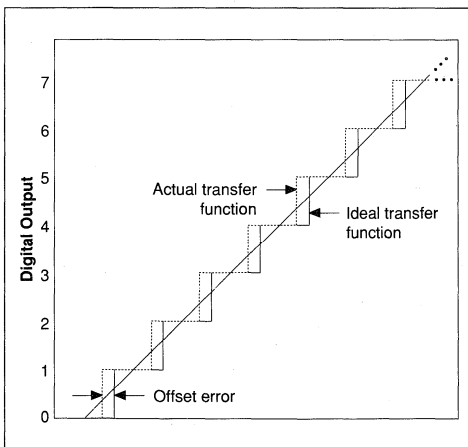
Sampling Time

It is the time required to charge the on-chip holding capacitor to the same value as on the analog input pin. The sampling time depends on the magnitude of the holding capacitor and the source impedance of the analog voltage input.

Offset Error (or Zero Error)

It is the difference between the first actual (measured) transition point and the first ideal transition point as shown in Figure 4. It can be corrected by the user by subtracting the offset error from each conversion result.

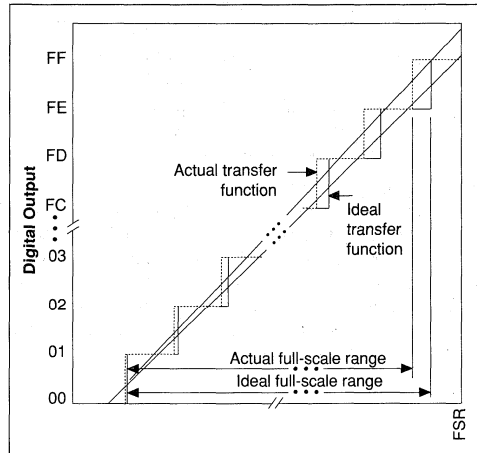
FIGURE 4 - OFFSET ERROR



Full Scale Error (or Gain Error)

It is the difference between the ideal Full Scale and the actual (measured) full scale range (see Figure 5). It is also called gain error, because the error changes the slope of the ideal transfer function creating a gain factor. It can be corrected by the user by multiplying each conversion result by the inverse of the gain.

FIGURE 5 - FULL SCALE ERROR

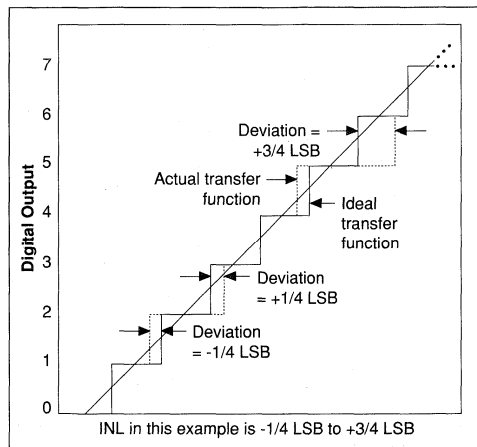


Integral Non-Linearity (INL), or Relative Error

It is the deviation of a transition point from its corresponding point on the ideal transfer point curve (Figure 6). The maximum difference is reported as the INL of the converter.

It is important to note that Full Scale Error and the Offset Error are normalized to match end transition points before measuring the INL.

FIGURE 6 - INTEGRAL NON-LINEARITY



Using the Analog to Digital Converter

HOW TO USE THE PIC16C71 A/D

The A/D in the PIC16C71 is easy to set up and use. There are a few considerations:

1. Select either V_{DD} or V_{REF} as reference voltage. More on using V_{REF} input later.
2. Select A/D conversion clock (tad): 2 tosc, 8 tosc, 32 tosc or trc (internal RC clock). For the first three options, make sure that tad ≥ 2.0 μs. If deterministic conversion time is required, select tosc time base. If conversion during SLEEP is required, select trc.
3. Channel Selection : If only one A/D channel is required, program the ADCON1 register to 03h. This configures the A/D pins as digital I/O. If multiple channels are required, prior to each conversion the new channel must be selected.
4. Sampling and Conversion: After a new channel is selected, a minimum amount of sampling time must be allowed before GO bit in ADCON0 is set to begin conversion. Once conversion begins, it is OK to select the next channel, **but sampling does not begin until current conversion is complete**. Therefore, it is always necessary to provide minimum required sampling time
 - i) after a conversion
 - ii) after a new channel is selected
 - iii) after A/D is turned on (ADON = 1).
5. Reading Result: Completion of conversion can be determined by either polling GO/DONE bit to cleared, polling the ADIF bit to be set, or waiting for an ADIF interrupt.

ADDITIONAL TIPS:

1. The GO bit and the ADON bit may not be set at once. After the A/D is turned on by setting ADON, at least 5μs time must be allowed before conversion begins, longer if sampling time requirement is not met within 5μs.
2. Aborting a conversion: A conversion can be aborted by clearing GO bit. The A/D converter will stop conversion and revert back to sampling state.
3. Using ADRES register as a normal register: The A/D only writes to ADRES at the end of a conversion. Therefore, it is possible to use ADRES as a normal file register between conversions and when A/D is off.

The following are a few examples of using the A/D.

Example 1: How to do a simple ADC conversion.

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Always ch2, internal RC OSC.
InitializeAD
    bsf        STATUS, 5    ; select pg1
    movlw     B'00000000'   ; select RA0-RA3...
    movwf    ADCON1        ; as analog inputs
    bcf        STATUS, 5    ; select pg0
    movlw     B'11010001'   ; select: RC osc, ch2...
    movwf    ADCON0        ; turn on A/D
Convert    call    sample-delay ; provide necessary sampling time
;
    bsf        ADCON0, 2    ; start new A/D conversion
loop
    btfsc    ADCON0, 2    ; A/D over?
    goto     loop          ; no then loop
;
    movf     ADRES, w      ; yes then get A/D value
;
```

A detailed code listing is in Appendix A.

Using the Analog to Digital Converter

Example 2: How to do sequential channel conversions.

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3 in a round robin fashion, internal RC OSC.
; Load results in 4 consecutive addresses starting at ADTABLE (10h)
;
InitializeAD
    bsf        STATUS, 5      ; select pgl
    movlw     B'00000000'    ; select RA0-RA3..
    movwf    ADCON1         ; as analog inputs
    bcf        STATUS, 5      ; select pg0
    movlw     B'11000001'    ; select: RC osc, ch0...
    movwf    ADCON0         ; turn on A/D
    movlw     ADTABLE       ; point fsr to top of...
    movwf    FSR            ; table

;
new_ad    call    sample_delay ; provide necessary sampling time
          bsf     ADCON0, 2    ; start new A/D conversion

loop
          btfs   ADCON0, 2    ; A/D over?
          goto   loop        ; no then loop

;
          movf   adres, w     ; yes then get A/D value
          movwf  0            ; load indirectly
          movlw  4            ; select next channel
          addwd  ADCON0       ; /
          bcf   ADCON0, 5     ; reset carry over bit.
; increment pointer to correct table offset.
          clrf   temp        ; clear temp register
          btfs   ADCON0, 3    ; test lsb of channel select
          bsf   temp, 0       ; set if chl selected
          btfs   ADCON0, 4    ; test msb of channel select
          bsf   temp, 1       ; /
          movlw  ADTABLE     ; get table address
          addwf  temp, w     ; add with temp
          movwf  FSR         ; move into indirect
          goto   new_ad

;
```

A detailed code listing is in Appendix B.

Using the Analog to Digital Converter

Example 3: How to write the interrupt handler for the ADC.

```
        org      0x00
        goto     start
        org      0x04
        goto     service_ad      ; interrupt vector
;
;      org      0x10
start
        movlw   B'00000000'      ;init I/O ports
        movwf   PORT_B
        tris    PORT_B
;
        call    InitializeAD
update
        bcf     flag,adover      ; reset software A/D flag
        call    SetupDelay      ; setup delay >= 10uS.
        bcf     ADCON0,adif     ; reset A/D int flag (ADIF)
        bsf     ADCON0,adgo     ; start new A/D conversion
        bsf     INTCON,gie      ; enable global interrupt
loop
        btfsc   flag,adover     ; A/D over?
        goto    update          ; yes start new conv.
        goto    loop            ; no then keep checking
; InitializeAD, initializes and sets up the A/D hardware.
; select ch0 to ch3, RC OSC., a/d interrupt.
InitializeAD
        bsf     STATUS, 5       ; select pg1
        movlw   B'00000000'     ; select RA0-RA3...
        movwf   ADCON1         ; as analog inputs
        bcf     STATUS, 5       ; select pg0
        clrf    INTCON          ; clr all interrupts
        bsf     INTCON, 6       ; enable A/D int.
        movlw   B'11010001'     ; select: RC osc, ch2...
        movwf   ADCON0         ; turn on A/D
        return
;
service_ad
        btfss   ADCON0, 1       ; A/D interrupt?
        retfie  ; no then ignore
        movf    ADRES, W        ; get A/D value
        return  ; do not enable int
;
```

A detailed code listing is in Appendix C.

Using the Analog to Digital Converter

Example 4: How to do conversions during sleep mode.

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3, internal RC OSC.
; While doing the conversion put unit to sleep. This will
; minimize digital noise interference.
; Note that ad's RC osc. has to be selected in this instance.
;
InitializeAD
    bsf        STATUS, 5        ; select pg1
    movlw     B'00000000'      ; select RA0-RA3...
    movwf    ADCON1           ; as analog inputs
    bcf        STATUS, 5        ; select pg0
    movlw     B'11000001'      ; select: RC osc, ch0...
    movwf    ADCON0           ; turn on A/D & ADIE
    movlw     ADTABLE          ; point fsr to top of...
    movwf    FSR              ; table
;
new_ad
    bsf        ADCON0, 2        ; start new A/D conversion
    sleep      ; goto sleep
; when A/D is over program will continue from here
;
    movf      ADRES, w         ; get A/D value
;
```

A detailed code listing is in Appendix D.

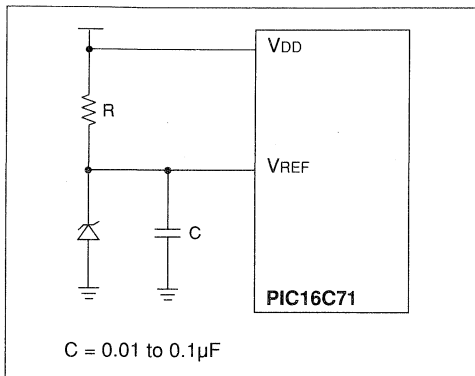
Using the Analog to Digital Converter

USING EXTERNAL REFERENCE VOLTAGE

When using external reference voltage, keep in mind that any analog input voltage must not exceed V_{REF} .

An inexpensive way to generate V_{REF} is by employing zener diode (Figure 9). Most common zener diodes offer 5% accuracy. Reverse bias current may be as low as $10\ \mu\text{A}$. However, larger currents ($1\text{mA} - 20\text{mA}$) are recommended for stability, as well as lower impedance of the V_{REF} source.

FIGURE 9 - LOW COST VOLTAGE REFERENCE



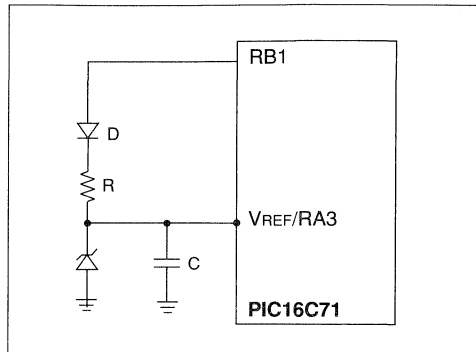
POWER MANAGEMENT IN USING V_{REF}

In power sensitive applications, user may turn on V_{REF} generator using another I/O pin as shown in Figure 10. Drive a "1" on RB1 pin in this example when using the A/D. Drive a "0" on RB1 pin when not using the A/D converter.

Note that this way RB1 is not floating. Even if V_{REF} decays to some intermediate voltage, it will not cause the input buffer on RB1 to draw current.

Alternately, use RA0, RA1 or RA2 pin to supply the current instead of RB1. Configure the RA pin as analog (this will turn off its input buffer). Then use it as a digital output (Figure 11).

FIGURE 10 - POWER-SENSITIVE APPLICATIONS #1



ZENERS AND REFERENCE GENERATORS

Finally, various reference voltage generator chips (typically using on-chip band-gap reference) are available. These are more accurate.

TABLE 1 - ZENERS AND REFERENCE GENERATORS

Zeners	V_z	Tolerance
1N746	3.3V	$\pm 5\%$
1N747	3.6V	$\pm 5\%$
1N748	3.9V	$\pm 5\%$
1N749	4.3V	$\pm 5\%$
1N750	4.7V	$\pm 5\%$
1N751	5.1V	$\pm 5\%$
1N752	5.6V	$\pm 5\%$
Voltage References	V_{REF}	Tolerance
AD580 (Maxim)	2.5V	$\pm 3\%$ to $\pm 0.4\%$
LM385	2.5V	$\pm 1.5\%$
LM1004	2.5V	$\pm 1.2\%$
LT1009 (LIN. Tech.)	2.5V	$\pm 0.2\%$
LT1019 (LIN. Tech.)	5.0V	$\pm 0.2\%$
LT1021 (LIN. Tech.)	5.0V	$\pm 0.05\%$ to $\pm 1\%$
LT1029 (LIN. Tech.)	5.0V	$\pm 0.2\%$ to $\pm 1\%$

Using the Analog to Digital Converter

VREF IMPEDANCE AND CURRENT SUPPLY REQUIREMENTS

Ideally, VREF should have as low a source impedance as possible. Referring to Figure 9, VREF source impedance $\approx R$. However, smaller R increases current consumption. Since VREF is used to charge capacitor arrays inside the A/D converter and the holding capacitor $C_{HOLD} \approx 51\text{pF}$, the following guideline should be met:

$$t_{ad} = 6(1\text{K} + R) 51.2\text{ pF} + 1.677\mu\text{s}$$

t_{ad} = conversion clock. For $t_{ad} = 2\mu\text{s}$ and for $C_{HOLD} = 50\text{ pF}$, $R_{VREF} \approx 50\Omega$.

For VREF impedance higher than this, the conversion clock (t_{ad}) should be increased appropriately.

FIGURE 11 - POWER-SENSITIVE APPLICATIONS #2

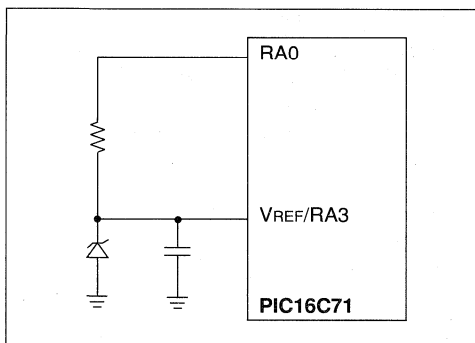


Table 2 gives examples of the maximum rate of conversion per bit, relating to the voltage reference impedance.

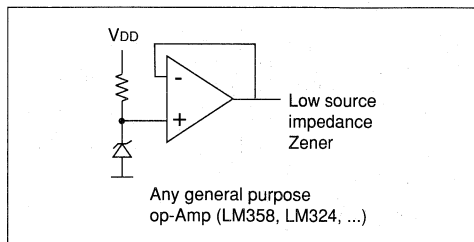
TABLE 2 - MAXIMUM RATE OF CONVERSION / BIT

R_{VREF}	$T_{ad} \text{ (Max)}$
1K	2.29 μs
5K	3.52 μs
10K	5.056 μs
50K	16.66 μs
100K	32.70 μs

Assumes no external capacitors.

To achieve a low source impedance when using a Zener diode, a voltage follower circuit is recommended. This is shown in Figure 11A.

FIGURE 11A - VOLTAGE FOLLOWER CIRCUIT



CONFIGURING PORT A INPUTS AS ANALOG OR DIGITAL

Two bits in ADCON1 register PCFG1 and PCFG0 control how pins RA0–RA3 are configured. When any of these pins are selected as analog:

- The digital input buffer is turned off to save current (see Figure 12). Reading the port will read this pin as '0'.
- TRIS bit still controls the output buffer on this pin. So, normally the TRIS bit will be set (input).
- However, if the TRIS bit is cleared, then the pin will output whatever is in the data latch.

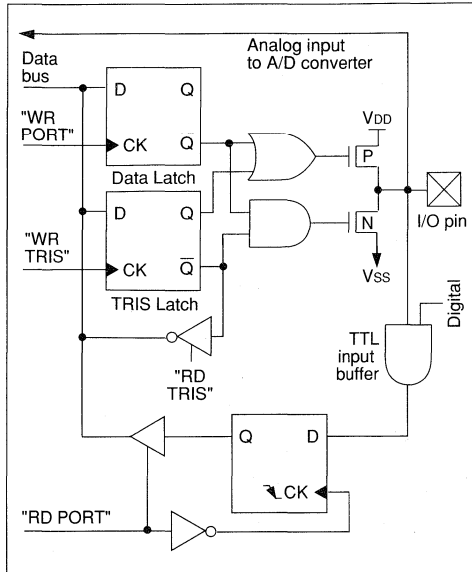
When any of these pins are selected as digital:

- The analog input still directly connects to the A/D and therefore the pin can be used as analog input.
- The digital input buffer is not disabled.

The user has, therefore, great flexibility in configuring these pins.

Using the Analog to Digital Converter

FIGURE 12 - BLOCK DIAGRAM OF RA0-RA3 PINS



CURRENT CONSUMPTION THROUGH INPUT BUFFER

A CMOS input buffer will draw current when the input voltage is around its threshold. (See Figure 13.)

In power-sensitive applications, the RA pins when used as analog inputs should be configured as "analog" to avoid unintended power drain.

Other considerations and tips:

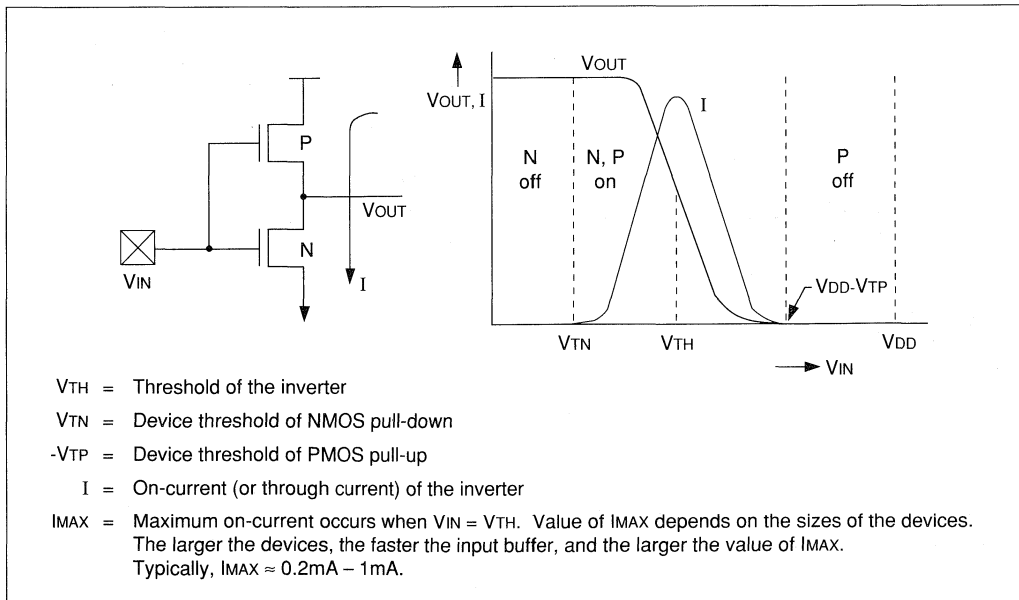
1. If possible, avoid any digital output next to analog inputs.
2. Avoid digital inputs that switch frequently (e.g., clocks) next to analog inputs.
3. If VREF is used, then no analog pin being sampled should exceed VREF.

SUMMARY

The PIC16C71 A/D converter is simple to use. It is versatile and low power.

AUTHORS: Sumit Mitra, Stan D'Souza, Russ Cooper, Logic Products Division

FIGURE 13 - A SIMPLE CMOS INPUT BUFFER:



Using the Analog to Digital Converter

APPENDIX A - SINGLE CHANNEL A/D (SAD)

MPASM B0.44

PAGE 1

```
;TITLE "Single channel A/D (SAD)"
;This program is a simple implementation of the PIC16C71's
;A/D. 1 Channel is selected (CH0).
;The A/D is configured as follows:
; Vref = +5V internal.
; A/D Osc. = internal RC
; A/D Channel = CH0
;Hardware: PICDEMO board.
; Stan D'Souza 7/6/93
;
; LIST P=16C71,F=INHX8M
; include "picreg.equ"
;
0010 TEMP equ 10h
0001 adif equ 1
0002 adgo equ 2
;
; org 0x00
;
;
0000 2810 goto start
;
; org 0x04
0004 281C goto service_int ;interrupt vector
;
;
; org 0x10
start
0010 3000 movlw B'00000000' ;set port b as
0011 0086 movwf PORT_B ;all outputs
0012 0066 tris PORT_B ; /
;
0013 201D call InitializeAD
update
0014 0809 movf ADRES,W ;get A/D value
0015 0086 movwf PORT_B ;output to port b
0016 2025 call SetupDelay ;setup time >= 10uS.
0017 1088 bcf ADCON0,adif ;clear int flag
0018 1508 bsf ADCON0,adgo ;start new conversion
;
loop
0019 1888 btfsc ADCON0,adif ;A/D done?
001A 2814 goto update ;yes then update new value.
001B 2819 goto loop ;no then keep checking
;
;no interrupts are enabled, so if the program reaches here,
;it should be returned with the global interrupts disabled.
service_int
001C 0008 return ;do not enable global.
;
```

Using the Analog to Digital Converter

MPASM B0.44

PAGE 2

```
                ;InitializeAD, initializes and sets up the A/D hardware.
                ;Select ch0 to ch3 as analog inputs, fosc/2 and read ch0.
InitializeAD
001D 1683        bsf     STATUS,5           ;select pg1
001E 3000        movlw   B'00000000'       ;select ch0-ch3...
001F 0108        movwf   ADCON1           ;as analog inputs
0020 1283        bcf     STATUS,5           ;select pg0
0021 30C1        movlw   B'11000001'       ;select:RC,ch0..
0022 0088        movwf   ADCON0           ;turn on A/D.
0023 0189        clrf   ADRES             ;clr result reg.
0024 0008        return

                ;
                ;This routine is a software delay of 10uS for the a/d setup.
                ;At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
                ;a value of 3 to give 9uS, plus the move etc should result in
                ;a total time of > 10uS.
SetupDelay
0025 3003        movlw   .3
0026 0090        movwf   TEMP

                SD
0027 0B90        decfsz  TEMP
0028 2827        goto   SD
0029 0008        return
                END

Errors   :    0
Warnings :    0
```

Using the Analog to Digital Converter

APPENDIX B

MPASM B0.44

PAGE 1

```
;TITLE "A/D in Sleep Mode"
;This program is a simple implementation of the PIC16C71's
;A/D feature. This program demonstrates
;how to do a A/D in sleep mode on the PIC16C71.
;The A/D is configured as follows:
;   Vref = +5V internal.
;   A/D Osc. = internal RC
;   A/D Interrupt = OFF
;   A/D Channels = ch 0
;
;The ch0 A/D result is displayed as a 8-bit binary value
;on 8 LEDs connected to port b.
;Hardware: PICDEMO board.
;   Stan D'Souza 7/6/93
;
;   LIST P=16C71,F=INHX8M
;
;   include "picreg.equ"
;
0010      equ      10h
0001      adif     equ      1
0002      adgo     equ      2
;
;
;   org      0x00
;
0000 2810      goto    start
;
;   org      0x04
0004 281B      goto    service_int ;interrupt vector
;
;
;   org      0x10
start
0010 3000      movlw   B'00000000' ;make port b all
0011 0086      movwf  PORT_B ;outputs.
0012 0066      tris   PORT_B ; /
;
0013 201C      call   InitializeAD
update
0014 0809      movf   ADRES,W
0015 0086      movwf  PORT_B ;save in table
0016 2025      call   SetupDelay ;
0017 1088      bcf   ADCON0,adif ;clr A/D flag
0018 1508      bsf   ADCON0,adgo ;start new A/D conversion
;
0019 0063      sleep
```

3

Using the Analog to Digital Converter

MPASM B0.44

PAGE 2

```
001A 2814          goto    update      ;wake up and update
;
;service_int
001B 0008          return           ;do not enable int
;
;InitializeAD, initializes and sets up the A/D hardware.
InitializeAD
001C 1683          bsf     STATUS,5      ;select pg1
001D 3000          movlw  B'00000000'    ;select ch0-ch3...
001E 0108          movwf  ADCON1         ;as analog inputs
001F 1283          bcf     STATUS,5      ;select pg0
0020 30C1          movlw  B'11000001'    ;select:internal RC, ch0.
0021 0088          movwf  ADCON0         ;turn on A/D
0022 018B          clrf   INTCON         ;clear all interrupts
0023 170B          bsf    INTCON,ADIE    ;enable A/D
0024 0008          return
;
;This routine is a software delay of 10uS for the A/D setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.
SetupDelay
0025 3003          movlw  .3
0026 0090          movwf  TEMP
SD
0027 0B90          decfsz TEMP
0028 2827          goto  SD
0029 0008          return
;
END

Errors   :    0
Warnings :    0
```


Using the Analog to Digital Converter

APPENDIX C

MPASM B0.44

PAGE 1

```
;TITLE      "Single channel A/D with interrupts"
;This program is a simple implementation of the PIC16C71's
;A/D. 1 Channel is selected (CH0). A/D interrupt is turned on,
;hence on completion of A/D conversion, an interrupt is generated.
;The A/D is configured as follows:
;          Vref = +5V internal.
;          A/D Osc. = internal RC Osc.
;          A/D Interrupt = On
;          A/D Channel = CH0
;
;The A/D result is displayed as a 8-bit value on 8 LEDs connected
;to portb.
;Hardware: PICDEMO board.
;          Stan D'Souza 7/6/93.
;
LIST P=16C71,F=INHX8M
;
include "picreg.equ"
;
0010      flag      equ      10
0011      TEMP      equ      11
0000      adover    equ      0
0001      adif      equ      1
0002      adgo      equ      2
0006      adie      equ      6
0007      gie       equ      7
0005      rp0       equ      5
;
;          org      0x00
0000 2810      goto    start
;
;          org      0x04
0004 281C      goto    service_ad      ;interrupt vector
;
;          org      0x10
start
0010 3000      movlw   B'00000000'      ;init I/O ports
0011 0086      movwf  PORT_B
0012 0066      tris   PORT_B
;
0013 2022      call   InitializeAD
update
0014 1010      bcf    flag,adover      ;reset software A/D flag
0015 202B      call   SetupDelay      ;setup delay >= 10uS.
0016 1088      bcf    ADCON0,adif      ;reset A/D int flag (ADIF)
0017 1508      bsf    ADCON0,adgo      ;start new A/D conversion
0018 178B      bsf    INTCON,gie      ;enable global interrupt
loop
0019 1810      btfsc  flag,adover      ;A/D over?
001A 2814      goto   update          ;yes start new conv.
001B 2819      goto   loop           ;no then keep checking
```

Using the Analog to Digital Converter

MPASM B0.44

PAGE 2

```

;
service_ad
001C 1C88      btfss  ADCON0,adif  ;A/D interrupt?
001D 0009      retfie                ;no then ignore
001E 0809      movf   ADRES,W        ;get A/D value
001F 0086      movwf  PORT_B        ;output to port b
0020 1410      bsf    flag,adover  ;A/D done set
0021 0008      return                ;do not enable int
;
;
;InitializeAD, initializes and sets up the A/D hardware.
;select ch0, RC OSC., A/D interrupt.
InitializeAD
0022 1683      bsf    STATUS,rp0      ;select pg1
0023 3000      movlw  B'00000000'  ;select ch0-ch3...
0024 0108      movwf  ADCON1        ;as analog inputs
0025 1283      bcf    STATUS,rp0      ;select pg0
0026 018B      clrf   INTCON        ;clr all interrupts
0027 170B      bsf    INTCON,adie    ;enable A/D int.
0028 30C1      movlw  B'11000001'  ;select:RC osc,ch0...
0029 0088      movwf  ADCON0        ;turn on A/D
002A 0008      return
;
;This routine is a software delay of 10uS for the A/D setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.
SetupDelay
002B 3003      movlw  .3
002C 0091      movwf  TEMP
;
SD
002D 0B91      decfsz TEMP
002E 282D      goto   SD
002F 0008      return
;
;
END

Errors      :    0
Warnings    :    0
```

Using the Analog to Digital Converter

APPENDIX D

MPASM B0.44

PAGE 1

```
;TITLE      "A/D using Multiple Channels"
;This program is a simple implementation of the PIC16C71's
;A/D feature. This program demonstrates
;how to select multiple channels on the PIC16C71.
;The A/D is configured as follows:
;      Vref = +5V internal.
;      A/D Osc. = internal RC osc.
;      A/D Interrupt = Off
;      A/D Channels = all in a "Round Robin" format.
;      A/D results are stored in ram locations as follows:
;      ch0 -> ADTABLE + 0
;      ch1 -> ADTABLE + 1
;      ch2 -> ADTABLE + 2
;      ch3 -> ADTABLE + 3
;
;The ch0 A/D result is displayed as a 8-bit value on 8 LEDs
;connected to port b.
;Hardware: PICDEMO board.
;          Stan D'Souza 7/6/93.
;
;          LIST P=16C71,F=INHX8M
;
;          include "picreg.equ"
;
0010      TEMP      equ      10h
0001      adif      equ      1
0002      adgo      equ      2
;
0006      ch2       equ      6
0007      ch3       equ      7
000C      flag      equ      0C
0020      ADTABLE   equ      20
;
;          org      0x00
;
;
0000 2810      goto      start
;
;          org      0x04
0004 2823      goto      service_int      ;interrupt vector
;
;          org      0x10
start
0010 3000      movlw   B'00000000'      ;make port b
0011 0086      movwf   PORT_B          ;as all outputs
0012 0066      tris    PORT_B          ;
;
0013 2024      call    InitializeAD
```

Using the Analog to Digital Converter

MPASM B0.44

PAGE 2

```
update
0014 0809      movf  ADRES,W
0015 0080      movwf  0           ;save in table
0016 3020      movlw  ADTABLE      ;chk if ch0
0017 0204      subwf  FSR,W       ; /
0018 1D03      btfss  STATUS,Z     ;yes then skip
0019 281C      goto   NextAd      ;else do next channel
001A 0809      movf  ADRES,W     ;get A/D value
001B 0086      movwf  PORT_B      ;output to port b

NextAd
001C 202E      call   NextChannel ;select next channel
001D 203A      call   SetupDelay  ;set up > = 10uS
001E 1088      bcf   ADCON0,adif   ;clear flag
001F 1508      bsf   ADCON0,adgo   ;start new A/D conversion

loop
0020 1888      btfsc  ADCON0,adif   ;A/D done?
0021 2814      goto   update      ;yes then update
0022 2820      goto   loop        ;wait till done

;
service_int
0023 0008      return           ;do not enable int

;
;InitializeAD, initializes and sets up the A/D hardware.
InitializeAD
0024 1683      bsf   STATUS,5       ;select pg1
0025 3000      movlw  B'00000000'   ;select ch0-ch3...
0026 0108      movwf  ADCON1      ;as analog inputs
0027 1283      bcf   STATUS,5       ;select pg0
0028 30C1      movlw  B'11000001'   ;select:fosc/2, ch0.
0029 0088      movwf  ADCON0      ;turn on A/D
002A 3020      movlw  ADTABLE      ;get top of table address
002B 0084      movwf  FSR         ;load into indirect reg
002C 0189      clrf  ADRES        ;clr result reg.
002D 0008      return

;
```

Using the Analog to Digital Converter

MPASM B0.44

PAGE 3

```

;NextChannel, selects the next channel to be sampled in a
;"round-robin" format.
NextChannel
    movlw 0x08          ;get channel offset
    addwf ADCON0       ;add to conf. reg.
002E 3008
002F 0788
0030 1288             bcf  ADCON0,5       ;clear any carry over
;increment pointer to correct A/D result register
    clrf  TEMP
0031 0190
0032 1988             btfsc ADCON0,3      ;test lsb of chnl select
0033 1410             bsf  TEMP,0          ;set if ch1 or ch3
0034 1A08             btfsc ADCON0,4      ;test msb of chnl select
0035 1490             bsf  TEMP,1          ;set if ch0 or ch2
0036 3020             movlw ADTABLE       ;get top of table
0037 0710             addwf TEMP,W        ;add with temp
0038 0084             movwf FSR          ;allocate new address
0039 0008             return
;
;This routine is a software delay of 10uS for the A/D setup.
;At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
;a value of 3 to give 9uS, plus the move etc should result in
;a total time of > 10uS.
SetupDelay
    movlw .3
003A 3003
003B 0090             movwf TEMP
SD
003C 0B90             decfsz TEMP
003D 283C             goto  SD
003E 0008             return
;
END

Errors : 0
Warnings : 0
```

3

Using the Analog to Digital Converter

Implementing Wake-up on Key Stroke

INTRODUCTION

Microchip's PIC16CXX family of microcontrollers are ideally suited to directly interface to a keypad. The high 4 bits of PortB (RB4 - RB7) have internal pull-ups and can trigger a "change on port state" interrupt. This interrupt, if enabled, will wake the microcontroller from sleep. In most battery powered applications a microcontroller is exercised when a key is pressed, e.g. in a remote keyless entry system. The life of the battery can be extended by using PIC16CXX microcontrollers. This can be done by putting the PIC16CXX microcontroller into sleep mode for most of the time and wake-up only when a key is pressed.

IMPLEMENTATION

Figure 1 depicts an application where four keys are connected to RB4 - RB7. Internal pull-ups are used to maintain a high level on these inputs. In this example, LEDs are connected to RB0 - RB3. When SW1 is pressed, LED1 is turned on and when SW2 is pressed, LED2 is turned on and so on. The PIC16CXX is normally in sleep mode with the "change on port state" interrupt enabled. When SW1 is pressed, RB4 goes low and triggers an interrupt. Since the PIC16CXX is in sleep, it first wakes up and starts executing code at the interrupt vector. Note that if the global interrupt is enabled, the program execution after an interrupt is at the interrupt vector, if the global interrupt is not enabled, the program starts executing right after the sleep instruction.

After waking up, a 20 - 40 msec. de-bounce delay is executed which checks the port for a key hit and depending on which key is hit, its associated LED is turned on. The LEDs are used purely for demonstration purposes. In a remote keyless entry application, the remote code would be transmitted when the appropriate key is hit.

Figure 2 depicts a 4x4 keypad interface to the PIC16CXX. When using the PIC16CXX in a keypad application, the internal pull-ups on RB4 - RB7 can be enabled eliminating the need for external pull-up resistors. The series 100Ω resistors are used for ESD protection, and are recommended in keypad interface applications.

AUTHOR: Stan D'Souza, Logic Products Division

SUMMARY

The PIC16CXX is ideally suited to interface directly to a Keypad application. Built in pull-up resistors and very low sleep current make it a very good candidate for battery powered remote operations and applications.

Performance:

Code Size: 64 words

RAM Locations Used: 0 bytes

FIGURE 1 - 4 KEY INTERFACE TO PIC16CXX

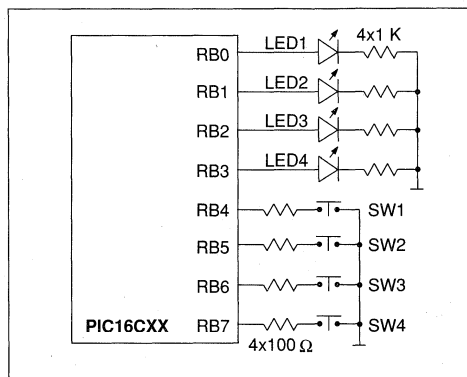
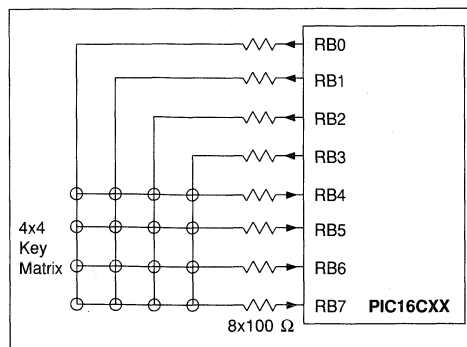


FIGURE 2 - 4X4 KEYPAD INTERFACE TO PIC16CXX



Implementing Wake-up on Key Stroke

16c5x/XX Cross-Assembler V3.05.06 BETA Wed Apr 28 16:09:44 1993 Page 1

```
Line   PC   Opcode

0001           ;This program demonstrates the wake-up on Keystroke feature of the
0002           ;PIC16C71. Port B pins RB4 - RB7 can be configured as inputs with internal
0003           ;pull up resistors, also the interrupt associated with the change on input
0004           ;on RP4 - RB7 can be set up to wake the chip from sleep. If the
0005           ;global interrupt is enabled just before sleep, the program will vector to
0006           ;the interrupt vector (0004). If not the chip will continue execution
0007           ;just after the next instruction following slpccp.
0008           ;In this example code, the port B is initialized to input 4 keys at
0009           ;RB4 - RB7. RB0 - RB3 are configured to drive LEDs corresponding to
0010           ;which key is hit (LED on RB0 when RB4 is hit and so on).
0011           ;Sleep is executed. When any keys is hit, the processor wakes
0012           ;up and jumps to the interrupt vector. The corresponding LED is
0013           ;turned on and after the key is released, the whole process is repeated.
0014           ;
0015           LIST P=16C71, F=INHX8M
0016           ;
0017           0002   z           equ     2
0018           0007   RBPU       equ     7
0019           0010   temp       equ     10h
0020           0001   OptionReg  equ     1h
0021           include "picreg.   equ"
0022           0000           LIST L=ON
0023
0024
0025           ;
0026           0000   org     0
0027           0000   2805   goto   start
0028           ;
0029           0000   org     4
0030           0004   2808   goto   ServiceInterrupt
0031           ;
0032           ;
0033           start
0034           0005   2024   call   InitPortB   ;initialize port B
0035           loop
0036           0006   0063   sleep          ;sleep till key is hit
0037           ;nop
0038           0007   2806   goto   loop
0039           ;
0040           ServiceInterrupt
0041           0008   180B   btfsc  INTCON,RBIF ;change on rb int?
0042           0009   280D   goto   ServiceWakup ;yes then service
0043           000A   128B   bcf   INTCON,RTIE ;clear RTCC int mask
0044           000B   110B   bcf   INTCON,RTIF ;clear flag
0045           000C   0008   return
```


Implementing Wake-up on Key Stroke

16c5x/XX Cross-Assembler V3.05.06 BETA Wed Apr 28 16:09:44 1993 Page 2

```
Line  PC      Opcode
0046          ;
0047          ;This routine checks which keys is hit and lights up the
0048          ;corresponding LED associated with it. eg. RB0's LED when
0049          ;RB4's key is pressed. Finally it waits till all keys have
0050          ;been released before returning form the service routine.
0051          ServiceWakup
0052  000D  118B      bcf     INTCON,RBIE      ;clear mask
0053  000E  0906      comf   PORT_B,w         ;read PORT_B
0054  000F  100B      bcf     INTCON,RBIF      ;clear flag
0055  0010  2035      call   delay16          ;do de-bounce for 16mSecs
0056  0011  0906      comf   PORT_B,w         ;read port B again
0057  0012  39F0      andlw  B'11110000'      ;mask outputs
0058  0013  0090      movwf  temp              ;save in temp
0059  0014  0E10      swapf  temp,w           ;switch low and high
0060  0015  0086      movwf  PORT_B           ;send as outputs.
0061  0016  2018      call   KeyRelease       ;check for key release
0062  0017  0009      retfie
0063          ;
0064          ;This sub-routine, waits till all key have been released
0065          ;In order to save power, the chip is in sleep mode till
0066          ;all keys are released.
0067          KeyRelease
0068  0018  2035      call   delay16          ;do debounce
0069  0019  0906      comf   PORT_B,w         ;read PORT_B
0070  001A  100B      bcf     INTCON,RBIF      ;clear flag
0071  001B  158B      bsf     INTCON,RBIE      ;enable mask
0072  001C  39F0      andlw  B'11110000'      ;clear outputs
0073  001D  1903      btfs   STATUS,z         ;key still pressed?
0074  001E  0008      return
0075  001F  0063      sleep
0076  0020  118B      bcf     INTCON,RBIE      ;on wake up clear mask
0077  0021  0906      comf   PORT_B,w         ;read PORT_B
0078  0022  100B      bcf     INTCON,RBIF      ;clear flag
0079  0023  2818      goto   KeyRelease       ;try again
0080          ;
0081          ;
0082          ;This sub-routine, initializes PortB.
0083          InitPortB
0084  0024  1683      bsf     STATUS,RP0      ;select bank 1
0085  0025  3003      movlw  B'00000011'      ;Port_A digital I/O
0086  0026  0088      movwf  ADCON1           ;
0087  0027  3000      movlw  0                 ;
0088  0028  0085      movwf  PORT_A           ;set port a as outputs
0089  0029  30F0      movlw  B'11110000'      ;RB0-RB3 outputs
0090  002A  0086      movwf  PORT_B           ;RB4-RB7 inputs
0091  002B  1381      bcf     OptionReg,RBPU  ;enable pull up
0092  002C  1283      bcf     STATUS,RP0      ;select page 0
0093  002D  0186      clrf   PORT_B           ;init port B
0094  002E  0185      clrf   PORT_A           ;make port a all low
0095  002F  1405      bsf     PORT_A,0        ;make first bit high
0096  0030  118B      bcf     INTCON,RBIE      ;disable mask
0097  0031  0806      movf   PORT_B,w         ;read port
0098  0032  100B      bcf     INTCON,RBIF      ;clear flag
0099  0033  158B      bsf     INTCON,RBIE      ;enable mask
0100  0034  0009      retfie
0101          ;
```

Implementing Wake-up on Key Stroke

16c5x/XX Cross-Assembler V3.05.06 BETA Wed Apr 28 16:09:44 1993 Page 3

```
Line   PC   Opcode
0102           ;delay16 waits for approx 16.4mSecs using RTCC interrupts
0103           ;fosc speed is 4Mhz.
0104           delay16
0105   0035 1683           bsf   STATUS,RP0   ;select page 1
0106   0036 3007           movlw B'00000111' ;fosc/256 -> RTCC
0107   0037 0081           movwf OptionReg   ;
0108   0038 1283           bcf   STATUS,RP0   ;select page 0
0109   0039 0181           clrf  RTCC
0110   003A 110B           bcf   INTCON,RTIF  ;clear flag
0111   003B 168B           bsf   INTCON,RTIE  ;enable mask
0112           CheckAgain
0113   003C 1D0B           btfss INTCON,RTIF ;timer overflowed?
0114   003D 283C           goto  CheckAgain  ;no check again
0115   003E 128B           bcf   INTCON,RTIE ;else clear mask
0116   003F 110B           bcf   INTCON,RTIF ;clear flag
0117   0040 0008           return
0118           ;
0119           0000           end
```



AN566

Using the PortB Interrupt on Change as an External Interrupt

INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. To address these applications, there is the PIC16CXX microcontroller family of products. This family has numerous peripheral and special features to better address user applications.

One feature is the interrupt on change of the PORTB pins. This "interrupt on change" is caused when any of the RB<7:4> pin, configured as input, changes levels. When used in conjunction with the software programmable weak internal pull-ups, a direct interface to a keypad is possible. This is shown in application note AN552 (Implementing Wake-up on Key Stroke). Another way to use the "interrupt on change" feature would be as additional external interrupt sources. This allows the PIC16CXX devices to support multiple external interrupts, in addition to the INT pin.

This application note will discuss some of the issues in using PortB as additional external interrupt pins, and will show some examples. These examples can be easily modified to suit your particular needs.

USING A PORTB INPUT FOR AN EXTERNAL INTERRUPT

The interrupt source(s) cannot simply be directly connected to the PortB pins, and expect the interrupt to function the same as the interrupt (INT) pin. The characteristic of the interrupt signal must also be known to develop the microcontrollers hardware/software. After we know this, we can determine the best way to structure the program to handle this signal. These characteristics include:

1. Trigger interrupt on rising, falling, or both edges
2. What is the pulse width of the interrupt (high time / low time)

It is easy to understand the need of knowing which edge triggers the external interrupt service routine. This allows one to ensure that the interrupt service routine is only entered for the desired edge, with all other edges ignored. Not so clear is pulse width of the interrupt. This determines the amount of additional overhead that the software routine may need.

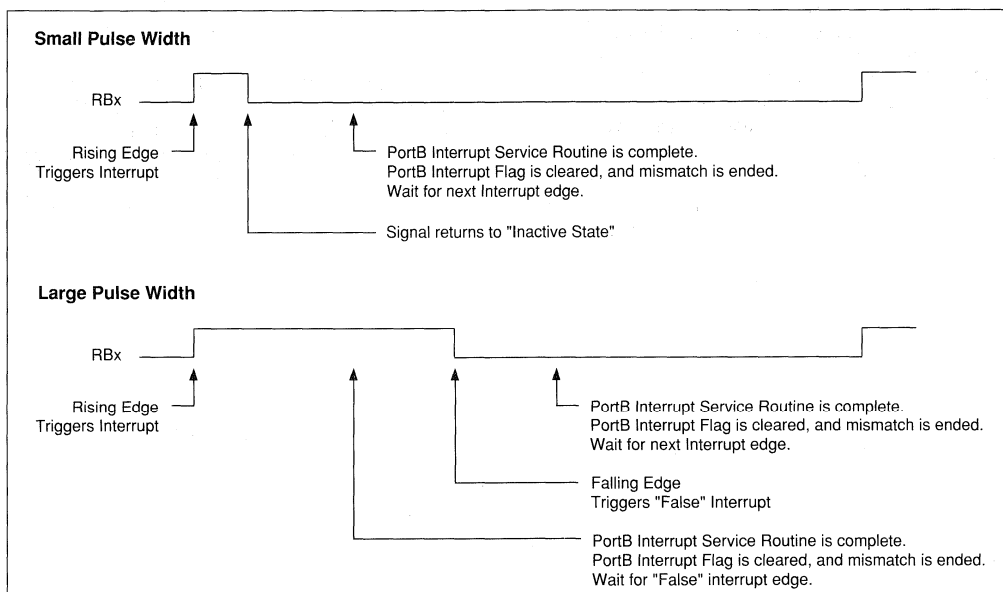
PortB as External Interrupt

Figure 1 shows the two cases for the interrupt signal versus the time to complete the interrupt service routine. The first waveform is when the signal makes the low-to-high-to-low transitions before the interrupt service routine has completed (interrupt flag cleared). When the interrupt flag has been cleared the interrupt signal has already returned to the inactive level. The next transition of the signal is due to another interrupt request. An interrupt signal with this characteristic will be called a small pulse width signal. The second waveform is when the signal only makes the low-to-high transitions before the interrupt service routine has completed (interrupt flag cleared). The next transition (high-to-low) will return the interrupt signal to the inactive level. This will generate a "false" interrupt, that will need to be cleared. Then the following transition (low-to-high) will be a "true" interrupt. An interrupt signal with this characteristic will be called a wide pulse width signal.

An interrupt pulse with a small pulse width requires less overhead than a wide pulse width. A small pulse width signal must be less than the minimum execution time of the interrupt service routine, while a wide pulse width must be greater than the maximum time through the interrupt service routine.

Example 1 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a small pulse width. In this case since the interrupt pulse width is small, the pulse has gone high and then low again before PortB is read to end the mismatch condition. So when PortB is read it will read a low signal and will again be waiting for the rising edge transition.

FIGURE 1 - INTERRUPT STEPS FOR SMALL AND WIDE PULSE WIDTHS



Example 1: Single Interrupt with a Small Pulse Width

```
PER_INT BTFFS  INTCON, RBIF          ; PortB interrupt?
                GOTO  OTHER_INT       ; Other interrupt
                :                      ; Do task for INT on RB7
                :                      ;
CLR_RBINTF    MOVF  PORTB, 1          ; Read PortB (to itself) to end
                :                      ; mismatch condition
                BCF  INTCON, RBIF     ; Clear the RB interrupt flag.
                RETFIE                ; Return from interrupt
OTHER_INT     :                      ; Do what you need to here
                :
                RETFIE                ; Return from interrupt
```

PortB as External Interrupt

Example 2 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a wide pulse width. In this case since the interrupt pulse width is large, the pulse is still high before PortB is read to end the mismatch condition. So when PortB is read it will read a high signal and will generate an interrupt on the next falling edge transition (which should be ignored).

Example 2: Single Interrupt with a Wide Pulse Width

```
PER_INT BTFSS  INTCON, RBIF          ; PortB interrupt?
                GOTO  OTHER_INT      ; Other interrupt
                BTFSC  PORTB, RB7    ; Check for rising edge
                GOTO  CLR_RBINTF     ; Falling edge, clear PortB int
                :                      ; flag
                :                      ; Do task for INT on RB7
                :
CLR_RBINTF     MOVF  PORTB, 1          ; Read PortB (to itself) to end
                :                      ; mismatch condition
                BCF   INTCON, RBIF    ; Clear the RB interrupt flag.
                RETFIE                ; Return from interrupt
OTHER_INT      :                      ; Do what you need to here
                :
                RETFIE                ; Return from interrupt
```

Example 3: Interrupt on Change

```
PER_INT BTFSS  INTCON, RBIF          ; PortB interrupt?
                GOTO  OTHER_INT      ; Other interrupt
CLR_RBINTF     MOVF  PORTB, 1          ; Read PortB (to itself) to end
                :                      ; mismatch condition
                BCF   INTCON, RBIF    ; Clear the RB interrupt flag.
                :                      ; Do task for INT on RB7
                :                      ;
                RETFIE                ; Return from interrupt
OTHER_INT      :                      ; Do what you need to here
                :
                RETFIE                ; Return from interrupt
```

PortB as External Interrupt

Using PortB Inputs for Multiple Interrupts

The previous examples have been for a single external interrupt on PORTB. This can be extended to support up to 4 external interrupts. To do this requires additional software overhead, to determine which of the PortB pins (RB<7:4>) caused the interrupt. Care should be taken in the software to ensure that no interrupts are lost.

In this example, the interrupt sources on RB7, RB5, and RB4 have a small pulse width, while the interrupt source on pin RB6 is wide and should cause a trigger on the rising edge.

SUMMARY

The PortB interrupt on change feature is both a very convenient method for direct interfacing to an external keypad, with no additional components, but is also versatile in its uses. The ability to add up to four additional external interrupt. Of course hybrid solutions are also possible. That is, for example, using PORTB<6:1> as a 3x3 keypad, with PORTB7 as an external interrupt and PORTB0 as a general purpose I/O. The flexibility of this feature allows the user to implement a best fit design for the application.

Example 4: Multiple Interrupts with Different Pulse Widths

```
PER_INT BTFSS  INTCON, RBIF          ; PortB interrupt?
                GOTO  OTHER_INT      ; Other interrupt
;
; PortB change interrupt has occurred. Must determine which pin caused
; interrupt and do appropriate action. That is service the interrupt,
; or clear flags due to other edge.
;
                MOVF   PORTB, 0      ; Move PortB value to the W register
                ; This ends mismatch conditions
                MOVWF  TEMP          ; Need to save the PortB reading.
                XORWF  LASTPB, 1     ; XOR last PortB value with the new
                ; PortB value.
CK_RB7         BTFSC  LASTPB, RB7    ; Did pin RB7 change
                CALL  RB7_CHG       ; RB7 changed and caused the interrupt
CK_RB6         BTFSC  LASTPB, RB6    ; Did pin RB6 change
                CALL  RB6_CHG       ; RB6 changed and caused the interrupt
CK_RB5         BTFSC  LASTPB, RB5    ; Did pin RB5 change
                CALL  RB5_CHG       ; RB5 changed and caused the interrupt
CK_RB4         BTFSC  LASTPB, RB4    ; Did pin RB4 change
                GOTO  RB4_CHG       ; RB4 changed and caused the interrupt
;
RB7_CHG        :                    ; Do task for INT on RB7
                :                    ;
                RETURN
RB6_CHG BTFSC  PORTB, RB6            ; Check for rising edge
                RETURN              ; Falling edge, ignore
                :                    ; Do task for INT on RB6
                :                    ;
                RETURN
RB5_CHG        :                    ; Do task for INT on RB5
                :                    ;
                RETURN
RB4_CHG        :                    ; Do task for INT on RB4
                :                    ;
CLR_RBINTF     MOVF   TEMP, 0        ; Move the PortB read value to the
                MOVWF LASTPB        ; register LASTPB
                BCF   INTCON, RBIF   ; Clear the RB interrupt flag.
                RETFIE              ; Return from interrupt
;
OTHER_INT      :                    ; Do what you need to here
                :                    ;
                RETFIE              ; Return from interrupt
```

AUTHOR: Mark Palmer, Logic Products Division

Table Read

INTRODUCTION

To access data in Program Memory, a table read operation must be performed. The Table consists of a series of `RETLW K` instructions where, the 8 bit table constants are assigned to the literal K. The first instruction in the Table computes the offset to the table by using "addwf pcl" and consequently the program branches to the appropriate `retlw X` instruction (see Example 1).

Example 1.

```

.
.
movlw  offset ;load offset in w reg
call   Table
.
.
.
Table:
    addwf pcl    ;add offset to pc to
                ;generate a computed goto
    retlw 'A'   ;return the ASCII char A
    retlw 'B'   ;return the ASCII char B
    retlw 'C'   ;return the ASCII char C
.
.
.

```

The method is straight forward, however certain precautions have to be exercised when doing a Table read in the PIC16CXX.

IMPLEMENTATION

Program Counter Loading

The PC in the PIC16CXX is 13 bits wide. The low 8 bits (PCL) are mapped in RAM at location 02 and are directly readable and writable. The high 5 bits are not accessible directly and can only be written through the PCLATH (see Figure 1). The PCLATH is a r/w register with only 5 of its bits implemented <4:0>, all other bits are read as '0'.

FIGURE 1 - LOADING OF PC IN DIFFERENT SITUATIONS

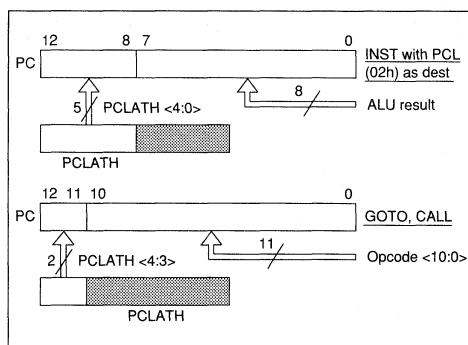


Table Read

SECTION 1

Call and Goto Instructions

When executing a call or goto, the low 11 bits are loaded directly from the instruction opcode. The high 2 bits are loaded from bits 3 and 4 of the PCLATH. It is a good practice to pre-load the PCLATH with the high byte of the location of the routine before executing the routine. This can be done as follows:

Example 2.

```
.  
.  
movlw HIGH Table ;load high 8 bit  
                ;address of Table  
movwf PCLATH    ;into PCLATH  
call Routine    ;execute Call  
                ;instruction  
.  
.
```

Note: If the program memory size is less than 2K bytes, then the above precaution is not necessary.

Computed Goto Instruction

Any instruction with PCL as the destination, will load the PCH with the 5 low bits from the PCLATH (see Fig 1). In Program 3, if the address where the call was made, were on page 0 and the address of the actual table were on say page 3, then on executing the computed goto, the program will goto a location in page 0 instead of a location on page 3. To avoid the program from branching "erratically" when doing a table read, the PCLATH should be pre-loaded with the high byte of the "Table" address. Example 3 shows how this can be done.

Example 3.

```
.  
org 0x80          ;code location in page 0  
movlw offset     ;load offset in w reg  
call Table  
.  
.  
org 0x0320       ;Table located in page 3  
Table  
addwf pcl        ;add offset to pc to  
                ;generate a computed goto  
retlw 'A'        ;return the ASCII char A  
retlw 'B'        ;return the ASCII char B  
retlw 'C'        ;return the ASCII char C  
.  
.  
.
```


When doing a computed goto for a table read, care should be taken about page boundaries. The ADDWF PCL instruction will not compute a value greater than 8 bits. In Example 4, the result of the computed goto will result in a branch to an unintended portion of the code. The user either has to be cautious as to where in a page the Table is resident or will have to monitor page roll-over and add it to the PCLATH ahead of the computed goto.

Example 4.

```

.
org 0x80          ;code location in
                 ;page 0

movlw  HIGH Table ;load PCLATH with hi
                 ;address

movwf  PCLATH    ; /

movlw  offset    ;load offset in w reg
call   Table

.
.
org 0x02ff       ;Table located end of
                 ;page 2

Table:
addwf  pcl      ;value in pc will not
                 ;roll over to page 3

retlw  'A'      ;return the ASCII
                 ;char A

retlw  'B'      ;return the ASCII
                 ;char B

retlw  'C'      ;return the ASCII
                 ;char C

.
.
.

```

To take care of both table location and page boundary crossing, it is necessary to do a 13 bit computed goto operation as shown in Example 5.

The code in Example 5 will allow the user to place and access a table anywhere in program memory.

Example 5

```

.
movlw  LOW Table ;get low 8 bits of
                 ;address

addwf  offset    ;do a 8 bit add
                 ;operation

movlw  HIGH Table ;get high 5 bits of
                 ;address

btfsc  status,c  ;page crossed?

addlw  1         ;yes then inc high
                 ;address

movwf  PCLATH    ;load high address in
                 ;latch

movf   offset,w  ;load computed offset
                 ;in w reg

call   Table

.
.
Table:
movwf  pcl      ;load computed offset
                 ;in PCL

retlw  'A'      ;return the ASCII
                 ;char A

retlw  'B'      ;return the ASCII
                 ;char B

retlw  'C'      ;return the ASCII
                 ;char C

.
.
.

```

Interrupts

Interrupts are handled like a call, i.e. the present PC+1 address is saved on the stack and the program vectors to location 0x04. In example 5 if the interrupt occurs just before the "movwf pcl" instruction (at label "Table"), then the present fetched instruction will be executed, i.e. movwf pcl, the new computed PC will be incremented and saved on stack (as the return from interrupt address) and the program will vector to the interrupt vector. On return from interrupt the program will go to the intended offset of the table + 1. This is a very undesirable result, so interrupts must be disabled during a table read operation.

Table Read

SECTION 2

Differences with PIC16C5X Code

The PIC16C5X has no PCH or PCLATH register, so the user has to take into consideration all the precautions mentioned in Section 1. In PIC16C5X, the location of the Table has to be in the top half of a 512 byte page. This restriction is not valid for the PIC16CXX family. To convert a table read operation from PIC16C5X code to the PIC16CXX code, the following should be done:

1. Remove any program memory bank select instructions (PIC16C56/57) if present. These are not necessary for the PIC16CXX.
2. Do a 13 bit computed goto operation (as shown in Example 5), when doing a table read operation.

Differences with PIC17C42

Unlike the PIC16CXX family, the PIC17C42 loads the PCLATH with the PCH value during a call or goto operation, however a computed goto, does not take care of page boundary crossing. A 16 bit computed jump address should be calculated before the table read is executed. Example 6 show how this is done.

Example 6

```
.
movlw  LOW Table  ;get low 8 bits of
                ;address
addwf  offset    ;do a 8 bit add
                ;operation
movlw  HIGH Table ;get high 8 bits of
                ;address
btfsc  status,c  ;page crossed?
addlw  1         ;yes then inc high
                ;address
movwf  PchBuffer ;load in temp
                ;location
call   Table
.
.
```

Table:

```
movf   PchBuffer,w;get high offset
movwf  PCLATH     ;load in latch
movf   offset,w  ;get low offset
movwf  pcl       ;load computed offset
                ;in PCL
retlw  'A'       ;return the ASCII
                ;char A
retlw  'B'       ;return the ASCII
                ;char B
retlw  'C'       ;return the ASCII
                ;char C
.
.
.
```

Example 6 allows the user to locate his Table at any program memory location, however for large table or tables which cross page boundaries, it is recommended that the "tablrd / tlrd" instruction be used, these instructions are specific for table read operations and are very code efficient.

As mentioned in Section 1, interrupts must be disabled during a table read using the "retlw K" instruction (Example 6). Note it is not necessary to disable interrupts when using the "tablrd/tlrd" instruction for table read operations.

AUTHOR: Stan D'Souza, Logic Products Division

Software Implementation of I²C Bus Master

INTRODUCTION

A set of software routines for Microchip's mid-range, high speed 8-bit EPROM-based microcontrollers to implement I²C Bus Master Mode is given. Some of the members of PIC16CXX family (e.g., PIC16C71, PIC16C84) do not have on chip hardware implementation of I²C Bus interface. This application note describes the software implementation of I²C interface routines. Only the master mode of I²C interface is implemented with the PIC16CXX Microcontroller being the only master and communicating to multiple I²C slaves.

This application note does not describe the I²C Bus specifications and the user is assumed to have an understanding of the I²C Bus. For detailed information on the bus, the user is advised to read the I²C Bus Specification document from Philips/Signetics (order number 98-8080-575). The I²C Bus is a 2-wire serial bus with multiple masters and multiple slaves connected to these to two wires. The two wires consists of a clock line (SCL) and a data line (SDA) with both the lines being bi-directional. Bi-directional communication is facilitated through the use of wire-and connection (the lines are either active-low or passive high). The I²C Bus protocol also allows collision detection, clock synchronization and hand-shaking for multi-master systems. The clock is always generated by the master, but the slave may hold it low to generate a wait state.

In most of the systems the microcontroller is the master and the external peripheral devices are slaves. In these cases this application note can be used to attach I²C slaves to PIC16CXX (the master) microcontroller. The multi-master system is not implemented because it is extremely difficult to meet all the I²C Bus timing specification using software. For a true slave or multi-master system, some interface hardware is necessary (like START & STOP bit detection).

In addition to the low-level single master I²C routines, a collection of high level routines with various message structures is given. These high level macros/routines can be used as canned routines to interface to most I²C Slave devices. As an example, the test program talks to two Serial EEPROMs (Microchip's 24LC04 & 24LC01).

IMPLEMENTATION

Two levels of software routines are provided. The low-level routines are given in "i2c_low.asm" and the high level routines are given in "i2c_high.asm". The routines are described later. The messages passed (communicated on the two wire network) are abbreviated and certain notation is used to represent Start, Stop and other conditions. These abbreviations are described at first in Table 1.

TABLE 1 - DESCRIPTION OF ABBREVIATIONS USED

Abbreviation	Explanation
S	Start Condition
P	Stop Condition
SivAR	Slave Address (for read operation)
SivAW	Slave Address (for write operations)
A	Acknowledge condition (positive ACK)
N	Negative Acknowledge condition (NACK)
D	Data byte, D[0] represents byte 0, D[1] represents second byte



Software Implementation of I²C Bus Master

Message Format

In the high level routines, the basic structure of the message implemented is given. Every I²C slave supports one or more message structures. For example, Microchip's 24LC04 Serial EEPROM supports the following message (to write a byte to Serial EEPROM at current address counter) S-SlvAW-A.-D-A-P which basically means the following sequence of operations are required :

- (a) Send Start Bit
- (b) Send Slave Address for Write Operations
- (c) Expect Acknowledge
- (d) Send Data Byte
- (e) Expect Acknowledge
- (f) Issue a STOP Condition

Slave Address

Both 10 bit and 7 Bit addressing schemes are implemented as specified by the I²C Bus specification. Before calling a certain sub-routine (high level or low-level), the address of the slave being addressed must be loaded using either "LOAD_ADDR_8" (for 7 bit address slaves) or "LOAD_ADDR_10" macro (for 10 bit address slaves). These macros not only load the address of the slaves for all the following operations, but also setup conditions for 7 or 10 bit addressing modes. See the macros section for more details.

CLOCK STRETCHING

In I²C Bus, the clock (SCL Line) is always provided by the master. However, the slave can hold the line low even though the master has released it. The master must check this condition and wait for the slave to release the clock line. This provides a built in wait state for the I²C Bus. The slave may pull the clock low and ask the master to wait indicating it is busy. This feature is implemented and can be turned on or off as an assembly time option (by setting `_ENABLE_BUS_FREE_TIME` flag to be TRUE or FALSE). If the clock is held low for too long, say 1 msec, then an error condition is assumed and an RTCC interrupt is generated.

ARBITRATION

The I²C Bus specifies both bit by bit and byte mode arbitration procedure for multi-master systems. However, the arbitration is not needed in a single master system, and therefore not implemented in this application note.

HARDWARE

Two I/O pins are used to emulate the Clock Line SCL and the data line SDA. In the example test program, RB0 is used as SCL and RB1 as SDA line. On initialization, these I/O lines are configured as input pins (tri-state) and their respective latches are loaded with 0s. To emulate the high state (passive), these lines are turned as inputs and to emulate the active low state, the pins are turned as outputs (with the assumption of having external pull-up resistors on both the lines).

Software Implementation of I²C Bus Master

I²C ROUTINES

Status Register (File Register "Bus_Status") :

The bit definitions of the status register are described in the table given below. These bits reflect the status of the I²C Bus.

Bit #	Name	Description
0	_Bus_Busy	1 = Start Bit transmitted 0 = STOP condition.
1	_Abort	It is set when a fatal error condition is detected. The user must clear this bit. This bit is set when Clock Line (SCL) is stuck low.
2	_Txmt_Progress	1 = transmission in progress.
3	_Rcv_Progress	1 = reception in progress.
4	_Txmt_Success	1 = transmission successfully completed. 0 = error condition.
5	_Rcv_Success	1 = reception successfully completed. 0 = error condition.
6	_Fatal_Error	1 = FATAL error occurred. The communication was aborted.
7	_ACK_Error	1 = slave sent not $\overline{\text{ACK}}$ while the master was expecting an $\overline{\text{ACK}}$. This may happen for example if the slave was not responding to a message.

3

Control Register (File Register "Bus_Control") :

The bit definitions of the control register are described in the table given below. These bits must be set by the software prior to performing certain operations. Some of the high level routines described later in this section set these bits automatically.

Bit #	Name	Description
0	_10BitAddr	1 = 10 bit slave addressing 0 = 7 bit addressing.
1	_Slave_RW	1 = READ operation 0 = WRITE operation.
2	_Last_Byte_Rcv	1 = last byte must be received. Used to send not $\overline{\text{ACK}}$.
3,4,5	—	Unused bits, can be used as general purpose bits.
6	_SlaveActive	A status bit indicating if a slave is responding. This bit is set or cleared by calling the I ² C_TEST_DEVICE macro. See description of this I ² C_TEST_DEVICE macro.
7	_TIME_OUT_	A status bit indicating if a clock is stretched low for more than 1 msec, indicating a bus error. On this time out, the operation is aborted.

Software Implementation of I²C Bus Master

Low Level :

Function Name	Description
InitI ² CBus_Master	Initializes Control/Status Registers, and set SDA & SCL lines. Must be called on initialization.
TxmtStartBit	Transmits a START (S) condition.
TxmtStopBit	Transmits a STOP (P) condition.
LOAD_ADDR_8	The 7 bit slave's address must be passed as a constant parameter.
LOAD_ADDR_10	The 10 bit slave's address must be passed as a constant parameter.
Txmt_Slave_Addr	Transmits a Slave address. Prior to calling this routine, the address of the slave being addressed must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 routines. Also the Read/Write condition must be set in the control register.
SendData	Transmits a byte of data. Prior to calling this routine, the byte to be transmitted must be loaded into DataByte file register.
GetData	Receives a byte of data in DataByte file register. If the data byte to be received is the last byte, the _Last_Byte_Rcv bit in control register must be set prior to calling this routine.

Software Implementation of I²C Bus Master

MACROS

High Level :

The high level routines are implemented as a mixture of function calls and macros. These high level routines call the low level routines described above. In most cases only a few of the high level routines may be used and the user can remove or not include the routines not necessary to conserve program memory space. Examples are given for a few functions.

I²C_TEST_DEVICE

Parameters : None

Purpose : To test if a slave is present on the network

Description : Before using this macro, the address of the slave being tested must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 macro. Is the slave under test is present, then "_SlaveActive" status bit (in Bus_Control file register) is set. If not, then this bit is set 0, indicating that the slave is either not present on the network or is not listening.

Message : S-SlvAW-A-P

Example :

```
LOAD_ADDR_8 0xA0 ; 24LC04 address
I2C_TEST_DEVICE
btfs _SlaveActive ; See If slave is responding
goto SlaveNotPresent ; 24LC04 is not present
; Slave is present
; Continue with program
```

I²C_WR

Parameters : _BYTES_, _SourcePointer_
BYTES Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer Data Start Buffer pointer in RAM (file registers)

Purpose : A basic macro for writing a block of data to a slave

Description : This macro writes a block of data (no of bytes = _BYTES_) to a slave. The starting address of the block of data is _SourcePointer_. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Txmt_Success bit)

Message : S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P

Example :

```
btfs _Bus_Busy ; Check if bus is free
goto $-1
LOAD_ADDR_8 _Slave_1_Addr
I2C_WR 0x09, DataBegin ;
```

Software Implementation of I²C Bus Master

PC_WR_SUB

Parameters : `_BYTES_`, `_SourcePointer_`, `_Sub_Address_`
`_BYTES_` Number of bytes starting from RAM pointer `_SourcePointer_`
`_SourcePointer_` Data Start Buffer pointer in RAM (file Registers)
`_Sub_Address_` Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : Same as I²C_WR function, except that the starting address of the slave is also specified. For example, while writing to an I²C Memory Device, the sub-addr specifies the starting address of the memory. The I²C_WR may prove to be more efficient than this macro in most situations. Advantages will be found for Random Address Block Writes for Slaves with Auto Increment Sub-addresses (like Microchip's 24CXX series Serial EEPROMs)

Message : *S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P*

Example :

`LOAD_ADDR_8_Slave_2_Addr` ; Load addr of 7 bit slave

`I2C_WR_SUB 0x08, DataBegin+1, 0x30`

In the above example , 8 Bytes of data starting from addr (`DataBegin+1`) is written to 24LC04 Serial EEPROM beginning at `0x30` address

PC_WR_SUB_SWINC

Parameters : `_BYTES_`, `_SourcePointer_`, `_Sub_Address_`
`_BYTES_` Number of bytes starting from RAM pointer `_SourcePointer_`
`_SourcePointer_` Data Start Buffer pointer in RAM (file Registers)
`_Sub_Address_` Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : Same as I²C_WR_SUB function, except that the sub-address (incremented) is sent after every data byte. A very inefficient message structure and the Bus is given up after each data byte. This is useful for when the slave does not have an auto-increment sub-address feature.

Message : *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*

S-SlvAW-A-(SubA+1)-A-D[1]-A-P

and so on until #of Bytes

Software Implementation of I²C Bus Master

I²C_WR_BYTE_MEM

Parameters : _*BYTES_*, _*SourcePointer_*, _*Sub_Address_*
 *BYTES* Number of bytes starting from RAM pointer *_SourcePointer_*
 *SourcePointer* Data Start Buffer pointer in RAM (file Registers)
 _*Sub_Address_* Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-address

Description : Same as *I²C_WR_SUB_SWINC*, except that a delay is added between each message. This is necessary for some devices like EEPROMs which accept only a byte at a time for programming (devices without on chip ram buffer) and after each byte a delay is necessary before a next byte is written.

Message : *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*
 Delay 1 msec
 S-SlvAW-A-(SubA+1)-A-D[1]-A-P
 Delay 1 msec
 and so on until #of Bytes

I²C_WR_BUF_MEM

Parameters : _*BYTES_*, _*SourcePointer_*, _*Sub_Address_*, _*Device_BUF_SIZE_*
 *BYTES* Number of bytes starting from RAM pointer *_SourcePointer_*
 *SourcePointer* Data Start Buffer pointer in RAM (file Registers)
 _*Sub_Address_* Sub-address of the Slave
 _*Device_BUF_SIZE_* the slaves on-chip buffer size

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : This Macro/Function writes *#of_BYTES_* to an I²C memory device. However some devices, esp. EEPROMs must wait while the device enters into programming mode. But some devices have an on-chip temp. data hold buffer and is used to store data before the device actually enters into programming mode. For example, the 24C04 series of Serial EEPROMs from Microchip have an 8 byte data buffer. So one can send 8 bytes of data at a time and then the device enters programming mode. The master can either wait until a fixed time and then retry to program or can continuously poll for ACK bit and then transmit the next Block of data for programming.

Message : *I²C_SUB_WR* operations are performed in loop and each time data buffer of *BUF_SIZE* is output to the device. Then the device is checked for busy and when not busy another block of data is written.

Software Implementation of I²C Bus Master

I²C_READ

- Parameters : `_BYTES_`, `_DestPointer_`
- `_BYTES_` Number of bytes starting from RAM pointer `_SourcePointer_`
`_DestPointer_` Data Start Buffer pointer in RAM (file Registers)
- Purpose : A basic macro for reading a block of data from a slave
- Description : This macro reads a block of data (no of bytes = `_BYTES_`) from a slave. The starting address of the block of data is `_DestPointer_`. If an error occurs, the message is aborted and the user must check Status flags (e.g. `_Rcv_Success` bit). Note that on the last byte to receive, NACK is sent.
- Message : *S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P*

Example :

```
LOAD_ADDR_10 _Slave_3_Addr
I2C_READ 8, DataBegin
btfss  _Rcv_Success
goto   ReceiveError
goto   ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10 bit slave and stored in the master's ram starting at address DataBegin.

Software Implementation of I²C Bus Master

PC_READ_SUB

Parameters : `_BYTES_`, `_DestPointer_`, `_SubAddress`

<code>_BYTES_</code>	Number of bytes starting from RAM pointer <code>_SourcePointer_</code>
<code>_DestPointer_</code>	Data Start Buffer pointer in RAM (file Registers)
<code>_SubAddress_</code>	Sub-address of the slave

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data (no of bytes = `_BYTES_`) from a slave starting at slave's sub-address `_SubAddress`. The data received is stored in master's ram starting at address `_DestAddress`. If an error occurs, the message is aborted and the user must check Status flags (e.g. `_Rcv_Success` bit).

This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address between the sub-address write & the following reads, a STOP condition is not issued and a "REPEATED START" condition is used so that an other master will not take over the bus, and also that no other master will overwrite the sub-address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a memory device (e.g. : 24CXX serial of Serial EEPROMs from Microchip).

Message : *S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P*

Example :

```
LOAD_ADDR_10 _Slave_3_Addr
I2C_READ_SUB 8, DataBegin, 0x60
btfss _Rcv_Success
goto ReceiveError
goto ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10 bit slave (starting at address 0x60) and stored in the master's ram starting at address DataBegin.

PC_READ_BYTE or PC_READ_STATUS

Parameters : `_DestPointer_`

<code>_DestPointer_</code>	Data Start Buffer pointer in RAM (file Registers)
----------------------------	---

Purpose : To read a Status Byte from Slave

Description : Several I²C Devices can send a Status Byte upon reception of the control byte. This Macro reads a Status byte from a slave to the master's RAM location `_DestPointer_`. This function is basically the same as `I2C_READ` for a single byte read. As an example of this command, the 24Cxx serial Serial EEPROM from Microchip will send the memory data at the current location when `I2C_READ_STATUS` function is called. On successful operation of this command, `WREG = 1` else `WREG = 0` on errors.

Message : *S-SlvAR-A-D-A-N-P*

Software Implementation of I²C Bus Master

PC_WR_SUB_WR

Parameters : `_Bytes1_`, `_SrcPtr1_`, `_SubAddr_`, `_Bytes2_`, `_SrcPtr2_`

<code>_Bytes1_</code>	Number of bytes in the first data block
<code>_SrcPtr1_</code>	Starting address of first data block
<code>_SubAddr_</code>	Sub-address of the slave
<code>_Bytes2_</code>	Number of bytes in the second data block
<code>_SrcPtr2_</code>	Starting address of second data block

Purpose : To send two blocks of data to a slave at its sub address

Description : This Macro writes two blocks of data (of variable length) starting at slave's sub-address `_SubAddr_`. This Macro is essentially the same as calling `I2C_WR_SUB` twice, but a STOP bit is not sent in between the transmission of the two blocks. This way the Bus is not given up.

This function may be useful for devices which need two blocks of data in which the first block may be an extended address of a slave device. For example, a large I²C memory device, or a teletext device with an extended addressing scheme, may need multiple bytes of data in the first block that represents the actual physical address and is followed by a second block that actually represents the data.

Message : *S-SlvW-A-SubA-A-D1[0]-A-.....D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P*

PC_WR_SUB_RD

Parameters : `_Count1_`, `_SrcPtr_`, `_SubAddr_`, `_Count2_`, `_DestPtr_`

<code>_Count1_</code>	Length of the source buffer
<code>_SrcPtr_</code>	Source pointer address
<code>_SubAddr_</code>	Sub-address of the slave
<code>_Count2_</code>	Length of the destination buffer
<code>_DestPtr_</code>	Address of destination buffer

Purpose : To send a block of data and then receive a block of data

Description : This macro writes a block of data (of length `_Count1_`) to a slave starting at sub-address `_SubAddr_` and then reads a block of data (of length `_Count2_`) to the master's destination buffer (starting at address `_DestPtr_`). Although this operation can be performed using previously defined Macros, this function does not give up the bus in between the block writes and block reads. This is achieved by using the Repeated Start Condition.

Message : *S-SlvW-A-SubA-A-D1[0]-A-.....A-D1[N-1]-A-S-SlvR-A-D2[0]-A-.....A-D2[M-1]-N-P*

Software Implementation of I²C Bus Master

PC_WR_COM_WR

Parameters : `_Count1_`, `_SrcPtr1_`, `_Count2_`, `_SrcPtr2_`

<code>_Count1_</code>	Length of the first data block
<code>_SrcPtr1_</code>	Source pointer of the first data block
<code>_Count2_</code>	Length of the second data block
<code>_SrcPtr2_</code>	Source pointer of the second data block

Purpose : To send two blocks of data to a slave in one message

Description : This macro writes a block of data (of length `_Count1_`) to a slave and then sends another block of data (of length `_Count2_`) without giving up the bus.
For example, this kind of transaction can be used in an I²C LCD driver where a block of control & address information is needed and then another block of actual data to be displayed is needed.

Message : *S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P*

APPLICATIONS

The I²C Bus is a simple two wire serial protocol for inter-IC communications. A lot of peripheral devices (acting as slaves) are available in the market with I²C interface (e.g. serial EEPROM, clock/calendar, I/O Port expanders, LCD drivers, A/D converters, etc.). Although some of the PIC16CXX devices do not have on chip I²C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I²C bus can be implemented using software.

*AUTHOR: Amar Palacherla,
Logic Products Division*

Appendix A - I²C.H

```

;*****
; I2C Bus Header File
;*****
_ClkOut      equ    (_ClkIn >> 2)

;
; Compute the delay constants for setup & hold times
;
_40uS_Delay  set    (_ClkOut/250000)
_47uS_Delay  set    (_ClkOut/212766)
_50uS_Delay  set    (_ClkOut/200000)

#define_OPTION_INIT (0x00 | 0x03) ; Prescaler to RTCC for Approx 1 mSec timeout
;
#define _SCL      _ portb,0
#define _SDA      _ portb,1

#define _SCL_TRIS 0
#define _SDA_TRIS 1

#define _WRITE_   0
#define _READ_    1

;
; Register File Variables

CBLOCK 0x0C
SlaveAddr
SlaveAddrHi
DataByte
BitCount
Bus_Status
Bus_Control
DelayCount
DataByteCopy
SubAddr
ScrPtr
tempCount
StoreTemp_1
End_I2C_Ram
ENDC

; Slave Addr must be loader into this reg
; for 10 bit addressing mode
; load this reg with the data to be transmitted
; The bit number (0:7) transmitted or received
; Status Reg of I2C Bus for both TXMT & RCVE
; control Register of I2C Bus
; copy of DataByte for Left Shifts (destructive)
; sub-address of slave (used in I2C_HIGH.ASM)
; source pointer for data to be transmitted
; a temp variable for scratch RAM
; a temp variable for scratch RAM, do not disturb contents
; unused, only for ref of end of RAM allocation

```

Software Implementation of I²C Bus Master

```
*****
;
; I2C Bus Status Reg Bit Definitions
;
*****
#define Bus_Busy      Bus_Status,0
#define Abort        Bus_Status,1
#define Txmt_Progress Bus_Status,2
#define Rcv_Progress Bus_Status,3
#define Txmt_Success Bus_Status,4
#define Rcv_Success  Bus_Status,5
#define Fatal_Error  Bus_Status,6
#define ACK_Error    Bus_Status,7
;
*****
; I2C Bus Control Register
;
*****
#define _10BitAddr  Bus_Control,0
#define Slave_RW    Bus_Control,1
#define Last_Byte_Rcv Bus_Control,2
#define SlaveActive Bus_Control,6
#define TIME_OUT_  Bus_Control,7
;
*****
; General Purpose Macros
;
*****
RELEASE_BUS MACRO
    bsf _rp0 ; select page 1
    bsf _SDA ; tristate SDA
    bsf _SCL ; tristate SCL
    bcf _Bus_Busy ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
;
ENDM
;
; A MACRO To Load 8 OR 10 Bit Address To The Address Registers
;
; SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
; on 8 or 10 bit addressing modes
;
LOAD_ADDR_10 MACRO SLAVE_ADDRESS
    bsf _10BitAddr ; Slave has 10 bit address
;
*****
```

Software Implementation of I²C Bus Master

```
movlw (SLAVE_ADDRESS & 0xff)
movwf SlaveAddr
movlw ((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0
movwf SlaveAddr+1
; load low byte of address
; 10 bit addr is 11110XX0
; hi order address

ENDM

LOAD_ADDE_8 MACRO SLAVE_ADDRESS
    bcf 10BitAddr
    movlw (SLAVE_ADDRESS & 0xff)
    movwf SlaveAddr
    ; Set for 8 Bit Address Mode

ENDM
```


Appendix B - TEST.ASM

```

Title      "I2C Master Mode Implemetation"
Subtitle   "Rev 0.1      :   01 Mar 1993"
;*****
;
;      Software Implementation Of I2C Master Mode
;
; * Master Transmitter & Master Receiver Implemented in software
; * Slave Mode implemented in hardware
;
; *      Refer to Signetics/Philips I2C-Bus Specification
;
;      The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
;
; RB1 is SDA      (Any I/O Pin May Be used instead)
; RB0/INT is SCL (Any I/O Pin May Be used instead)
;
; *****
; Processor      16C71
; Radix          DEC
;
; _ClkIn equ 16000000 ; Input Clock Frequency Of PIC16C71
;
; include "d:\pictools\16Cxx.h"
;
; #define Slave_1_Addr 0xA0 ; Serial EEPROM #1
; #define Slave_2_Addr 0xA8 ; Serial EEPROM #2
; #define Slave_3_Addr 0xD6 ; Slave PIC16CXX
;
; #define ENABLE_BUS_FREE_TIME TRUE
; #define CLOCK_STRETCH_CHECK TRUE
; #define INCLUDE_HIGH_LEVEL_I2C TRUE
;
; include "i2c.h"
;
; CBLOCK End_I2C_Ram
; SaveStatus ; copy of STATUS Reg
; SaveWReg ; copy of WREG
; byteCount
; HoldData
;
; ENDC
;
; CBLOCK 0x20

```

Software Implementation of I²C Bus Master

```
DataBegin      ; Data to be read or written is stored here
ENDC

ORG 0x00
goto Start
ORG 0x04
;*****
; Interrupt Service Routine
;
; For I2C routines, only RTCC interrupt is used
; RTCC Interrupts enabled only if Clock Stretching is Used
; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
; MUST set TIME_OUT_ flag saying possibly a FATAL error ocured
; The user may choose to retry the operation later again
;*****
;*****
;*****
Interrupt:
; Save Interrupt Status (WREG & STATUS regs)
movwf SaveWReg      ; Save WREG
swapf status,w      ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
movwf SaveStatus    ; Save STATUS Reg
if _CLOCK_STRETCH_CHECK
btfss rtif
goto MayBeOtherInt  ; other Interrupts
bsf TIME_OUT_      ; MUST set this Flag, can take other desired actions here
bcf rtif
endif
; Check For Other Interrupts Here, This program usesd only RTCC & INT Interrupt
;
MayBeOtherInt:
NOP
; RestoreIntStatus:
swapf SaveStatus,w  ; Restore Interrupt Status
movwf status        ; restore STATUS Reg
swapf SaveWReg      ; restore WREG
swapf SaveWReg,w
retfie
;*****
; Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
; include "i2c_high.asm"
```

Software Implementation of I²C Bus Master

```
*****
endif
;
;*****

ReadSlave1:
;
; EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
;
;   LOAD_ADDR_8   _Slave_1_Addr
wait1:
I2C_TEST_DEVICE
btfss   _SlaveActive   ; See If slave is responding
goto    wait1          ; if stuck for ever, recover from WDT, can use other schemes
clrwdt
I2C_READ_SUB   8, DataBegin+1, 0x50
;
; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
;
;   LOAD_ADDR_8   _Slave_2_Addr
wait2:
I2C_TEST_DEVICE
btfss   _SlaveActive   ; See If slave is responding
goto    wait2          ; if stuck for ever, recover from WDT, can use other schemes
clrwdt
I2C_READ_SUB   8, DataBegin+1, 0x60

return
;
;*****

ReadSlave3:
;
;   LOAD_ADDR_8   _Slave_3_Addr
wait3:
I2C_TEST_DEVICE
btfss   _SlaveActive   ; See If slave is responding
goto    wait3          ; if stuck for ever, recover from WDT, can use other schemes
clrwdt
I2C_READ_SUB   8, DataBegin, 0

return
;
;*****
; Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
;*****
```


Software Implementation of I²C Bus Master

Appendix C - LOW.ASM

```
*****
;
; Low Level I2C Routines
;
; Single Master Transmitter & Single Master Receiver Routines
; These routines can very easily be converted to Multi-Master System
; when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
; detection is available.
;
; The generic high level routines are given in I2C_HIGH.ASM
;
*****

;*****
; I2C Bus Initialization
;
;*****
InitI2CBus_Master:
    bcf     _rp0
    movf   _portb,w
    andlw  0xFC
    movwf  _portb
    RELEASE_BUS
    clrf   Bus_Status
    clrf   Bus_Control
    return
;
;*****
; Send Start Bit
;
;*****
TxmtStartBit:
    bsf    _rp0
    bsf    _SDA
    bsf    _SCL
; Setup time for a REPEATED START condition (4.7 us)
;
; call    Delay40uSec
; only necessary for setup time
;
*****
```

```

;
; bcf _SDA ; give a falling edge on SDA while clock is high
; call Delay47uSec ; only necessary for START HOLD time
; bsf _Bus_Busy ; on a start condition bus is busy
; return
;
;*****
; Send Stop Bit
;*****
;
;*****
; TxmtStopBit:
; bcf _rp0 ; select page 1
; bcf _SCL
; bcf _SDA ; set SDA low
; bcf _SCL ; Clock is pulled up
; call Delay40uSec ; Setup Time For STOP Condition
; bsf _SDA ; give a rising edge on SDA while CLOCK is high
;
; if _ENABLE_BUS_FREE_TIME
; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
;
; call Delay47uSec
;
; bcf _Bus_Busy ; on a stop condition bus is considered Free
; return
;
;*****
; Abort Transmission
;*****
;
; Send STOP Bit & set Abort Flag
;*****
;
;*****
; AbortTransmission:
; call TxmtStopBit
; bsf _Abort
; return
;*****
; Transmit Address (1st Byte)& Put in Read/Write Operation
;
; Transmits Slave Addr On the 1st byte and set LSB to R/W operation
;

```

Software Implementation of I²C Bus Master

```

; Slave Address must be loaded into SlaveAddr reg
; The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
;
; On Success, return TRUE in WREG, else FALSE in WREG
;
; If desired, the failure may be tested by the bits in Bus Status Reg
;
;*****
Txmt_Slave_Addr:
    bcf   ACK_Error      ; reset Acknowledge error bit
    btfss 10BitAddr
    goto SevenBitAddr
;
    btfss Slave_RW
    goto TenBitAddrWR ; For 10 Bit WR simply send 10 bit addr
;
; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
; and then Hi Byte Only for read operation
;
TenBitAddrRd:
    bcf   Slave_RW      ; temporarily set for WR operation
    call TenBitAddrWR
    btfss Txmt_Success
    retlw FALSE
;
    call TxmtStartBit
    bsf   Slave_RW      ; send A REPEATED START condition
;
    movf  SlaveAddr+1,W
    movwf DataByte
    bsf   DataByte,LSB ; Read Operation
    call SendData
    goto AddrSendTest ; send ONLY high byte of 10 bit addr slave
;
; if successfully transmitted, expect an ACK bit
;
    btfss Txmt_Success
    goto AddrSendFail ; if not successful, generate STOP & abort transfer
;
TenBitAddrWR:
    movf  SlaveAddr+1,W
    movwf DataByte
    bcf   DataByte,LSB ; WR Operation
;
; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)

```



```

; then save RETURN Address Pointer
;
; call    SendData    ; send high byte of 10 bit addr slave
;
; if successfully transmitted, expect an ACK bit
;
    btfs    Txmt_Success
    goto    AddrSendFail
;
; if not successful, generate STOP & abort transfer
;
    movf    SlaveAddr,W
    movwf   DataByte
    goto    EndTxmtAddr
;
; load addr to DataByte for transmission
;
SevenBitAddr:
    movf    SlaveAddr,W
    movwf   DataByte
    bcf     DataByte,LSB
    btfsc  Slave_RW
    bsf     DataByte,LSB
; Read Operation
;
    call    SendData    ; send 8 bits of address, bus is our's
;
; if successfully transmitted, expect an ACK bit
;
    AddrSendTest:
    btfs    Txmt_Success    ; skip if successful
    goto    AddrSendFail
    clrwdt
    retlw   TRUE
;
    AddrSendFail:
    clrwdt
    btfs    ACK_Error
    retlw   FALSE
;
; Address Not Acknowledged, so send STOP bit
;
    call    TxmtStopBit
    retlw   FALSE
;
; *****
; Transmit A Byte Of Data
;
; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
; and INT Interrupt on Rising edge is enabled and also RTCC timeout interrupt is enabled
; The clock stretching slows down the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock stretching, then

```



Software Implementation of I²C Bus Master

```

; this feature can be disabled during Assembly time by setting
; _CLOCK_STRETCH_ENABLED must be set to FALSE.
; *****
SendData:
;
; TxmtByte & Send Data are same, Can check errors here before calling TxmtByte
; For future compatibility, the user MUST call SendData & NOT TxmtByte
;
; goto TxmtByte
;
TxmtByte:
    movf   DataByte,w
    movwf  DataByteCopy
    bsf    Txmt_Progress
    bcf    Txmt_Success
    movlw  0x08
    movwf  BitCount
    bsf    rp0
    if _CLOCK_STRETCH_CHECK
        ; set RTCC to INT CLK timeout for 1 mSec
        ; do not disturb user's selection of RPUB in OPTION Register
    ;
    movf  option,w
    andlw OPTION_INIT
    movwf option
    endif

TxmtNextBit:
    clrwdt
    bcf    SCL
    rlf   DataByteCopy
    bcf   _SDA
    btfsc C
    bsf   SDA
    call Delay47uSec
    bsf   SCL
    call Delay40uSec
    if _CLOCK_STRETCH_CHECK
        bcf rp0
    clrf rtcc
    bcf rtif
    bsf rtie
    bcf TIME_OUT_
    Check_SCL_1:
        btfsc TIME_OUT
        goto Bus_Fatal_Error
; clear WDT, set for 18 mSec
; MSB first, Note Databyte Is Lost
; guarantee min LOW TIME tLOW & Setup time
; set clock high, check if clock is high, else clock being stretched
; guarantee min HIGH TIME THIGH
; clear RTCC
; clear any pending flags
; enable RTCC Interrupt
; reset timeout error flag
; if RTCC timeout or Error then Abort & return
; Possible FATAL Error on Bus

```

```

; if clock not being stretched, it must be high
; loop until SCL high or RTCC timeout interrupt
; Clock good, disable RTCC interrupts

    bcf rp0
    bcfss SCL
    goto Check_SCL_1
    bcf rt1e
    bsf rp0
endif
    decfsz BitCount
    goto TxmtNextBit
; Check For Acknowledge
;
; SCL ; reset clock
; SDA ; Release SDA line for Slave to pull down
    call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
    bsf SCL ; clock for slave to ACK
    call Delay40uSec ; guarantee min HIGH TIME tHIGH
    bcf rp0 ; select PAGE 0 to test PortB pin SDA
    btfsc SDA ; SDA should be pulled low by slave if OK
    goto TxmtErrorAck
;
    bsf rp0
    bcf SCL ; reset clock
;
    bcf Txmt_Progress ; reset TXMT bit in Bus Status
    bsf Txmt_Success ; transmission successful
    bcf ACK_Error ; ACK OK
    return
_TxmtErrorAck:
    RELEASE_BUS
    bcf Txmt_Progress ; reset TXMT bit in Bus Status
    bcf Txmt_Success ; transmission NOT successful
    bsf ACK_Error ; No ACK From Slave
    return
;
; *****
; Receive A Byte Of Data From Slave
;
; assume address is already sent
; if last byte to be received, do not acknowledge slave (last byte is tested from
; Last_ByteRcv bit of control reg)
; Data Received on successful reception is in DataReg register
;
; *****
GetData:
    goto RcvByte
;
; *****

```



Software Implementation of I²C Bus Master

```

; RcvByte:
    bsf   Rcv_Progress      ; set Bus status for txmt progress
    bcf   Rcv_Success      ; reset status bit

    movlw 0x08
    movwf BitCount
    if _CLOCK_STRETCH_CHECK
        bsf   rp0
    ; set RTCC to INT CLK timeout for 1 mSec
    ; do not disturb user's selection of RPUB in OPTION Register
    ;
    movf  option,w
    andlw OPTION_INIT
    movwf option
    endif

RcvNextBit:
    clrwdt
    bsf   rp0              ; clear WDT, set for 18 mSec
    bcf   SCL              ; page 1 for TRIS manipulation
    bcf   SDA              ; can be removed from loop
    call Delay47uSec       ; guarantee min LOW TIME tLOW & Setup time
    bsf   SCL              ; clock high, data sent by slave
    call Delay40uSec       ; guarantee min HIGH TIME tHIGH
    if _CLOCK_STRETCH_CHECK
        bcf   rp0
    ; clear RTCC
    clrf  rtcc
    bcf   rtif             ; clear any pending flags
    bcf   rtie             ; enable RTCC Interrupt
    bcf   TIME_OUT_1      ; reset timeout error flag
    bcf   TIME_OUT_2
    Check_SCL_2:
        btfsc TIME_OUT_1
        goto Bus_Fatal_Error
        bcf   rp0
        btfss SCL
        goto Check_SCL_2
        bcf   rtie
        bcf   rp0
    endif
    bcf   rp0
    bcf   C
    btfsc SDA
    bsf   C
    ;
    rlf   DataByte
    decfsz BitCount
    goto RcvNextBit
;
; TEMP ??? DO 2 out of 3 Majority detect
; left shift data ( MSB first)

```

Software Implementation of I²C Bus Master

```
; Generate ACK bit if not last byte to be read,
; if last byte Generate NACK ; do not send ACK on last byte, main routine will send a STOP bit
;
;
    bsf rp0
    bcf SCL
    bcf SDA
    btrfsc Last_Byte_Rcv
    bsf SDA
    call Delay47uSec
    bcf SCL
    call Delay40uSec
RcvEnd:
    bcf SCL ; reset clock
    bcf Rcv_Progress ; reset TXMT bit in Bus Status
    bsf Rcv_Success ; transmission successful
    bcf ACK_Error ; ACK OK
    return
;
    if CLOCK_STRETCH_CHECK
;***** Fatal Error On I2C Bus *****
;
; Slave pulling clock for too long or if SCL Line is stuck low.
; This occurs if during Transmission, SCL is stuck low for period longer than approx 1ms
; and RTCC times out ( approx 4096 cycles : 256 * 16 - prescaler of 16).
;*****
Bus_Fatal_Error:
; disable RTCC Interrupt
;
    bcf rtie ; disable RTCC interrupts, until next TXMT try
;
;
    RELEASE_BUS
;
; Set the Bus_Status Bits appropriately
;
    bsf Abort ; transmission was aborted
    bsf Fatal_Error ; FATAL Error occurred
    bcf Txmt_Progress ; Transmission Is Not in Progress
    bcf Txmt_Success ; Transmission Unsuccessful
;
    call TxmtStopBit ; Try sending a STOP bit, may be not successful
;
;
; return
```


Software Implementation of I²C Bus Master

```

movlw  (_SourcePointer_ - 1)
movwf  fsr
movf   indf,w
movwf  StoreTemp_1
movlw  Sub_Address
movwf  indf
call   i2c_block_write ; write _BYTES_H1 block of data
movf   StoreTemp_1,w
movwf  (_SourcePointer_ - 1) ; restore contents of (_SourcePointer_ - 1)
call   TxmtStopBit ; Issue a stop bit for slave to end transmission
ENDM
;*****
; I2C_Wr_Sub_SWINC
;
; Parameters :
;   _BYTES_           #of bytes starting from RAM pointer _SourcePointer_ (constant)
;   _SourcePointer_   Data Start Buffer pointer in RAM (file registers)
;   _Sub_Address_     Sub-address of Slave (constant)
;
; Sequence :
;   S-SlVwA-A-(SubA+0)-A-D[0]-A-P
;   S-SlVwA-A-(SubA+1)-A-D[1]-A-P
;   and so on until #of Bytes
;
; If an error occurs then the routine simply returns and user should check for
; flags in Bus_Status Reg (for eg. _Txmt_Success Flag
;
; Returns :   WREG = 1 on success, else WREG = 0
;
; COMMENTS : Very In-efficient, Bus is given up after every Byte Write
;
; Some I2C devices addressed with a sub-address do not increment automatically
; after an access of each byte. Thus a block of data sent must have a sub-address
; followed by a data byte.
;*****
I2C_Wr_Sub_SWINC MACRO _BYTES_,_SourcePointer_,_Sub_Address_
variable i ; TEMP ??? : Assembler Does Not Support This
i = 0

```

Software Implementation of I²C Bus Master

```

movf      (_Source_Pointer_ + i),w
movwf    SrcPtr
movf      (_Sub_Address_ + i),w
movwf    SubAddr
call     i2c_byte_wr_sub
; write a byte of data at sub address
        i++
        .endw
        ENDM
;
;
; Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
;
_i2c_byte_wr_sub:
call     TxmtStartBit
        bcf     Slave_RW
call     Txmt_Slave_Addr
        btfs   Txmt_Success
goto     block_wrl_fail
movf     SubAddr,w
movwf    DataByte
call     SendData
        btfs   Txmt_Success
goto     block_wrl_fail
movf     SrcPtr,w
movwf    DataByte
call     SendData
        btfs   Txmt_Success
goto     block_wrl_fail
goto     block_wrl_pass
; return back to called routine from either_block_wrl_pass or_block_wrl_fail
;
_block_wrl_fail:
call     TxmtStopBit
        retlw  FALSE
_block_wrl_pass:
call     TxmtStopBit
        retlw  TRUE
;
;*****
;
; I2C_WR_MEM_BYTE

```

Software Implementation of I²C Bus Master

```

;
; Some I2C devices like a EEPROM need to wait fo some time after every byte write
; (when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
; but in addition adds a delay after each byte.
;
; Some EEPROM memories (like Microchip's 24Cxx Series have on-chip data buffer), and hence
; this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
; for a block of data and then insert a delay until the whole buffer is written.
;
; Parameters :
;   _BYTES_           #of bytes starting from RAM pointer _SourcePointer_ (constant)
;   _SourcePointer_  Data Start Buffer pointer in RAM (file Registers)
;   _Sub_Address_    Sub-address of Slave (constant)
;
; Sequence :
;   S-SlVwA-A-(SubA+0)-A-D[0]-A-P      ; The user can chnage this value to desired delay
;   Delay 1 mSec
;   S-SlVwA-A-(SubA+1)-A-D[1]-A-P
;   Delay 1 mSec
;   and so on until #of Bytes
;
;*****
I2C_WR_BYTE_MEM MACRO _BYTES_, _SourcePointer_, _Sub_Address_
variable i
i = 0
while (i < _BYTES_)
movf (_SourcePointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call _i2c_byte_wr_sub ; write a byte of data at sub address
call Delay50uSec
i++
.endw
ENDM
;*****
;
; This Macro/Function writes #of _BYTES_ to an I2C memory device. However
; some devices, esp. EEPROMs must wait while the device enters into programming
; mode. But some devices have an onchip temp data hold buffer and is used to
; store data before the device actually enters into programming mode.
;

```

Software Implementation of I²C Bus Master

```

;
; For example, the 24C04 series of Serial EEPROMs from Microchip
; have an 8 byte data buffer. So one can send 8 bytes of data at a time
; and then the device enters programming mode. The master can either wait
; until a fixed time and then retry to program or can continuously poll
; for ACK bit and then transmit the next Block of data for programming
;
; Parameters :
;   BYTES           # of bytes to write to memory
;   SourcePointer_  Pointer to the block of data
;   SubAddress      Sub-address of the slave
;   Device_BUF_SIZE The on chip buffer size of the I2C slave
;
; Sequence of operations
;   I2C SUB_WR operations are performed in loop and each time
;   data buffer of BUF_SIZE is output to the device. Then
;   the device is checked for busy and when not busy another
;   block of data is written
;
; *****
; I2C_WR_BUF_MEM MACRO _BYTES, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_
;
;   variable i, j
;
;   if ( !_BYTES_ )
;       exitm
;   elif ( _BYTES_ <= _Device_BUF_SIZE_ )
;       I2C_WR_SUB _BYTES_, _SourcePointer_, _SubAddress_
;
;       exitm
;   else
;       i = 0
;       j = ( _BYTES_ / _Device_BUF_SIZE_ )
;       .while ( i < j )
;           I2C_WR_SUB _Device_BUF_SIZE_, ( _SourcePointer_ + i*_Device_BUF_SIZE_ +
; i*_Device_BUF_SIZE_ )
;
;           call IsSlaveActive
;           btfs SlaveActive
;           goto $-2
;
;           i++
;
; *****

```



Software Implementation of I²C Bus Master

```

.endw
j = (_BYTES_ - i*_Device_BUF_SIZE_)
    if (j)
        I2C_WR_SUB    j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ + i*_Device_BUF_SIZE_)
    endif
endif
ENDM

;*****
; I2C_READ
;
; The basic MACRO/procedure to read a block message from a slave device
;
; Parameters :
;   _BYTES_      : constant : #of bytes to receive
;   _DestPointer_ : destination pointer of RAM (File Registers)
;
; Sequence :
;   S-SlVAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
; If last byte, then Master will NOT Acknowledge (send NACK)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;        mode addressing must be set
;*****
MACRO _BYTES_, _DestPointer_
    I2C_READ
    movlw (_BYTES_ -1)
    movwf tempCount
    movlw _DestPointer_
    movwf fsr
    call i2c_block_read
ENDM

_i2c_block_read:
    call TxmStartBit
    bsf Slave_RW
    bcf Last_Byte_Rcv
    ; send START bit
    ; set for read operation
    ; not a last byte to rcv

```

Software Implementation of I²C Bus Master

```

call Txmt_Slave_Addr      ; if successful, then Txmt_Success bit is set
btfs Txmt_Success
goto block_rdl_loop
call TxmtStopBit
retlw FALSE
;
;_block_rdl_loop:
call GetData
movf DataByte,w
movwf indf
incf fsr
decfsz tempCount
goto block_rdl_loop
bsf Last_Byte_Rcv
call GetData
movf DataByte,w
movwf indf
call TxmtStopBit
retlw TRUE
;*****
;
; I2C_READ_SUB
; This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address
; Between the sub-address write & the following reads, a STOP condition is not issued and
; a "REPEATED START" condition is used so that an other master will not take over the bus,
; and also that no other master will overwrite the sub-address of the same salve.
; This function is very commonly used in accessing Random/Sequential reads from a
; memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
;
; Parameters :
;   _BYTES_           # of bytes to read
;   _DestPointer_    The destination pointer of data to be received.
;   _SubAddress_     The sub-address of the slave
;
; Sequence :
;   S-SlVAV-A-SubAddr-A-D[0]-A-.....-A-D[N-1]-N-P
;*****
I2C_READ_SUB MACRO _BYTES_, _DestPointer_, _SubAddress_
bcf Slave_RW           ; set for write operation
call TxmtStartBit     ; send START bit
call Txmt_Slave_Addr  ; if successful, then Txmt_Success bit is set

```



Software Implementation of I²C Bus Master

```
movlw SubAddress_
movwf DataByte
call SendData
;
; do not send STOP after this, use REPEATED START condition
;

I2C_READ_BYTES, DestPointer_
ENDM

;*****
;
; I2C_READ_STATUS
;
; This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
; send a status byte upon reception of a control byte
; This is basically same as I2C_READ_MACRO for reading a single byte
;
; For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory
; data at the current address location
;
; On success WREG = 1 else = 0
;
;*****
I2C_READ_STATUS_MACRO DestPointer_
;*****
call TxmtStartBit ; send START bit
bsf Slave_RW ; set for read operation
call Txmt_Slave_Addr ; if successful, then Txmt_Success bit is set
btfsc Txmt_Success
goto byte_rdl_loop ; read a byte
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw FALSE ; Error : may be device not responding
_byte_rdl_loop:
bsf Last_Byte_Rcv ; last byte to rcv, so send NACK
call GetData
movf DataByte,w
movwf DestPointer_
call TxmtStopBit ; Issue a stop bit for slave to end transmission
btfsc Rcv_Success
retlw FALSE
retlw TRUE
```


Software Implementation of I²C Bus Master

```

ENDM
;*****
I2C_READ_BYTE MACRO DestPointer_
I2C_READ_STATUS MACRO DestPointer_
ENDM
;*****
;
; I2C_WR_SUB_WR
;
; This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This
; may be useful for devices which need 2 blocks of data in which the first block may be an
; extended address of a slave device. For example, a large I2C memory device, or a teletext
; device with an extended addressing scheme, may need multiple bytes of data in the 1st block
; that represents the actual physical address and is followed by a 2nd block that actually
; represents the data.
;
; Parameters :
;
; BYTES1 1st block #of bytes
; SourcePointer1_ Start Pointer of the 1st block
; SubAddress_ Sub-Address of slave
; BYTES2 2st block #of bytes
; SourcePointer2_ Start Pointer of the 2nd block
;
; Sequence :
; S-SLW-A-SubA-DI[0]-A-....-DI[N-1]-A-D2[0]-A-....A-D2[M-1]-A-P
;
; Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
; a STOP bit is not sent (bus is not given up) in between
; the two I2C_WR_SUB
;
; Check Txmt_Success flag for any transmission errors
;*****
I2C_WR_SUB_WR MACRO _COUNT1_, _SourcePointer1_, _SubAddress_, _COUNT2_, _SourcePointer2_
movlw (_COUNT1_ + 1)
movwf tempCount
movlw (_SourcePointer1_ - 1)
movwf fsr
movf indf,w

```



Software Implementation of I²C Bus Master

```
movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ -1)
movlw Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ -1)
call i2c_block_write ; write_BYTES_+1 block of data
;
movf StoreTemp_1,w
movwf (_SourcePointer_ - 1) ; restore contents of (_SourcePointer_ - 1)
; Block 1 write over
; Send Block 2
movlw COUNT2
movwf tempCount
movlw SourcePointer2_
movwf fsr
call block_wrl_loop
;
call TxmtStopBit ; Issue a stop bit for slave to end transmission
ENDM
;*****
;
; I2C_WR_SUB_RD
;
; This macro writes a block of data from SourcePointer of length COUNT1_ to a slave
; at sub-address and then Reads a block of length COUNT2_ to destination
; address pointer
;
; Message Structure :
; S-S1VW-A-SubA-A-D1[0]-A-.....-A-DI[N-1]-A-S-S1VR-A-D2[0]-A-.....-A-D2[M-1]-N-P
;
; Parameters :
; COUNT1_ Length Of Source Buffer
; SourcePointer_ Source Pointer Address
; Sub_Address_ The Sub Address Of the slave
; COUNT2_ The length of Destination Buffer
; DestPointer_ The start address of Destination Pointer
;*****
I2C_WR_SUB_RD MACRO _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_
movlw (_COUNT1_ + 1)
movwf tempCount
movlw (_SourcePointer_ - 1)
movwf fsr
movf indf,w
```

Software Implementation of I²C Bus Master

```

movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ -1)
movlw Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ -1)
call i2c_block_write ; write_BYTES+1 block of data
;
movf StoreTemp_1,w
movwf (_SourcePointer1_ - 1) ; restore contents of (_SourcePointer_ - 1)
;
; Without sending a STOP bit, read a block of data by using a REPEATED
; Start Condition
;
I2C_READ _COUNT2_, _DestPointer_
ENDM
;*****
;
; I2C_WR_COM_WR
;
; This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
; the bus after sending the first block.
; For example, this kind of transaction is used in an LCD driver where a
; a block of control & address info is needed and then another block of actual data
; to be displayed is needed.
;
; Message Structure :
; S-S1VW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....-A-D2[M-1]-A-P
; NOTE : This message is same as calling two I2C_WR Macros, except that
; the bus is not given up between the sending of 2 blocks (this is
; done by not sending a STOP bit inbetween)
;
; Parameters :
; _COUNT1_ Length Of Source Buffer #1
; _SourcePointer1_ Source Pointer Address of 1st buffer
; _COUNT2_ The length of Destination Buffer
; _SourcePointer2_ Source Pointer Address of 2nd Buffer
;
;*****
I2C_WR_COM_WR MACRO _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_
movlw _COUNT1_
movwf tempCount
movlw _SourcePointer1_
movwf fsr
call i2c_block_write
;

```



Software Implementation of I²C Bus Master

```
; First block sent, now send 2nd block of data
;
    movlw COUNT2_
    movwf tempCount
    movlw SourcePointer2_
    movwf fsr
    call block_wrt_loop
;
    call TxmtStopBit      ; End of Double buffer txmt
    ENDM
;
; *****
; INCLUDE I2C Low Level Routines Here
; *****
include "i2c_low.asm"
```

Software Implementation of I²C Bus Master

Appendix E - I²C test.lst

```
MPASM 50.24
"I2C Master Mode Implementation"
"Rev 0.1 : 01 Mar 1993"

Title "I2C Master Mode Implementation"
SubTitle "Rev 0.1 : 01 Mar 1993"

;*****
;
; Software Implementation Of I2C Master Mode
;
; * Master Transmitter & Master Receiver Implemented in software
; * Slave Mode implemented in hardware
;
; * Refer to Sigmetics/Philips I2C-Bus Specification
;
; The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
;
; RB1 is SDA (Any I/O Pin May Be used instead)
; RB0/INT is SCL (Any I/O Pin May Be used instead)
;
; *****
; Processor 16C71
; Radix DEC
00F4 2400 _ClkIn equ 16000000 ; Input Clock Frequency Of PIC16C71
include "d:\pictools\i6cxx.h"
```

Software Implementation of I²C Bus Master

```

;
00A0 #define _Slave_1_Addr 0xA0 ; Serial EEPROM #1
00AC #define _Slave_2_Addr 0xAC ; Serial EEPROM #2
00D6 #define _Slave_3_Addr 0xD6 ; Slave PIC16CXX
0046 #define _ENABLE_BUS_FREE_TIME TRUE
0047 #define _CLOCK_STRETCH_CHECK TRUE
0048 #define _INCLUDE_HIGH_LEVEL_I2C TRUE

include "i2c.h"
;*****
; I2C Bus Header File
;*****
003D 0900 _ClkOut equ (_ClkIn >> 2)

;
; Compute the delay constants for setup & hold times
;
0010 _40uS_Delay set (_ClkOut/250000)
0012 _47uS_Delay set (_ClkOut/212766)
0014 _50uS_Delay set (_ClkOut/200000)

0049 #define _OPTION_INIT (0xC0 | 0x03) ; Prescaler to RTCC for Approx 1 msec timeout
;
004A #define _SCL _ portb,0
004E #define _SDA _ portb,1

004C #define _SCL_TRIS trisb,0
004D #define _SDA_TRIS trisb,1

0000 #define _WRITE_ 0
0001 #define _READ_ 1
; Register File Variables

```

```

CELOCK 0x0C
SlaveAddr ; Slave Addr must be loader into this reg
SlaveAddrHi ; for 10 bit addressing mode
DataByte ; load this reg with the data to be transmitted
BitCount ; The bit number (0:7) transmitted or received
Bus_Status ; Status Reg of I2C Bus for both TXMF & RCVE
Bus_Control ; control Register of I2C Bus
DelayCount
DataByteCopy ; copy of DataByte for Left Shifts (destructive)
SubAddr ; sub-address of slave (used in I2C_HIGH.ASM)
SrcPtr ; source pointer for data to be transmitted
tempCount ; a temp variable for scratch RAM
StoreTemp_1 ; a temp variable for scratch RAM, do not disturb contents
End_I2C_Ram ; unused, only for ref of end of RAM allocation
ENDC
;*****
; I2C Bus Status Reg Bit Definitions
;*****
#define Bus_Busy Bus_Status,0
#define Abort Bus_Status,1
#define Txmt_Progress Bus_Status,2
#define Rcv_Progress Bus_Status,3
#define Txmt_Success Bus_Status,4
#define Rcv_Success Bus_Status,5
#define Fatal_Error Bus_Status,6
#define ACK_Error Bus_Status,7
;*****
000C 0001
000D 0001
000E 0001
000F 0001
0010 0001
0011 0001
0012 0001
0013 0001
0014 0000
0014 0001
0015 0001
0016 0000
0016 0001
0017 0001
0018 0000
0018 0001
004E #define Bus_Busy Bus_Status,0
004F #define Abort Bus_Status,1
0050 #define Txmt_Progress Bus_Status,2
0051 #define Rcv_Progress Bus_Status,3
0052 #define Txmt_Success Bus_Status,4
0053 #define Rcv_Success Bus_Status,5
0054 #define Fatal_Error Bus_Status,6
0055 #define ACK_Error Bus_Status,7
;*****

```



Software Implementation of I²C Bus Master

```

movwf SlaveAddr+1          ; hi order address

ENDM

LOAD_ADDR_8 MACRO SLAVE_ADDRESS
    bcf _10BitAddr          ; Set for 8 Bit Address Mode
    movlw (SLAVE_ADDRESS & 0xff)
    movwf SlaveAddr
ENDM

                                CBLOCK _End_I2C_Ram
                                ; copy of STATUS Reg
                                ; copy of WREG
0018 0001 SaveStatus
0019 0001 SaveWReg
001A 0001 byteCount
001B 0001 HoldData
                                ENDC

                                CBLOCK 0x20
0020 0001 DataBegin          ; Data to be read or written is stored here
                                ENDC

                                ORG 0x00
                                goto Start
                                ORG 0x04
;*****
; Interrupt Service Routine
;
; For I2C routines, only RTCC interrupt is used
; RTCC Interrupts enabled only if Clock Stretching is Used
0000 2956

```

Software Implementation of I²C Bus Master

```

; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
; MUST set _TIME_OUT_ flag saying possibly a FATAL error ocured
; The user may choose to retry the operation later again
;
;*****
Interrupt:
;
; Save Interrupt Status (WREG & STATUS regs)
;
0004 0099      movwf   SaveWReg      ; Save WREG
0005 0E03      swapf   status,w    ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098      movwf   SaveStatus ; Save STATUS Reg if _CLOCK_STRETCH_CHECK RTCC Interrupts enabled only if Clock Stretching is Used
0007 1D0B      btfs    rtif
0008 280B      goto    MayBeOtherInt ; other Interrupts
0009 1791      bsf     TIME_OUT_    ; MUST set this Flag, can take other desired actions here
000A 110B      bcf     rtif
                        endif
;
; Check For Other Interrupts Here, This program usesd only RTCC & INT Interrupt
;
MayBeOtherInt:
NOP
;
RestoreIntStatus: ; Restore Interrupt Status
swapf SaveStatus,w
movwf status      ; restore STATUS Reg
swapf SaveWReg
swapf SaveWReg,w  ; restore WREG
retfie
;
;*****
; Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C

```


Software Implementation of I²C Bus Master

```
; S-SLVW-A-(SubA+1)-A-D[1]-A-P
; and so on until #of Bytes
;
; If an error occurs then the routine simply returns and user should check
; for flags in Bus_Status Reg (for eg. _Txmt_Success flag
;
; Returns : WREG = 1 on success, else WREG = 0
;
; COMMENTS : Very In-efficient, Bus is given up after every Byte Write
;
; Some I2C devices addressed with a sub-address do not increment
; automatically after an access of each byte. Thus a block of data
; sent must have a sub-address followed by a data byte.
;
; *****
; I2C_WR_SUB_SWINC MACRO _BYTES_, _SourcePointer_, _Sub_Address_
;
; variable i ; TEMP ????: Assembler Does Not Support This
;
; i = 0
;
; .while (i < _BYTES_)
;
; movf (_SourcePointer_ + i),w
; movwf SrcPtr
; movf (_Sub_Address_ + i),w
; movwf SubAddr
; call i2c_byte_wr_sub ; write a byte of data at sub address
; i++
; .endw
;
; *****
ENDM
```


Software Implementation of I²C Bus Master

```
;
; Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
;
i2c_byte_wr_sub:
0025 2057      call TxmtStartBit      ; send START bit
0026 1091      bcf Slave_RW      ; set for write operation
0027 206B      call Txmt_Slave_Addr  ; if successful, then Txmt_Success bit is set
0028 1E10      btfss Txmt_Success
0029 2835      goto block_wrl_fail ; end
002A 0814      movf SubAddr,w
002B 008E      movwf DataByte      ; start from the first byte starting at DataPointer_
002C 2095      call SendData      ; send next byte
002D 1E10      btfss Txmt_Success
002E 2835      goto block_wrl_fail ; end
002F 0815      movf SrcPtr,w
0030 008E      movwf DataByte      ; start from the first byte starting at DataPointer_
0031 2095      call SendData      ; send next byte
0032 1E10      btfss Txmt_Success
0033 2835      goto block_wrl_fail ; failed, return 0 in WREG
0034 2837      goto block_wrl_pass  ; successful, return 1 in WREG
;
; return back to called routine from either_block_wrl_pass or block_wrl_fail
block_wrl_fail:
0035 205F      call TxmtStopBit      ; Issue a stop bit for slave to end transmission
0036 3400      retlw FALSE
block_wrl_pass:
0037 205F      call TxmtStopBit      ; Issue a stop bit for slave to end transmission
0038 3401      retlw TRUE
;
;*****
```

Software Implementation of I²C Bus Master

```

;
; I2C_WR_MEM_BYTE
;
; Some I2C devices like a EEPROM need to wait fo some time after every byte write (when entered into
; internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC, but in addition adds a delay
; after each byte. Some EEPROM memories (like Microchip's 24Cxx Series have on-chip data buffer),
; and hence this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB for a
; block of data and then insert a delay until the whole buffer is written.
; Parameters :
; BYTES_#of bytes starting from RAM pointer _SourcePointer_ (constant)
; SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
; Sub_Address_ Sub-address of Slave (constant)
;
; Sequence :
; S-SlVwA-A-(SubA+0)-A-D[0]-A-P
; Delay 1 mSec ; The user can chnage this value to desired delay
; S-SlVwA-A-(SubA+1)-A-D[1]-A-P
; Delay 1 mSec
; and so on until #of Bytes
;
; *****
; I2C_WR_BYTE_MEM MACRO _BYTES_, _SourcePointer_, _Sub_Address_
;
; variable i ; TEMP ???? : Assembler Does Not Support This
;
; i = 0
;
; .while (i < _BYTES_)
movf (_SourcePointer_ + i),w
movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
;
; *****

```

Software Implementation of I²C Bus Master

```
call i2c_byte_wr_sub ; write a byte of data at sub address
call Delay50uSec

    i++
    .endw

ENDM

; *****
; I2C_WR_MEM_BUF
;
; This Macro/Function writes #of_BYTES_ to an I2C memory device. However some devices, esp. EEPROMs must
; wait while the device enters into programming mode. But some devices have an onchip temp data hold buffer
; and is used to store data before the device actually enters into programming mode. For example, the 24C04
; series of Serial EEPROMs from Microchip have an 8 byte data buffer. So one can send 8 bytes of data at a
; time and then the device enters programming mode. The master can either wait until a fixed time and then
; retry to program or can continuously poll for ACK bit and then transmit the next Block of data for programming
;
; Parameters :
;   BYTES_           # of bytes to write to memory
;   SourcePointer_  Pointer to the block of data
;   SubAddress_     Sub-address of the slave
;   Device_BUF_SIZE_ The on chip buffer size of the i2c slave
;
; Sequence of operations
; I2C_SUB_WR operations are performed in loop and each time
; data buffer of BUF_SIZE is output to the device. Then
; the device is checked for busy and when not busy another
; block of data is written
;
; *****
```

Software Implementation of I²C Bus Master

```
I2C_WR_BUF_MEM MACRO _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_

variable i, j

if ( !_BYTES_)

    exitm

elif ( _BYTES_ <= _Device_BUF_SIZE_)

    I2C_WR_SUB_BYTES_, _SourcePointer_, _SubAddress_

    exitm

    else

        i = 0
        j = ( _BYTES_ / _Device_BUF_SIZE_)
        .while (i < j)

            I2C_WR_SUB _Device_BUF_SIZE_, ( _SourcePointer_ + i*_Device_BUF_SIZE_),
            ( _SubAddress_ + i*_Device_BUF_SIZE_)

            call _IsSlaveActive
            btfs _SlaveActive
            goto $-2

        .endw

        i++

        j = ( _BYTES_ - i*_Device_BUF_SIZE_)

            if (j)

                I2C_WR_SUB j, ( _SourcePointer_ + i*_Device_BUF_SIZE_), ( _SubAddress_ + i*_Device_BUF_SIZE_)


```

```

endif
endif
ENDM
;*****
;
; I2C_READ
;
; The basic MACRO/procedure to read a block message from a slave device
;
; Parameters :
;   _BYTES_      : constant : #of bytes to receive
;   _DestPointer_ : destination pointer of RAM (File Registers)
;
; Sequence :
;   S-SlVAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
; If last byte, then Master will NOT Acknowledge (send NACK)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and
;        10 or 8 bit mode addressing must be set
;*****
I2C_READ MACRO _BYTES_, _DestPointer_

    movlw    (_BYTES_ -1)
    movwf   tempCount      ; -1 because, the last byte is used out of loop
    movlw   _DestPointer_
    movwf   fsr            ; FIFO destination address pointer

    call   i2c_block_read
ENDM

```

Software Implementation of I²C Bus Master

```
_i2c_block_read:
0039 2057      call    TxmtStartBit      ; send START bit
003A 1491      bsf     Slave_RW          ; set for read operation
003B 1111      bcf     Last_Byte_Rcv    ; not a last byte to rcv
003C 206B      call    Txmt_Slave_Addr   ; if successful, then _Txmt_Success bit is set
003D 1A10      btfs    Txmt_Success     ; end
003E 2841      goto   block_rdl_loop   ; end
003F 205F      call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0040 3400      retlw  FALSE           ; Error : may be device not responding
;
_block_rdl_loop:
0041 20CC      call    GetData         ; start receiving data, starting at Destination Pointer
0042 080E      movf   DataByte,w      ;
0043 0080      movwf  indf             ;
;
0044 0A84      incf   fsr              ;
0045 0B96      decfsz tempCount      ;
0046 2841      goto   block_rdl_loop  ; loop until desired bytes of data transmitted to slave
0047 1511      bsf   Last_Byte_Rcv   ; last byte to rcv, so send NACK
0048 20CC      call    GetData         ;
0049 080E      movf   DataByte,w     ;
004A 0080      movwf  indf           ;
004B 205F      call    TxmtStopBit    ; Issue a stop bit for slave to end transmission
004C 3401      retlw  TRUE           ;
;
;
; I2C_READ_SUB
; This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address Between the
; sub-addresses write & the following reads, a STOP condition is not issued and a "REPEATED START" condition is
; used so that an other master will not take over the bus, and also that no other master will overwrite the sub-
; address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a
```


Software Implementation of I²C Bus Master

```

;
; I2C_READ_STATUS
;
; This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can send a status byte
; upon reception of a control byte. This is basically same as I2C_READ_MACRO for reading a single byte.
;
; For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory data at the
; current address location
; On success WREG = 1 else = 0
;
; *****
; I2C_READ_STATUS_MACRO _DestPointer_
;
call TxmtStartBit ; send START bit
bsf Slave_RW ; set for read operation
call Txmt_Slave_Addr ; if successful, then Txmt_Success bit is set
btfsc Txmt_Success
goto byte_rdl_loop ; read a byte
call TxmtStopBit ; Issue a stop bit for slave to end transmission
retlw FALSE ; Error : may be device not responding
;
; byte_rdl_loop:
bsf Last_Byte_Rcv ; last byte to rcv, so send NACK
call GetData
movf DataByte,w
movwf DestPointer_
call TxmtStopBit ; Issue a stop bit for slave to end transmission
btfsc Rcv_Success
retlw FALSE
retlw TRUE
ENDM

```



```

; I2C_WR_SUB_RD
;
; This macro writes a block of data from SourcePointer of length _COUNT1_ to a
; slave at sub-address and then Reads a block of length _COUNT2_ to
; destination address pointer
;
;
; Message Structure :
; S-SlW-A-SubA-A-D1[0]-A-....-A-D1[N-1]-A-S-SlVR-A-D2[0]-A-.....A-D2[M-1]-N-P
;
; Parameters :
;   _COUNT1_      Length Of Source Buffer
;   _SourcePointer_ Source Pointer Address
;   _Sub_Address_  The Sub Address Of the slave
;   _COUNT2_     The length of Destination Buffer
;   _DestPointer_  The start address of Destination Pointer
;
; *****

```

```

I2C_WR_SUB_RD MACRO _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_

movlw (COUNT1_ + 1)
movwf tempCount
movlw (SourcePointer_ - 1)
movwf fsr
movf indf,w
movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ - 1)
movlw Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ - 1)
call i2c_block_write ; write _BYTES_+1 block of data
;
movf StoreTemp_1,w
movwf (SourcePointer1_ - 1) ; restore contents of (_SourcePointer_ - 1)

```



Software Implementation of I²C Bus Master

```
I2C_WR_COM_WR      MACRO      _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_

    COUNT1_
    movlw tempCount
    movwf SourcePointer1_
    movwf fsr
    call i2c_block_write
    ;
    ; First block sent, now send 2nd block of data
    ;
    movlw _COUNT2_
    movwf tempCount
    movlw _SourcePointer2_
    movwf fsr
    call _block_wrl_loop
    ;
    call TxmtStopBit ; End of Double buffer txmt

ENDM

;*****
;
; ***** I2C Low Level Routines Here *****
;*****

include "i2c_low.asm"
;*****
;
; Low Level I2C Routines
;
; Single Master Transmitter & Single Master Receiver Routines
; These routines can very easily be converted to Multi-Master System
; when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
```


Software Implementation of I²C Bus Master

```
0059 1406      bsf      SCL      ; clock is high
              ;
              ; Setup time for a REPEATED START condition (4.7 uS)
              ;
005A 210D      call     Delay40uSec ; only necessary for setup time
              ;
005B 1086      bcf      SDA      ; give a falling edge on SDA while clock is high
              ;
005C 210B      call     Delay7uSec  ; only necessary for START HOLD time
              ;
005D 1410      bsf      Bus_Busy   ; on a start condition bus is busy
              ;
005E 0008      return

;*****
;      Send Stop Bit
;
;*****

TxmtStopBit:
005F 1683      bsf      rp0      ; select page 1
0060 1006      bcf      SCL
0061 1086      bcf      SDA      ; set SDA low
0062 1406      bsf      SCL      ; Clock is pulled up
0063 210D      call     Delay40uSec ; Setup Time For STOP Condition
0064 1486      bsf      SDA      ; give a rising edge on SDA while CLOCK is high
              ;
              if _ENABLE_BUS_FREE_TIME
              ; delay to make sure a START bit is not sent immediately after a STOP,
              ; ensure BUS Free Time tBUF
              ;
0065 210B      call     Delay7uSec
```


Software Implementation of I²C Bus Master

```
006B 1390      bcf      ACK_Error      ; reset Acknowledge error bit
006C 1C11      brfss     10BitAddr
006D 2886      goto     SevenBitAddr
;
006E 1C91      brfss     Slave_RW
006F 287D      goto     TenBitAddrWR
;
; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then
; Repeated Start and then Hi Byte Only for read operation
;
TenBitAddrRd:
0070 1091      bcf      Slave_RW      ; temporarily set for WR operation
0071 207D      call     TenBitAddrWR
0072 1E10      brfss     Txmt_Success ; skip if successful
0073 3400      retlw    FALSE
0074 2057      call     TxmtStartBit ; send A REPEATED START condition
0075 1491      bsf      Slave_RW      ; For 10 bit slave Read
;
0076 080D      movf     SlaveAddr+1,W
0077 008E      movwf    DataByte
0078 140E      bsf     DataByte,LSB  ; Read Operation
0079 2095      call     SendData      ; send ONLY high byte of 10 bit addr slave
007A 288C      goto     AddrSendTest  ; 10 Bit Addr Send For Slave Read Over
;
; if successfully transmitted, expect an ACK bit
;
007B 1E10      brfss     Txmt_Success  ; if not successful, generate STOP & abort transfer
007C 2890      goto     AddrSendFail
;
TenBitAddrWR:
```


Software Implementation of I²C Bus Master

```

;
;
TxmtByte:
0096 080E      movf  DataByte,w
0097 0093      movwf DataByteCopy
; make copy of DataByte
0098 151C      bsf  Txmt_Progress
; set Bus status for txmt progress
0099 121C      bcf  Txmt_Success
; reset status bit
009A 3008      movlw 0x08
009B 008F      movwf BitCount
009C 1683      bsf  rp0
if _CLOCK_STRETCH_CHECK
; set RTCC to INT CLK timeout for 1 mSec
; do not disturb user's selection of RPUB in OPTION Register
;
009D 0801      movf  option,w
009E 39C3      andlw OPTION_INIT
; defined in I2C.H header file
009F 0081      movwf option
endif

TxmtNextBit:
00A0 0064      clrwdt
; clear WDT, set for 18 mSec
00A1 1006      bcf  SCL
00A2 0D93      rlf  DataByteCopy
; MSB first, Note DataByte Is Lost
00A3 1086      bcf  SDA
00A4 1803      btfsz c
00A5 1486      bsf  SDA
00A6 210B      call Delay47uSec
; guarantee min LOW TIME tLOW & Setup time
00A7 1406      bsf  SCL
; set clock high, check if clock is high, else
; clock being stretched
00A8 210E      call Delay40uSec
; guarantee min HIGH TIME tHIGH
if _CLOCK_STRETCH_CHECK
00A9 1283      bcf  rp0
00AA 0181      clrf rtcc
; clear RTCC
00AB 110B      bcf  rtif
; clear any pending flags
00AC 168E      bsf  rtie
; enable RTCC Interrupt

```

Software Implementation of I²C Bus Master

```
00AD 1391      bcf     TIME_OUT_      ; reset timeout error flag
                Check_SCL_1:
00AE 1B91      btfsz   TIME_OUT_      ; if RTCC timeout or Error then Abort & return
00AF 28FC      goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00B0 1283      bcf     rp0
00B1 1C06      btfsz   SCL
00B2 28AE      goto    Check_SCL_1 ; if clock not being stretched, it must be high
00B3 128B      bcf     rtie      ; loop until SCL high or RTCC timeout interrupt
00B4 1683      bsf     rp0      ; Clock good, disable RTCC interrupts

                endif
00B5 0B8F      decfsz  BitCount
00B6 28A0      goto    TxmtNextBit ;

                ; Check For Acknowledge
                ;
00B7 1006      bcf     SCL      ; reset clock
00B8 1486      bsf     SDA      ; Release SDA line for Slave to pull down
00B9 210B      call    Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
00BA 1406      bsf     SCL      ; clock for slave to ACK
00BB 210D      call    Delay40uSec ; guarantee min HIGH TIME tHIGH
00BC 1283      bcf     rp0      ; select PAGE 0 to test PortB pin SDA
00BD 1886      btfsz   SDA      ; SDA should be pulled low by slave if OK
00BE 28C5      goto    TxmtErrorAck ;

00BF 1683      bsf     rp0
00C0 1006      bcf     SCL      ; reset clock

00C1 1110      bcf     Txmt_Progress ; reset TXMT bit in Bus Status
00C2 1610      bsf     Txmt_Success ; transmission successful
00C3 1390      bcf     ACK_Error ; ACK OK
00C4 0008      return
                TxmtErrorAck:
```


Software Implementation of I²C Bus Master

```

00EA 1886      btfscc SDA
00EB 1403      bsf c
;
00EC 0D8E      rlf DataByte
00ED 0B8F      decfsz BitCount
00EE 28D5      goto RcvNextBit
;
; Generate ACK bit if not last byte to be read,
; if last byte Generate NACK
; do not send ACK on last byte, main routine will send a STOP bit
;
00EF 1683      bsf rp0
00F0 1006      bcf SCL
00F1 1086      bcf SDA
00F2 1911      btfscc Last_Byte_Rcv
00F3 1486      bsf SDA
00F4 210B      call Delay47uSec
00F5 1406      bcf SCL
00F6 210D      call Delay40uSec
; RcvEnd:
00F7 1006      bcf SCL
00F8 1190      bcf Rcv_Progress
00F9 1690      bsf Rcv_Success
00FA 1390      bcf ACK_Error
00FB 0008      return
;
if _CLOCK_STRETCH_CHECK
;*****
; Fatal Error On I2C Bus
;
; Slave pulling clock for too long or if SCL Line is stuck low.
; This occurs if during Transmission, SCL is stuck low for period longer
; than approx. 1mS and RTCC times out (approx 4096 cycles : 256 * 16 -
; prescaler of 16).

```


Software Implementation of I²C Bus Master

```
*****
Bus_Fatal_Error:
; disable RTCC Interrupt
;
rtcc ; disable RTCC interrupts, until next TXMF try

RELEASE_BUS

00FD 1683 ; Set the Bus_Status Bits appropriately
00FD 1486 ;
00FE 1406 ;

0100 1490 ; Abort ; transmission was aborted
0101 1710 ; Fatal_Error ; FATAL Error occurred
0102 1110 ; Txmt_Progress ; Transmission Is Not in Progress
0103 1210 ; Txmt_Success ; Transmission Unsuccessful
;
0104 205F call TxmtStopBit ; Try sending a STOP bit, may be not successful
;
0105 0008 return
;
*****
endif
*****
; General Purpose Delay Routines
;
; Delay4uS is wait loop for 4.0 uSec
; Delay7uS is wait loop for 4.7 uSec
; Delay50uS is wait loop for 5.0 uSec
;
```

Software Implementation of I²C Bus Master

```
*****
;
;
Delay50uSec:
movlw    (_50uS_Delay-5)/3 + 1)
DlyK
movwf    DelayCount
decfsz   DelayCount
goto     $-1
return
;
Delay47uSec:
movlw    ((47uS_Delay-8)/3 + 1)
goto     DlyK
;
Delay40uSec:
movlw    ((_40uS_Delay-8)/3 + 1)
goto     DlyK
;
*****
endif
*****
;
;
ReadSlave1:
;
; EEPROM (24C04) may be in write mode (busy), check for ACK by sending a
; control byte
LOAD_ADDR_8 _Slave_1_Addr
010F 1011
010F 30A0
```

```
0110 008C
        wait1:
        I2C_TEST_DEVICE

0112 2011
0113 1F11 btffs SlaveActive ; See If slave is responding
0114 2912 goto wait1 ; if stuck for ever, recover from WDT, can use other schemes
0115 0064 clrwtd

        I2C_READ_SUB 8, DataBegin+1, 0x50

0116 1091
0116 2057
0117 206B
0119 3050
0119 008E
011A 2095
011C 3007
011C 0096
011D 3021
011E 0084
0120 2039

        ;
        ; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
        ;
        LOAD_ADDR_8 _Slave_2_Addr

0121 1011
0121 30AC
0122 008C
```


Software Implementation of I²C Bus Master

```
wait3:
I2C_TEST_DEVICE

0137 2011
0138 1F11 btffs SlaveActive ; See If slave is responding
0139 2937 goto wait3 ; if stuck for ever, recover from WDT, can use other schemes
013A 0064 clrwdt
I2C_READ_SUB 8, DataBegin, 0

013B 1091
013B 2057
013C 206B
013E 3000
013E 008E
013F 2095
0141 3007
0141 0096
0142 3020
0143 0084
0145 2039

0146 0008
return
;
;*****
; ; Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
;*****
FillDataBuf:
```

Software Implementation of I²C Bus Master

```

0147 3000      movlw 0x00      ; start address location of EEPROM array
0148 00A0      movwf DataBegin ; 1st byte of data to be sent is start address
0149 3021      movlw DataBegin+1 ; data starts following address (RAM Pointer)
014A 0084      movwf fsr
014B 3008      movlw 8      ; fill RAM with 8 bytes , this data is written to
                                ; EEPROM (slave)
014C 009A      movwf byteCount
014D 3055      movlw 0x55    ; pattern to fill with is 0x55 & 0xAA
014E 009B      movwf HoldData
                                XI:
014F 099B      comf HoldData
0150 081B      movf HoldData,w
0151 0080      movwf indf
0152 0A84      incf fsr      ; point to next location
0153 0B9A      decfsz byteCount
0154 294F      goto XI
0155 0008      return

;
;*****
;
; Main Routine (Test Program)
;
; SINGLE MASTER, MULTIPLE SLAVES
;
;*****
Start:
0156 204D      call InitI2CBus_Master ; initialize I2C Bus
0157 178B      bsf gie          ; enable global interrupts
                                ;
0158 2147      call FillDataBuf ; fill data buffer with 8 bytes of data (0x55, 0xAA)
                                ;

```

Software Implementation of I²C Bus Master

```
; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit
; Addr
; Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
;
0159 1810      btfscc Bus_Busy      ; is Bus Free, ie. has a start & stop bit been
; detected (only for multi master system)
015A 2959      goto    $-1        ; a very simple test, unused for now

LOAD_ADDR_8   _Slave_1_Addr

015B 1011
015B 30A0
015C 008C

I2C_WR        0x09, DataBegin

015E 3009
015E 0096
015F 3020
0160 0084
0162 2019
0162 205F

;
; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
;
0164 1810      btfscc Bus_Busy      ; is Bus Free, ie. has a start & stop bit been
; detected (only for multi master system)
0165 2964      goto    $-1        ; a very simple test, unused for now
```

Software Implementation of I²C Bus Master

```
LOAD_ADDR_8    __Slave_2_Addr
0166 1011
0166 30AC
0167 008C

I2C_WR_SUB    0x08, DataBegin+1, 0x30

0169 3009
0169 0096
016B 3020
016B 0084
016D 0800
016D 0097
016E 3030
016F 0080
0171 2019
0172 0817
0172 00A0
0174 205F
0175 210F    call    ReadSlave1
; read a byte from slave from current address
;

LOAD_ADDR_8    __Slave_3_Addr
0176 1011
0176 30D6
0177 008C
0179 30CC    movlw  0xCC
017A 00A0    movwf DataBegin
I2C_WR_SUB    0x01,DataBegin, 0x33

017B 3002
```


Software Implementation of I²C Bus Master

```
017B 0096  
017D 301F  
017D 0084  
017F 0800  
017F 0097  
0180 3033  
0181 0080  
0183 2019  
0184 0817  
0184 009F  
0186 205F  
0187 2134  
  
call ReadSlave3 ; Read From Slave PIC  
;  
  
self clrwdt  
goto self  
;  
;*****
```

END

Software Implementation of I²C Bus Master

Software Implementation of Asynchronous Serial I/O

INTRODUCTION

The PIC16CXX series from Microchip Technology, Inc., are mid-range, high performance EPROM based 8-bit microcontrollers. Some of the members of this series (like PIC16C71 and PIC16C84) do not have on-chip hardware asynchronous serial port. This application note describes the Interrupt driven Software implementation of Asynchronous Serial I/O (Half Duplex RS-232 Communications) using PIC16CXX microcontrollers. These microcontrollers can operate at very high speeds with a minimum of 250 ns cycle time (with input clock frequency of 16 MHz). To test the RS-232 routines, a simple Digital Volt Meter (DVM)/Analog Data Acquisition Systems has been implemented using PIC16C71 in which upon reception of a command from host (IBM™ PC), an 8-bit value of the selected A/D channel is transmitted back to host.

IMPLEMENTATION

A half duplex Interrupt driven software implementation of RS-232 communications using PIC16C71 is described in detail below. The transmit pin used in the example code is RB7 and receive pin is connected to RTCC/RA4 pin (see Figure 2). Of course these pins are connected with appropriate voltage translation to/from RS-232/CMOS levels. The voltage translation is given described with schematics in the hardware section of this application note.

Transmit Mode

The transmit mode in software is quite straight forward to implement using interrupts. Once the input clock frequency and baud rate is known, the number of clock cycles per bit can be computed. The on-chip Real Time Clock Counter (RTCC) along with the prescaler can be used to generate interrupt on RTCC overflow. This RTCC overflow interrupt can be used as timing to send each bit. The Input clock frequency (“_CkIn”) and the Baud Rate (“_BaudRate”) are programmable by the user and the RTCC time-out value (the period for each bit) is computed at assembly time. Whether the prescaler must be assigned to RTCC or not is also determined at assembly time. This computation is done in the header file “rs232.h”. Note that very high speed transmissions can be obtained if transmission is done with pulserly software delays instead of interrupt driven. However the processor will be totally dedicated to this job.

Transmission of a byte is performed by calling “PutChar” function and the data byte in the “TxReg” is transmitted out. Before calling this function (“PutChar”), the data must be loaded into TxReg and also made sure that serial port is free. The serial port is free when both _txmtProgress and _rcvOver bits are cleared (see description of these bits in the Serial Status/Control Reg table given later).

Summary of “PutChar” function :

- 1) Make sure _txmtProgress & _rcvOver bits are cleared
- 2) Load TxReg with data to be transmitted
- 3) CALL PutChar function

Receive Mode

The reception mode implementation is slightly different from the transmit mode. Unlike the transmit Pin (TX pin in the example code is RB7, but could be any I/O pin), the receive pin (RX Pin) must be connected to RTCC/RA4 Pin. This is because in reception, the Start Bit which is asynchronous in nature, must be detected. To detect the start bit, when put in Reception mode, the RTCC module is configured to counter mode. The OPTION register is configured so that RTCC module is put in counter mode (increment on external clock on RTCC/RA4 Pin) and set to increment on falling edge on RTCC/RA4 pin with no prescaler assigned. After this configuration setup, RTCC (File Reg 1) is loaded with 0xFF. A falling edge on RTCC Pin will make RTCC roll over from 0xFF to 0x00, thus generating an interrupt indicating a Start Bit. The RTCC/RA4 pin is sampled again to make sure the transition on RTCC is not a glitch. Once the start bit has been detected, the RTCC module is reconfigured to increment on internal clock and the prescaler is assigned to it depending on input master clock frequency and the baud rate (configured same way as the transmission mode).

The software serial port is put in reception mode when a call is made to function “GetChar”. Before calling this function make sure serial port is free (i.e. _txmtProgress and _rcvOver status bits must be 0). On completion of reception of a byte, the data is stored in RxReg and _rcvOver bit is set to 0.

Summary of “GetChar” function:

- 1) Make sure _txmtProgress & _rcvOver bits are cleared
- 2) CALL GetChar function
- 3) The received Byte is in TxReg after _rcvOver bit is cleared

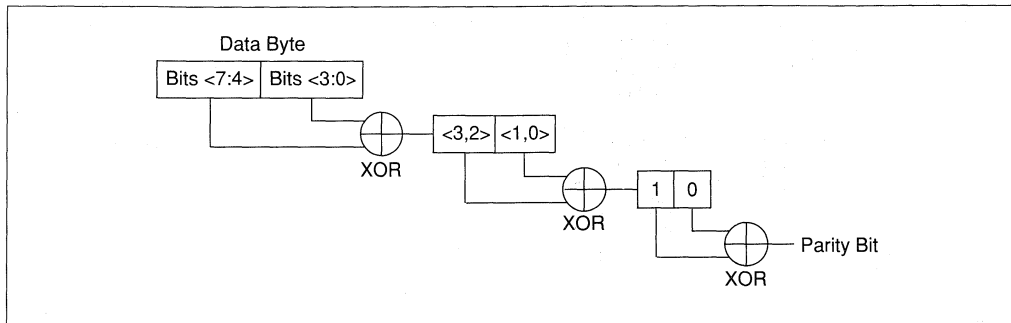
Software Implementation of Asynchronous Serial I/O

Parity Generation

Parity can be enabled at assembly time by setting “_PARITY_ENABLE” flag to TRUE. If enabled, the parity can be set to either EVEN or ODD parity. In transmission mode, if parity is enabled, the parity bit is computed and transmitted as the 9th bit. On reception, the parity is computed on the received byte and compared to the 9th bit received. If a match does not occur the parity error bit is set in the RS-232 Status/Control

Register (_ParityErr bit of SerialStatus reg). The parity bit is computed using the algorithm shown in Figure 1. This algorithm is highly efficient using PIC16CXX’s SWAPF and XORWF instructions (with ability to have the destination as either file register itself or W register) and the sub-routine (called “GenParity”) is in file “txmtr.asm”.

FIGURE 1 - AN EFFICIENT PARITY GENERATION SCHEME IN SOFTWARE



Assembly Time Options

The firmware is written as a general purpose routines and the user must specify the following parameters before assembling the program. The Status/Control register is also described below:

TABLE 1 - LIST OF ASSEMBLY TIME OPTIONS

_CkIn	Input clock frequency of the processor.
_BaudRate	Desired Baud Rate. Any valid value can be used. The highest Baud Rate achievable depends on Input Clock Freq. 600 to 4800 Baud was tested using 4 MHz Input Clock. 600 to 19200 Baud was tested using 10 MHz Input Clock. Higher rates can be obtained using higher Input Clock Frequencies. Once the _BaudRate & _CkIn are specified, the program automatically selects all the appropriate timings.
_DataBits	Can specify 1 to 8 data bits.
_StopBits	Limited to 1 Stop Bit. Must be set to 1.
_PARITY_ENABLE	Parity Enable Flag. Set it to TRUE or FALSE. If PARITY is used, then set it to TRUE, else FALSE. See “_ODD_PARITY” flag description below.
_ODD_PARITY	Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else mEVEN Parity Scheme is used. This Flag is ignored if _PARITY_ENABLE is set to FALSE.
_USE_RTSCS	RTS & CTS Hardware handshaking signals. If set to FALSE, no hardware handshaking is used. If set to TRUE, RTS & CTS use up 2 I/O Pins of PortB.

Software Implementation of Asynchronous Serial I/O

TABLE 2 - BIT ASSIGNMENTS OF SERIAL STATUS/CONTROL REGISTER ("SERIALSTATUS" REG)

Bit #	Name	Description
0	_txmtProgress	1 = Transmission in progress. 0 = Transmission line free.
1	_txmtEnable	Set this bit to 1 on initialization to enable transmission. This bit may be used to abort a transmission. The transmission is aborted if in the middle of a transmission (i.e. when _txmtProgress bit is 1) _txmtEnable bit is set to 0. This bit gets automatically set when PutChar function is called.
2	_rcvProgress	1 = Middle of a byte reception. 0 = Reception of a byte (in RxReg) is complete and is set to 1 when a valid start bit is detected in reception mode.
3	_rcvOver	1 = Completion of reception of a byte. The user's code can poll this bit after calling "GetChar" function and check to see if it is set. When set, the received byte is in RxReg. Other status bits should also be checked for any reception errors.
4	_ParityErr	1 = Parity error on reception (irrespective of Even Or Odd parity chosen). Not applicable if No Parity is used.
5	_FrameErr	1 = Framing error on reception.
6		Unused
7	_parityBit	The 9th bit of transmission or reception. In transmission mode, the parity bit of the byte to be transmitted is set in this bit. In receive mode, the 9th bit (or parity bit) received is stored in this bit. Not Applicable if no parity is used.

Software Implementation of Asynchronous Serial I/O

Hardware

The hardware is primarily concerned with voltage translation from RS-232 to CMOS levels and vice versa. Three circuits are given below and the user may choose which ever best suits his application. The primary difference between each solution is cost versus number of components. Circuits in Figure 3 and 4 are very low cost but have more components than the circuit in Figure 2. The circuit in Figure 2 interfaces to RS-232 line using a single chip (MAX-232) and single +5V supply. The circuit in Figure 3 is a low cost RS-232 Interface but requires two chips and a single +5V supply source.

Figure 4 shows a very low cost RS-232 Interface to an IBM PC with no external power requirements. The circuit draws power from RS-232 line (DTR) and meets the spec of drawing power less than 5mA. This requires that the host to communicate must assert DTR high and RTS low. The power is drawn from DTR line and this requires that DTR to be asserted high and must be at least 7V. The negative -5 to -10 V required by LM339 is drawn from RTS line and thus the host must assert RTS low. This circuit is possible because of the low current consumption of PIC16C71 (typical 2 mA).

FIGURE 2 - SINGLE CHIP FOR RS-232 INTERFACE (SINGLE +5V SUPPLY)

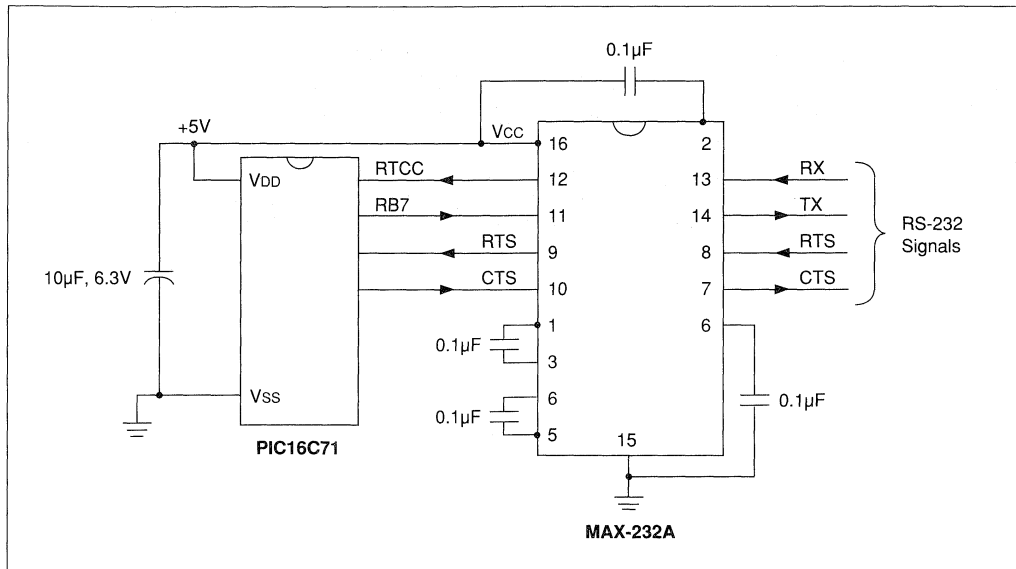
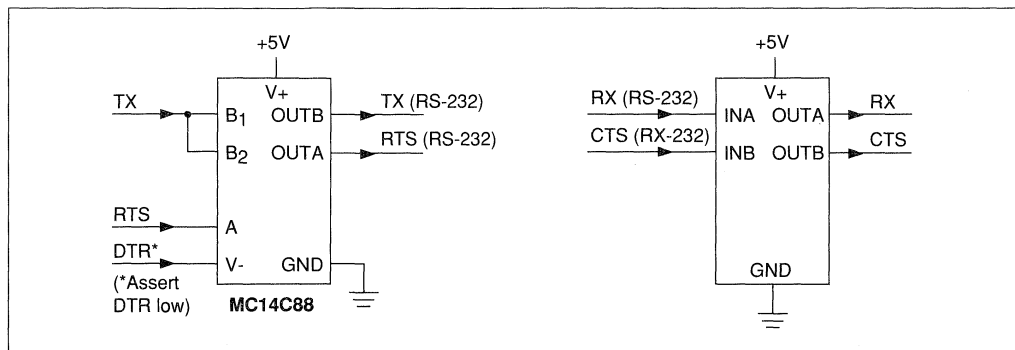
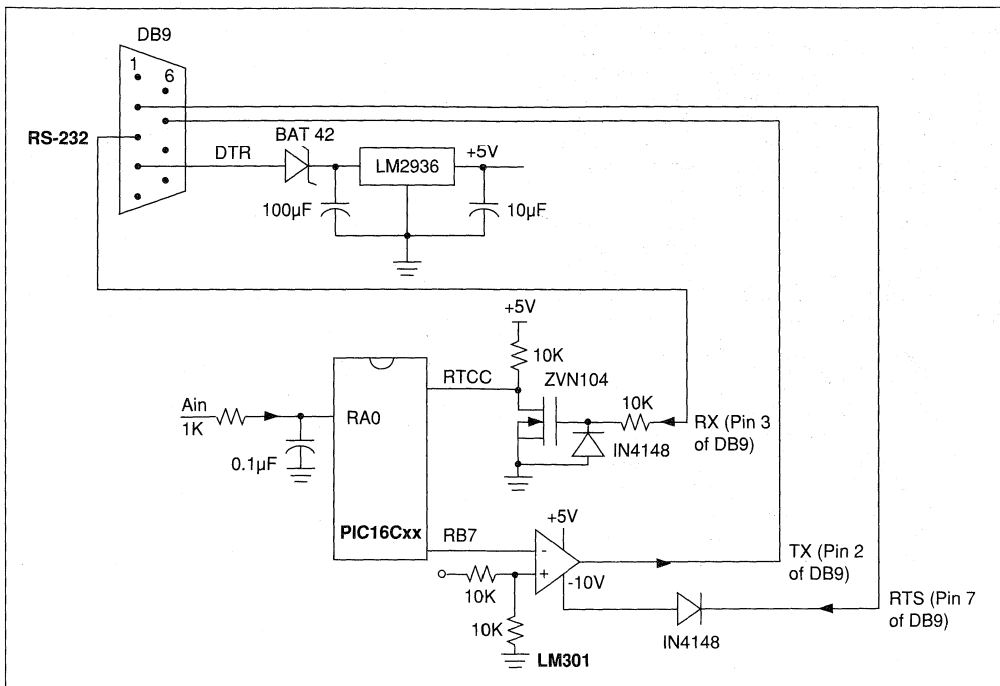


FIGURE 3 - LOW COST RS-232 INTERFACE (TWO CHIPS, SINGLE +5V SUPPLY)



Software Implementation of Asynchronous Serial I/O

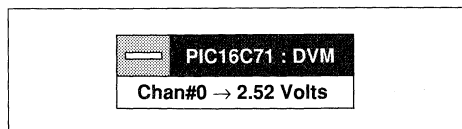
FIGURE 4 - LOW COST, LOW POWER RS-232 INTERFACE (POWER SUPPLIED BY RS-232 LINES)



Test Program

To test the transmission and reception modules, a main program is written in which the PIC16C71 waits to receive a command from a host through the RS-232. On reception of a byte (valid commands are 0x00, 0x01, 0x02 & 0x03), the received byte is treated as the PIC16C71's A/D channel number and the requested channel is selected, an A/D conversion is started and when the conversion is complete (in about 20 uS) the digital data (8 bits) are transmitted back to the host. A Microsoft Windows program running on an IBM PC/AT was written to act as a host and collect the A/D data from PIC16C71 via an RS-232 port. The Windows program (DVM.EXE) runs as a background job and displays the A/D data in a small window (similar to the CLOCK program that comes with MS Windows). The windows program and the PIC16C71 together act like a data acquisition system or a digital volt meter (DVM). The block diagram of the system is shown in Figure 2. The input clock frequency is fixed at 4 MHz and RS-232 parameters are set to 1200 Baud, 8-bits, 1 Stop Bit and No Parity. The program during development stage was also tested at 1200, 2400, 4800 Baud Rates @ 4 MHz Input Clock and up to 19200 Baud @ 10 MHz input clock frequency (all tests were performed with No Parity, Even Parity and Odd Parity at 8 and 7 Data Bits).

FIGURE 5 - MS WINDOWS PROGRAM FETCHING A/D DATA FROM PIC16C71 VIA RS-232



Software Implementation of Asynchronous Serial I/O

Source Code

The PIC16CXX source code along with the MS Windows DVM Program (executable running on an IBM PC/AT under MS Windows 3.1 or higher) is available on Microchip's BBS. The assembly code for PIC16CXX must be assembled using Microchip's Universal Assembler MPASM. The code cannot be assembled using the older assemblers without significant modifications. It is suggested that user's who do not have the new assembler MPASM, must change to the new version.

The MS Windows Program (DVM.EXE) runs under MS Windows 3.1 or higher. The program does not have any menus and shows up as a small window displaying A/D Data and runs as a background job. There are a few command line options and are described below :

- Px : x is the comm port number (e.g. - P2 selects COM2). Default is COM1
- Cy : y is the number of A/D channels to display. Default is one channel (channel #1)
- Sz : z is a floating point number that represents the scaling factor (For example - S5.5 would display the data as $5.5 * \text{A/D} / 256$). The default value is 5.0 volts. -S0 will display the data in raw format without any scaling.

<i>AUTHOR: Amar Palacherla, Logic Products Division</i>

Appendix A - RS232.H

```

NOLIST
;*****
; RS-232 Header File
;*****
; PIC16C6X/7X/8X
;*****
ClkOut      equ    (ClkIn >> 2)      ; Instruction Cycle Freq = CLKIN/4
;
CyclesPerBit set    (ClkOut/_BaudRate)
tempCompute set    (CyclesPerBit >> 8)
;
;*****
; Auto Generation Of Prescaler & Rtcc Values
; Computed during Assembly Time
;*****
; At first set Default values for RtccPrescale & RtccPreLoad
;
RtccPrescale set    0
RtccPreLoad  set    CyclesPerBit
UsePrescale  set    FALSE

if (_tempCompute >= 1)
RtccPrescale set    0
RtccPreLoad  set    (CyclesPerBit >> 1)
UsePrescale  set    TRUE
endif

if (tempCompute >= 2)
RtccPrescale set    1
RtccPreLoad  set    (CyclesPerBit >> 2)
endif

if (tempCompute >= 4)
RtccPrescale set    2
RtccPreLoad  set    (CyclesPerBit >> 3)
endif

if (tempCompute >= 8)
RtccPrescale set    3
RtccPreLoad  set    (CyclesPerBit >> 4)
endif

```

Software Implementation of Asynchronous Serial I/O

```
if (tempCompute >= 16)
    set RtccPrescale 4
    set RtccPreLoad (CyclesPerBit >> 5)
endif

if (tempCompute >= 32)
    set RtccPrescale 5
    set RtccPreLoad (CyclesPerBit >> 6)
endif

if (tempCompute >= 64)
    set RtccPrescale 6
    set RtccPreLoad (CyclesPerBit >> 7)
endif

if (tempCompute >= 128)
    set RtccPrescale 7
    set RtccPreLoad (CyclesPerBit >> 8)
endif

;
if (RtccPrescale == 0) && (RtccPreLoad < 60))
    messg "Warning : Baud Rate May Be Too High For This Input Clock"
endif

; Compute RICC & Presclaer Values For 1.5 Times the Baud Rate for Start Bit Detection
;

SBitCycles set (ClkOut/_BaudRate) + ((_ClkOut/4)/_BaudRate)
tempCompute set (SBitCycles >> 8)

SBitPrescale set 0
SBitRtccLoad set _SbitCycles

if (tempCompute >= 1)
    set SBitPrescale 0
    set SBitRtccLoad (SBitCycles >> 1)
endif

if (tempCompute >= 2)
    set SBitPrescale 1
    set SBitRtccLoad (SBitCycles >> 2)
endif
```

Software Implementation of Asynchronous Serial I/O

```
if (tempCompute >= 4)
    set 2
    SBitPrescale (SBitCycles >> 3)
    set SBitRtccLoad
endif

if (tempCompute >= 8)
    set 3
    SBitPrescale (SBitCycles >> 4)
    set SBitRtccLoad
endif

if (tempCompute >= 16)
    set 4
    SBitPrescale (SBitCycles >> 5)
    set SBitRtccLoad
endif

if (tempCompute >= 32)
    set 5
    SBitPrescale (SBitCycles >> 6)
    set SBitRtccLoad
endif

if (tempCompute >= 64)
    set 6
    SBitPrescale (SBitCycles >> 7)
    set SBitRtccLoad
endif

if (tempCompute >= 128)
    set 7
    SBitPrescale (SBitCycles >> 8)
    set SBitRtccLoad
endif

;
;*****
;
;#define _Cycle_Offset1 i8
LOAD_RTCC MACRO K, Prescale
    if (UsePrescale == 0)
        movlw -K + _Cycle_Offset1
    else
        movlw -K + (_Cycle_Offset1 >> (Prescaler+1))
    endif
    movwf _rtcc
; Note that Prescaler is cleared when RTCC is written
; Re Load RTCC init value + INT Latency Offset
;*****
```

Software Implementation of Asynchronous Serial I/O

```
*****
ENDM
LOAD_BITCOUNT MACRO
    movlw _DataBits+_StopBits
    movwf BitCount
    if _PARITY_ENABLE
        movlw _2
        movwf ExtraBitCount
    endif
ENDM
;
; *****
; Pin Assignments
; *****
#define RX_MASK 0x10 ; RX pin is connected to RA4, ie. bit 4
#define RX_Pin _porta,4 ; RX Pin : RA4
#define RX_RxTemp,4
#define TX _portb,7 ; TX Pin , RB7
#define RTS _portb,5 ; RTS Pin, RB5, Output signal
#define CTS _portb,6 ; CTS Pin, RB6, Input signal

#define txmtProgress SerialStatus,0
#define txmtEnable SerialStatus,1

#define rcvProgress SerialStatus,2
#define rcvOver SerialStatus,3
#define ParityErr SerialStatus,4
#define FrameErr SerialStatus,5

#define parityBit SerialStatus,7
; *****
_OPTION_SBIT set 0x38 ; Increment on Ext Clock (falling edge), for START Bit Detect
    if UsePrescale
        _OPTION_INIT set 0x00 ; Prescaler is used depending on Input Clock & Baud Rate
    else
        _OPTION_INIT set 0x08
    endif
*****
```

Software Implementation of Asynchronous Serial I/O

```
CBLOCK 0x0C
TxReg
RxReg
    RxTemp
    SerialStatus
    BitCount
    if _PARITY_ENABLE
        ExtraBitCount
    endif
    SaveWReg
    SaveStatus
temp1, temp2
ENDC
; *****
; Transmit Data Holding/Shift Reg
; Rcv Data Holding Reg
; Txmt & Rcv Status/Control Reg
; Parity & Stop Bit Count
; temp hold reg of WREG on INT
; temp hold reg of STATUS Reg on INT
; *****
```

LIST

Software Implementation of Asynchronous Serial I/O

```

; Usage :
;
; An example is given in the main program on how to Receive & Transmit Data
; In the example, the processor waits until a command is received. The command is interpreted
; as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
; is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
;
;
; The RS-232 Control/Status Reg's bits are explained below :
;
; "SerialStatus" : RS-232 Status/Control Register
;
; Bit 0 : txmtProgress (1 if transmission in progress, 0 if transmission is complete)
; After a byte is transmitted by calling "putChar" function, the
; user's code can poll this bit to check if transmission is complete.
; This bit is reset after the STOP bit has been transmitted
; Set this bit to 1 on initialization to enable transmission.
; This bit can be used to Abort a transmission while the transmitter
; is in progress (i.e when _txmtProgress = 1)
; Indicates if the receiver is in middle of reception.It is reset when
; a byte is received.
; This bit indicates the completion of Reception of a Byte. The user's
; code can poll this bit after calling "GetChar" function. Once "GetChar"
; function is called, this bit is 0 and is set to 1 after reception of
; a complete byte (parity bit if enabled & stop bit)
; Bit 4 : ParityErr A 1 indicates Parity Error on Reception (for both even & odd parity)
; Bit 5 : FrameErr A 1 indicates Framing Error On Reception
;
; Bit 6 : unused_ Unimplemented Bit
;
; Bit 7 : parityBit The 9 th bit of transmission or reception (status of PARITY bit
; if parity is enabled)
;
; To Transmit A Byte Of Data :
; 1) Make sure _txmtProgress & _rcvOver bits are cleared
; 2) Load TxReg with data to be transmitted
; 3) CALL PutChar function
;
; To Receive A Byte Of Data :
; 1) Make sure _txmtProgress & _rcvOver bits are cleared
; 2) CALL GetChar function
; 3) The received Byte is in TxReg after _rcvOver bit is cleared
;
; *****
;
; Processor 16C71
; Radix DEC
; EXPAND
; include "d:\pictools\16Cxx.h"
;

```



Software Implementation of Asynchronous Serial I/O

```
; Initialize A/D Converter
; <RA0:RA3> Configure as Analog Inputs, VDD as Vref
; A/D Clock Is Internal RC Clock
; Select Channel 0
;
; Program Memory : 6 locations
; Cycles : 7
;
;*****
InitAtod:
    bsf rp0
    clrf adcon1
    bcf rp0
    movlw 0xCl
    movwf adcon0
    return
;
;*****
; Main Program Loop
;
; After appropriate initialization, The main program wait for a command from RS-232
; The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
; After a command is received, the appropriate A/D Channel is selected and when conversion is
; completed the A/D Data is transmitted back to the Host. The controller now waits for a new
; command.
;*****
Start:
    call InitSerialPort
    call InitAtod
;
    WaitForNextSel:
        if _USE_RTSCS
            bcf rp0
            bcf RTS ; ready to accept data from host
        endif
        call GetChar ; wait for a byte reception
        btfsc rcvOver ; _rcvOver Gets Cleared when a Byte Is Received (in RxCReg)
        goto $-1 ; USER can perform other jobs here, can poll _rcvOver bit
;
; A Byte is received, Select The Desired Channel & TMMXT the desired A/D Channel Data
;
    bcf rp0 ; make sure to select Page 0
    movf RxCReg,w ; WREG = Commanded Channel # (0 thru 3)
    call GetADCon0 ; Get ADCON0 Reg Constant from Table Lookup
    movwf adcon0 ; Load ADCON0 reg, selecting the desired channel
    nop
;
    bsf go ; start conversion
```

Software Implementation of Asynchronous Serial I/O

```

btfscc done
goto $-1 ; Loop Until A/D Conversion Done

;
;
movf adres,w
movwf TxReg
if _USE_RTSCCTS
bsf RTS ; Half duplex mode, transmission mode, ask host not to send data
btfscc CTS ; Check CTS signal if host ready to accept data
goto $-1
endif
call PutChar
btfscc txmtProgress
goto $-1 ; Loop Until Transmission Over, User Can Perform Other Jobs

;
;
goto WaitForNextSel ; wait for next selection (command from Serial Port)
;
;
; *****
; ***** RS-232 Routines *****
; *****
; ***** Interrupt Service Routine *****
;
;
; Only RTCC Interrupt is used. RTCC Interrupt is used as timing for Serial Port Receive & Transmit
; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
; Transmit Modules.
;
; Transmission :
; RTCC is setup for Internal Clock increments and interrupt is generated when
; RTCC overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
; desired BAUD RATE.
;
; Reception :
; When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
; and preloaded with 0xFF. When a Falling Edge is detected on RTCC pin, RTCC is
; rolls over and an interrupt is generated (thus Start Bit Detect). Once the start
; bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
; with a certain value for regular timing interrupts to Poll RTCC pin (i.e RX pin).
;
; *****
; *****
Interrupt:
btfscc rtif ; other interrupt, simply return & enable GIE
retfie
;
; Save Status On INT : WREG & STATUS Regs
;
;
movwf SaveWReg
swapi status,w
movwf SaveStatus
;
; btfscc txmtProgress

```


Software Implementation of Asynchronous Serial I/O

Appendix C - RCVR.ASM

```
*****
;
;   GetChar Function
;   Receives a Byte Of Data
;   When reception is complete, _rcvOver Bit is cleared
;   The received data is in RxCReg
;
;   Program Memory : 15 locations (17 locations if PARITY is used)
;   Cycles         : 16 (18 if PARITY is USED)
;
*****
GetChar:
    bsf    _rcvOver                ; Enable Reception, this bit gets reset on Byte Rcv Complete
    LOAD_BITCOUNT
    cibr   RxCReg
    bcf    FrameErr
    bcf    ParityErr
    bsf    rp0
    movlw _OPTION_SBIT
    movwf _option
    bcf    rp0
    movlw _0xFF
    movwf _rtcc
    bcf    rtif
    bsf    rtie
    retfie
;
; *****
;   Internal Subroutine
;   entered from Interrupt Service Routine when Start Bit Is detected.
;
;   Program Memory : 14 locations
;   Cycles         : 12 (worst case)
;
*****
_SBitDetected:
    bcf    rp0
    btfsc RX_Pin
    goto  _FalseStartBit
    bsf    rcvProgress
    bsf    rp0
    movlw _OPTION_INIT | SBitPrescale) ; Switch Back to INT Clock
    movwf _option
    bcf    rp0
    LOAD_RTCC (SBitRtccLoad), SBitPrescale
    goto  _RestoreIntStatus
;
; *****
;   Make sure Start Bit Interrupt is not a Glitch
;   False Start Bit
```

Software Implementation of Asynchronous Serial I/O

```

_FalseStartBit:
    movlw    0xFF
    movwf   _rtcc
    goto    RestoreIntStatus
;
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 28 locations ( 43 locations with PARITY enabled)
; Cycles        : 24 Worst Case
;*****
_RcvNextBit:
    rp0
    bsf     (_OPTION_INIT | RtccPrescale) ; Switch Back to INT Clock
    movwf  _option                       ; Set Option Reg Located In Page 1
;
; rp0
; porta,w
; movwf   RxTemp
; LOAD_RTCC
; movf   RtccPreLoad, RtccPrescale      ; Macro to reload RTCC
; movf   porta,w
; xorwf  RxTemp,w
; andlw  RX_MASK
; btfsc  z
; goto   PinsSampled
; SampleAgain:
; movf   porta,w
; movwf  RxTemp
; movwf  _PinSampled
; if     _PARITY_ENABLE
; movf   BitCount
; z
; btfsc _RcvP_Or_S
; goto   _endif
;
; decfsz BitCount
; goto   NextRcvBit
;
; if _PARITY_ENABLE
; RcvP_Or_S:
; decfsz ExtraBitCount
; goto   RcvParity
; endif
;
; _RcvStopBit:
; btfss  RX
; bsf    FrameErr
;
; may be framing Error or Glitch

```


Appendix D - TXMTR.ASM

```
*****
;
; PutChar Function
;
; Function to transmit A Byte Of Data
; Before calling this routine, load the Byte to be transmitted into TxReg
; Make sure txmtProgress & _rcvOver bits (in Status Reg) are cleared before
; calling this routine
;
; Program Memory : 6 locations (10 locations if PARITY is Used)
; Cycles : 8 (13 if PARITY is Used)
;
;*****
PutChar:
    bsf txmtEnable ; enable transmission
    bsf txmtProgress
    LOAD_BITCOUNT ; Macro to load bit count
;
    if PARITY_ENABLE
        mov TxReg,W
        call GenParity ; If Parity is used, then Generate Parity Bit
    endif
;
    call TxmtStartBit
    bsf _rtie ; Enable RTCC Overflow INT
    retfie ; return with _GIE Bit Set
;
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 15 locations (25 locations if PARITY is used)
; Cycles : 13 Worst Case
;
;*****
_TxmtNextBit:
    bcf _rp0
    LOAD_RTCC RTCCPreLoad, RtccPrescale ; Macro to reload RTCC
;
    if PARITY_ENABLE
        movf BitCount
        btfsz _z
        goto ParityOrStop
    endif
;
    decfsz BitCount
```

Software Implementation of Asynchronous Serial I/O

```

    goto _NextTxmtBit
;
    if _PARITY_ENABLE
_ParityOrStop:
    decfsz ExtraBitCount
    goto _SendParity
    endif
;
_StopBit:
    bsf TX
    rtle
    bcf _txmtProgress
    goto _RestoreIntStatus
;
_NextTxmtBit:
    rrf TxReg
    btfss carry
    bcf TX
    btfsc carry
    bsf TX
;
    btfss txmtEnable
    bsf _rtle
;
    goto _RestoreIntStatus
;
    if _PARITY_ENABLE
_SendParity:
    btfss parityBit
    btfsc parityBit
    bsf TX
    goto _RestoreIntStatus
    endif
;
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 9 locations
; Cycles : 10
;
;*****
_TxmtStartBit:
    bsf rp0
    movlw (_OPTION_INIT | RtcclPrescale)
    movwf _option
; Set Option Reg Located In Page 1

```



```

; make sure to select Page 0
; Send Start Bit
; Prepare for Timing Interrupt

    bcf     _RP0
    bcf     TX
    movlw  -RtccPreLoad
    movwf  rtcc
    bcf     rtif
    return

;***** Generate Parity for the Value in WREG *****
;
;
; The parity bit is set in _parityBit (SerialStatus,7)
; Common Routine For Both Transmission & Reception
;
; Program Memory : 13 locations
; Cycles         : 14
;*****
; if _PARITY_ENABLE
GenParity:
    movwf  temp2
    swapf  temp2,w
    xorwf  temp2,w
    movwf  temp1
    rrf    temp1
    rrf    temp1
    xorwf  temp1
    incf   temp1
    rrf    temp1
; Parity bit is in Bit 0 of temp1
;
; if _ODD_PARITY
    bsf   _parityBit
    btfsc temp1,0
    bcf   _parityBit
    else
    bcf   _parityBit
    btfsc temp1,0
    bsf   _parityBit
    endif
    return
    endif
;*****

```

Software Implementation of Asynchronous Serial I/O

Appendix E - RS232.LST

```
MPASM B0.24
"RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
"Software Implementation : Interrupt Driven"

PAGE 1

TITLE "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
SUBTITLE "Software Implementation : Interrupt Driven"

;*****
; Software Implementation Of RS232 Communications Using PIC16CXX
; Half-Duplex
;
; These routines are intended to be used with PIC16C6X/7X family. These routines can be
; used with processors in the 16C6X/7X family which do not have on board Hardware Async
; Serial Port.
; MX..
;
; Description :
; Half Duplex RS-232 Mode Is implemented in Software.
; Both Reception & Transmission are Interrupt driven
; Only 1 peripheral (RTCC) used for both transmission & reception
; RTCC is used for both timing generation (for bit transmission & bit polling)
; and Start Bit Detection in reception mode.
; This is explained in more detail under Interrupt Subroutine.
; Programmable Baud Rate (speed dependng on Input Clock Freq.), programmable
; #of bits, Parity enable/disable, odd/even parity is implemented.
; Parity & Framing errors are detected on Reception
;
; RS-232 Parameters
;
; The RS-232 Parameters must be defined as shown below:
;
; ClkIn : Input Clock Frequency of the processor
; (NOTE : RC Clock Mode Is Not Suggested due to wide variations)
; BaudRate : Desired Baud Rate. Any valid value can be used.
; The highest Baud Rate achievable depends on Input Clock Freq.
; 600 to 4800 Baud was tested using 4 Mhz Input Clock
; 600 to 19200 Baud was tested using 10 Mhz Input Clock
; Higher rates can be obtained using higher Input Clock Frequencies.
; Once the BaudRate & ClkIn are specified the program
; automatically selectes all the appropriate timings
; Databits : Can specify 1 to 8 Bits.
; StopBits : Limited to 1 Stop Bit. Must set it to 1.
; PARITY_ENABLE : Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
; is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
; description below
;
;*****
```

Software Implementation of Asynchronous Serial I/O

```

;
; ODD_PARITY      : Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
;                 EVEN Parity Scheme is used.
;                 This Flag is ignored if _PARITY_ENABLE is set to FALSE.
;
;
; Usage :
;
; An example is given in the main program on how to Receive & Transmit Data
; In the example, the processor waits until a command is received. The command is interpreted
; as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
; is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
;
; The RS-232 Control/Status Reg's bits are explained below :
;
; "SerialStatus" : RS-232 Status/Control Register
;
; Bit 0 : txmtProgress (1 if transmission in progress, 0 if transmission is complete)
;         After a byte is transmitted by calling "PutChar" function, the
;         user's code can poll this bit to check if transmission is complete.
;         This bit is reset after the STOP bit has been transmitted
; Bit 1 : txmtEnable   Set this bit to 1 on initialization to enable transmission.
;         This bit can be used to Abort a transmission while the transmitter
;         is in progress (i.e when txmtProgress = 1)
; Bit 2 : rcvProgress Indicates if the receiver is in middle of reception. It is reset when
;         a byte is received.
; Bit 3 : rcvOver     This bit indicates the completion of Reception of a Byte. The user's
;         code can poll this bit after calling "GetChar" function. Once "GetChar"
;         function is called, this bit is 0 and is set to 1 after reception of
;         a complete Byte (parity bit if enabled & stop bit)
; Bit 4 : ParityErr   A 1 indicates Parity Error on Reception (for both even & odd parity)
; Bit 5 : FrameErr    A 1 indicates Framing Error On Reception
; Bit 6 : unused      Unimplemented Bit
; Bit 7 : parityBit   The 9 th bit of transmission or reception (status of PARITY bit
;                     if parity is enabled)
;
; To Transmit A Byte Of Data :
; 1) Make sure _txmtProgress & _rcvOver bits are cleared
; 2) Load TxReg with data to be transmitted
; 3) CALL PutChar function
;
; To Receive A Byte Of Data :
; 1) Make sure _txmtProgress & _rcvOver bits are cleared
; 2) CALL GetChar function
; 3) The received Byte is in TxReg after _rcvOver bit is cleared
;
; *****
Processor      16C71

```



Software Implementation of Asynchronous Serial I/O

```

include
Radix DEC
EXPAND
"d:\pictools\16Cxx.h"

;*****
;***** Setup RS-232 Parameters
;*****
000F 4240 equ 1000000 ; Input Clock Frequency is 4 Mhz
04B0 04B0 set 1200 ; Baud Rate (bits per second) is 1200
0008 0008 set 8 ; 8 bit data, can be 1 to 8
0001 0001 set 1 ; 1 Stop Bit, 2 Stop Bit is not implemented
0046 #define _PARITY_ENABLE FALSE ; NO Parity
0047 #define _ODD_PARITY FALSE ; EVEN Parity, if Parity enabled
0048 #define _USE_RTSCTS FALSE ; NO Hardware Handshaking is Used

include "rs232.h"

;*****
;*****
;***** Table Of ADCON0 Reg
;*****
0000 2811 org ResetVector
; goto Start
;
0004 2824 org IntVector
; goto Interrupt
;*****
;***** Table Of ADCON0 Reg
;*****
; Inputs : WREG (valid values are 0 thru 3)
; Returns In WREG, ADCON0 Value, selecting the desired Channel
;
; Program Memory : 6 locations
; Cycles : 5
;*****
GetADCon0:
andlw 0x03 ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)

```

Software Implementation of Asynchronous Serial I/O

```

0006 0782          addwf    pcl
0007 34C1          retlw   (0xCl | (0 << 3)) ; channel 0
0008 34C9          retlw   (0xCl | (1 << 3)) ; channel 1
0009 34D1          retlw   (0xCl | (2 << 3)) ; channel 2
000A 34D9          retlw   (0xCl | (3 << 3)) ; channel 3

GetADCon0_End:

if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
    MESSG    "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
endif
;
;*****
;      Initialize A/D Converter
; <RA0:RA3> Configure as Analog Inputs, VDD as Vref
; A/D Clock Is Internal RC Clock
; Select Channel 0
; Program Memory : 6 locations
; Cycles         : 7
;*****

InitAtod:
    bsf    rp0
    clrf  adcon1
    bcf   rp0
    movlw 0xCl
    movwf adcon0
    return
;*****
;
;      Main Program Loop
;*****
;
; After appropriate initialization, The main program wait for a command from RS-232
; The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
; After a command is received, the appropriate A/D Channel is selected and when conversion is
; completed the A/D Data is transmitted back to the Host. The controller now waits for a new
; command.
;*****

Start:
    call  InitSerialPort
    call  InitAtod
;
WaitForNextSel:
if _USE_RTSCTS
    bcf  rp0
    bcf  RTS
endif
; ready to accept data from host

```



Software Implementation of Asynchronous Serial I/O

```

0013 205D      ; wait for a byte reception
0014 198F      ; _rcvOver Gets Cleared when a Byte Is Received (in RxCReg)
0015 2814      ; USER can perform other jobs here, can poll _rcvOver bit

;
; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
;
0016 1283      bcf      GetChar
0017 080D      btfs    rcvOver
0018 2005      goto    $-1

;
; make sure to select Page 0
; WREG = Commanded Channel # (0 thru 3)
; Get ADCON0 Reg Constant from Table Lookup
; Load ADCON0 reg, selecting the desired channel
;
001A 0000      nop
001B 1508      bsf      rp0
001C 1908      movf   ExReg,w
001D 281C      call   GetADCon0
001E 0809      movwf  adcon0
001F 008C      nop

;
; start conversion
; Loop Until A/D Conversion Done
;
0020 203B      call   PutChar
0021 180F      btfs    txmtProgress
0022 2821      goto    $-1

;
; Loop Until Transmission Over, User Can Perform Other Jobs
; wait for next selection (command from Serial Port)
;
0023 2813      goto    WaitForNextSel

;
;*****
; RS-232 Routines
;*****
;
; Interrupt Service Routine
;
; Only RTCC Interrupt Is used. RTCC Interrupt is used as timing for Serial Port Receive & Transmit
; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
; Transmit Modules.
;
; Transmission :
; RTCC is setup for Internal Clock increments and interrupt is generated when
; RTCC overflows. Prescaler is assigned, depending on the INPUT CLOCK & the
; desired BAUD RATE.
;
; Reception :
; When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
; and preloaded with 0xFF. When a Falling Edge is detected on RTCC Pin, RTCC is
; rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
; bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
; with a certain value for regular timing interrupts to Poll RTCC Pin (i.e RX pin).
;
;*****

```

Software Implementation of Asynchronous Serial I/O

```

0024 1D0B      Interrupt:
0025 0009      btfs    rtif          ; other interrupt, simply return & enable GIE
                                ; Save Status On INT : WREG & STATUS Regs
                                ;
0026 0094      movwf   SaveWReg      ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0027 0E03      swapf  status,w      ;
0028 0095      movwf   SaveStatus ;
                                ;
0029 180F      btfs    txmtProgress ; Txmt Next Bit
002A 2842      goto   TxmtNextBit ;
002B 190F      btfs    rcvProgress ; Receive Next Bit
002C 287A      goto   SBitNextBit ; Must be start Bit
002D 286C      goto   SBitDetected ;
                                ;
002E 0E15      RestoreIntStatus:
002F 0083      swapf  SaveStatus,w ; restore STATUS Reg
0030 0E94      movwf  status ; save WREG
0031 0E14      swapf  SaveWReg,w ; restore WREG
0032 110B      bcf    rtif
0033 0009      retfie

; *****
;
;
; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
; Configure, RX_Pin (RTCC pin) as Input, which is used to poll data on reception
;
; Program Memory      :      8 locations
; Cycles              :      9
; *****
InitSerialPort:
    clrf    SerialStatus
;
0034 018F      bcf    rp0          ; select Page 0 for Port Access
0035 1283      bsf    TX           ; make sure TX pin is high on powerup, use RB Port Pullup
0036 1786      bsf    rp0          ; Select Page 1 for TrisB access
0037 1683      bcf    TX           ; set TX Pin As Output Pin, by modifying TRIS
0038 1386      if _USE_RTSCSTS
                                bcf    RTS
                                bsf    CTS
endif
0039 1605      bsf    RX_Pin      ; RTS is output signal, controlled by PIC16Cxx
003A 0008      return          ; CTS is input signal, controlled by the host
                                ; set RX Pin As Input for reception

```



Software Implementation of Asynchronous Serial I/O

0052 282E

```
        goto    RestoreIntStatus
;
;   if _PARITY_ENABLE
;   _SendParity:
;       btfss  parityBit
;       bcf    TX
;       btfsc  parityBit
;       bsf    TX
;       goto  RestoreIntStatus
endif

;
; *****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 9 Locations
; Cycles         : 10
; *****
; TxmtStartBit:
;   bsf    rp0
;   movlw  (OPTION_INIT | RtcPreScale) ; Set Option Reg Located In Page 1
;   movwf option                       ; make sure to select page 0
;   bcf    rp0
;   bcf    TX
;   movlw  -RtcPreLoad
;   movwf rtcc
;   bcf    rtif
;   return
; *****
; *****
; Generate Parity for the Value in WREG
;
; The parity bit is set in _parityBit (SerialStatus,7)
; Common Routine For Both Transmission & Reception
;
; Program Memory : 13 locations
; Cycles         : 14
; *****
; *****
; if _PARITY_ENABLE
GenParity:
    movwf temp2
    swapf temp2,w
    xorwf temp2,w
    movwf temp1
    rrf    temp1
```

0053 1683
0054 3008
0055 0081
0056 1283
0057 1386
0058 3030
0059 0081
005A 110B
005B 0008

Software Implementation of Asynchronous Serial I/O

```

    rrf    temp1
    xorwf  temp1
    incf   temp1
    rrf    temp1
    ; Parity bit is in Bit 0 of temp1
    ; Parity bit in Bit 0
;
; if _ODD_PARITY
    bsf   parityBit
    btfsc temp1,0
    bcf   parityBit
;
    else
    bcf   parityBit
    btfsc temp1,0
    bsf   parityBit
    endif
;
return
endif
;*****
005C 0008 ; The Receiver Routines are in File "rcvr.asm"
;*****
include "rcvr.asm"
;*****
; GetChar Function
;
; Receives a Byte Of Data
; When reception is complete, _rcvOver Bit is cleared
; The received data is in RxBReg
;
; Program Memory : 15 locations (17 locations if PARITY is used)
; Cycles : 16 (18 if PARITY is USED)
;*****
GetChar:
    bsf   rcvOver
    ; Enable Reception, this bit gets reset on Byte Rcv Complete
;*****
    movlw DataBits+_StopBits
    movwf BitCount
    if   PARITY_ENABLE
    movlw 2
    movwf ExtraBitCount
    endif
;*****
```

Software Implementation of Asynchronous Serial I/O

```

0060 018D      clr  RxReg
0061 128F      bcf  FrameErr
0062 120F      bcf  ParityErr
0063 1683      bsf  rp0
0064 3038      movlw OPTION_SBIT
0065 0081      movwf option
0066 1283      bcf  rp0
0067 30FF      movlw 0xFF
0068 0081      movwf rtcc
0069 110B      bcf  rtlf
006A 168B      bsf  rtie
006B 0009      retfie
;
; *****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory      : 14 locations
; Cycles              : 12 (worst case)
; *****
; SBitDetected:
006C 1283      bcf  rp0
006D 1A05      btfs  RX_Pin
006E 2877      goto FalseStartBit
006F 150F      bsf  rcvProgress
0070 1683      bsf  rp0
0071 3008      movlw (_OPTION_INIT | SBitPrescale)
0072 0081      movwf option
0073 1283      bcf  rp0
; LOAD_RTCC (SBitRtccLoad), SBitPrescale
; if(UsePrescale == 0)
0074 3090      movlw -(SBitRtccLoad) + _Cycle_Offset1
; else
0075 0081      movlw -(SBitRtccLoad) + (_Cycle_Offset1 >> (SBitPrescale+1))
; Re Load RTCC init value + INT La
endif
; Note that Prescaler is cleared when RTCC is written
0076 282E      goto RestoreIntStatus
;
; _FalseStartBit:
0077 30FF      movlw 0xFF
0078 0081      movwf rtcc
; reload RTCC with 0xFF for start bit detection

```

Software Implementation of Asynchronous Serial I/O

```

0079 282E      goto RestoreIntStatus
;*****
; Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
; Program Memory : 28 locations ( 43 locations with PARITY enabled)
; Cycles : 24 Worst Case
;*****
_RcvNextBit:
007A 1683      bsf rp0
007B 3008      movlw (OPTION_INIT | RtccPrescale) ; Switch Back to INT Clock
007C 0081      movwf option ; Set Option Reg Located In Page 1
;
007D 1283      bcf rp0
007E 0805      movf porta,w ; read RX pin immediately into WREG
007F 008E      movwf RxTemp
;
LOAD_RTCC RtccPreLoad, RtccPrescale ; Macro to reload RTCC

if (UsePrescale == 0)
0080 3042      movlw -RtccPreLoad + _Cycle_Offset1
else
movlw -RtccPreLoad + (Cycle_Offset1 >> (RtccPrescale+1)) ; Re Load RTCC init value + INT La
endif

0081 0081      movwf rtcc
0082 0805      movf porta,w
0083 060E      xorwf RxTemp,w
0084 3910      andlw RX_MASK
0085 1903      btfsz z
0086 2889      goto PinSampled
SampleAgain:
0087 0805      movf porta,w
0088 008E      movwf RxTemp

PinSampled:
if _PARITY_ENABLE
movf BitCount
btfsz z
goto RcvP_Or_S
endif
;
0089 0E90      decfsz BitCount
008A 2891      goto NextRcvBit
;
if _PARITY_ENABLE
RcvP_Or_S:

```


Four Channel Digital Voltmeter with Display and Keyboard

INTRODUCTION

The PIC16C71 is a member of a new family of 8-bit microcontrollers, namely the PIC16CXX. The salient features of the PIC16C71 are:

- Improved and enhanced instruction set
- 14-bit instruction word
- Interrupt capability
- On-chip four channel, 8-bit A/D converter

This application note demonstrates the capability of the PIC16C71. To make this application note easier for the end user, it has been broken down into four sub-sections:

- Section 1: Implements the multiplexing of four 7 segment LEDs with the PIC16C71.
- Section 2: Implements the multiplexing of four 7 segment LEDs as well as the sampling of a 4x4 Keypad.
- Section 3: Implements the multiplexing of four 7 segment LEDs as well as the sampling of one A/D channel.
- Section 4: Implements the multiplexing of four 7 segment LEDs, sampling a 4x4 keypad and four A/D channels.

IMPLEMENTATION

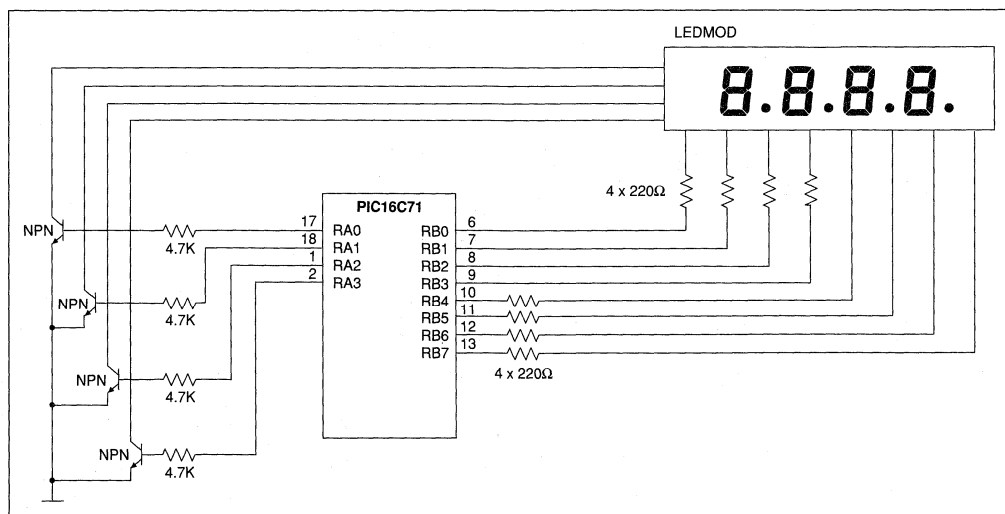
SECTION 1: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS

Hardware

The PIC16C71's I/O ports have an improved sink/source specification. Each I/O pin can sink up to 25 mA and source 20 mA, in addition total Port B source current is 100 mA and sink current is 150 mA. Port A is rated for 50 mA source current and 80 mA sink current. This makes the PIC16C71 ideal for driving 7 segment LEDs. Since the total number of I/O is limited to 13, the 8-bit Port B is used to drive the 8 LEDs, while external sink transistors or MOSFETs are used to sink the digit current (See Figure 1). Another alternative is to use ULN2003 open collector sink current drivers, which are available in 16 pin DIP or very small S0-16 packages. Each transistor on the ULN2003 can sink a maximum of 500 mA and the base drive can be directly driven from the Port A pins.

3

FIGURE 1 - MULTIPLEXING FOUR 7 SEGMENT LEDs



Four Channel Digital Voltmeter with Display and Keyboard

Software

The multiplexing is achieved by turning on each LED for a 5 msec duration every 20 msec. This gives an update rate of 50 Hz, which is quite acceptable to the human eye as a steady display. The 5 msec time base is generated by dividing the 4.096 MHz oscillator clock. The internal prescaler is configured to be a divide by 32 and assigned to the RTCC. The RTCC is pre-loaded with a value = 96. The RTCC will increment to FF and then roll over to 00 after a period = $(256-96) \cdot (32 \cdot 4 / 4096000) = 5$ msec. When the RTCC rolls over, the RTIF flag is set and since the RTIE and GIE bits are enabled, an interrupt is generated.

The software implements a simple timer which increments at a 1 second rate. Every second, the 4 nibble (two 8-bit registers MsdTime and LsdTime) are incremented in a BCD format. The lower 4 bits of LsdTime correspond to the least significant digit (LSD) on the display. The high 4 bits of LsdTime correspond to the second significant digit of the display and so on. Depending on which display is turned on, the corresponding 4 bit BCD value is extracted from either MsdTime or LsdTime, and decoded to a 7 segment display. The RTCC interrupt is generated at a steady rate of 5 msec and given an instruction time of 1 us. The entire display update program can reside in the interrupt service routine with no chance of getting an interrupt within an interrupt. The Code listing for Section 1 is in Appendix A.

SECTION 2: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND SCANNING A 4X4 KEYPAD

Hardware

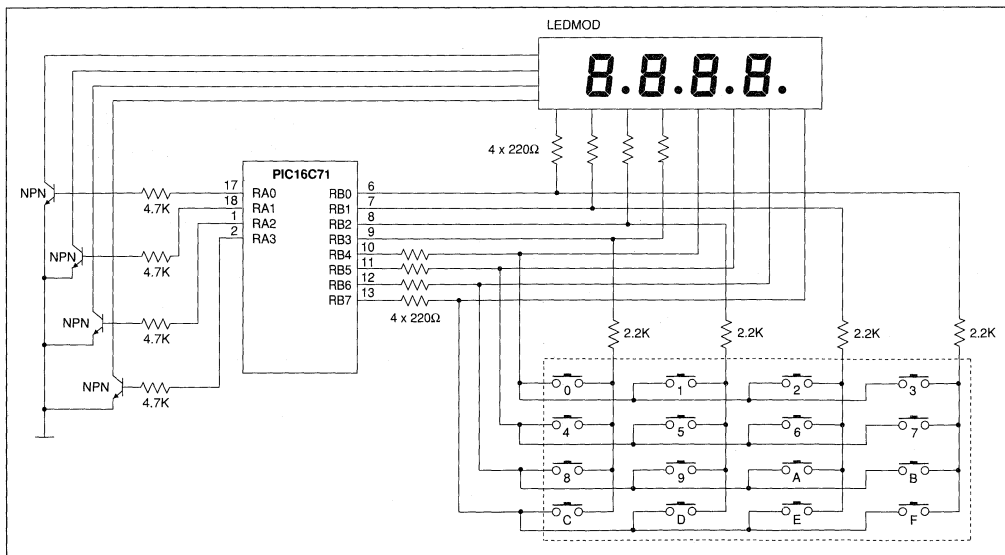
A 4x4 keypad can be very easily interfaced to the PIC16C71's Port B (see Figure 2). Internal pull-ups on pins RB4 to RB7 can be enabled and disabled by setting the RBPU bit in the OPTION register. The internal pull-ups have a value of 20K at 5V (typical). In order to sense a low level at the input, the switch is "connected" to ground through a 2.2K resistor. A key hit normally lasts from 50 msec to as long as a person holds the key down. In order not to miss any key hits, the keypad is sampled every 20 msec (just after the update of the MSD).

Software

To sample the keypad, the digit sinks are first disabled. Port B is then configured with RB4-RB7 as inputs and RB0-RB3 as outputs driven high. The pull-ups on RB4-RB7 are enabled. Sequentially RB0 to RB3 are made low while RB4 to RB7 are checked for a key hit (a low level). One key hit per scan is demonstrated in this program. Multiple key hits per scan can very easily be implemented. Once the key hit is sensed, a 40 msec debounce period elapses before key sampling is resumed. No more key hits are sensed until the present key is released. This prevents erroneous key inputs.

The program basically inputs the key hit and displays its value as a hexadecimal character on the multiplexed 7-segment LEDs. The Code Listing for Section 2 is in Appendix B.

FIGURE 2 - MULTIPLEXING FOUR 7 SEGMENT LEDs WITH A 4X4 KEYPAD



Four Channel Digital Voltmeter with Display and Keyboard

SECTION 3: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND THE A/D CHANNEL 0

Hardware

The four analog channels are connected to RA0-RA3. If any of these pins are used normally as digital I/O, they can momentarily be used as analog inputs. In order to avoid interference from the analog source, it is advisable to buffer the analog input through a voltage follower or op amp, however, it is not always necessary. Figure 3A and 3B show some typical configurations. In this application, the analog input is a potentiometer whose wiper is connected through an RC network to channel 0. The RC is necessary in order to smooth out the analog voltage. The RC does contribute to a delay in the sampling time, however the stability of the analog reading is greatly improved.

Software

The analog input is sampled every 20 msec. The digit sinks and the drivers are turned off i.e. Port A is configured as an input and Port B outputs are made low. A 1msec settling time is allowed for the external RC network connected to the analog input to settle and then the A/D conversion is started. The result is read then converted from an 8-bit binary value to a 3 digit BCD value which is then displayed on the 7 Segment LEDs. The Code Listing for Section 3 is in Appendix C.

FIGURE 3A - TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT

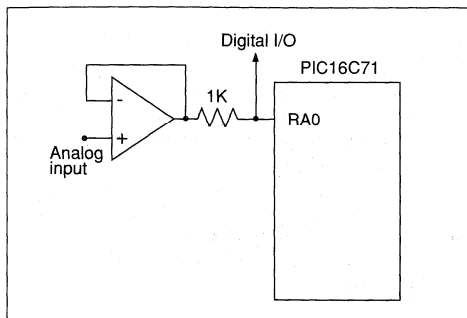
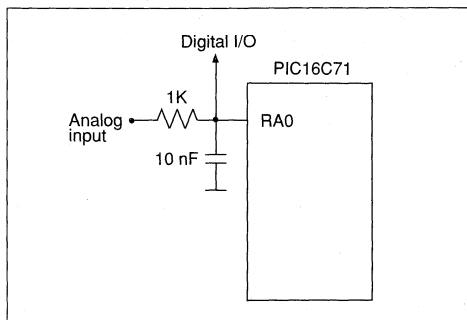


FIGURE 3B - TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT



Four Channel Digital Voltmeter with Display and Keyboard

SECTION 4: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS WITH A 4X4 KEYPAD AND 4 A/D CHANNELS

Hardware

This section essentially incorporated Sections 1, 2 and 3 to give a complete four channel voltmeter. Figure 4 shows a typical configuration. The analog channels are connected through individual potentiometers to their respective analog inputs and are sampled every 20 msec in a round robin format. The sampling rate can be increased to as fast as once every 5 msec if required. The keypad sampling need not be any faster than once every 20 msec.

Software

The program samples the analog inputs and saves the result in four consecutive locations starting at "ADVALUE", with channel 0 saved at the first location and so on. By default, channel 0 is displayed. If Key 1 is pressed, channel 1 is displayed and so on. Key hits > 3 are ignored. The Code Listing for Section 4 is in Appendix D.

Code Size

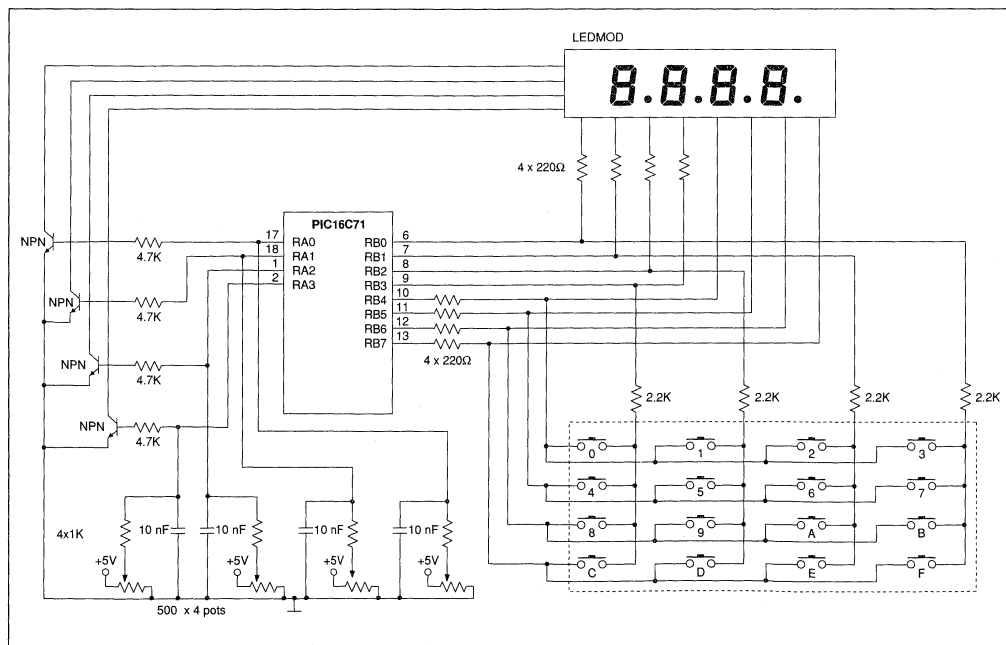
Section 1:	Program Memory:	139
	Data Memory:	6
Section 2:	Program Memory:	207
	Data Memory:	13
Section 3:	Program Memory:	207
	Data Memory:	17
Section 4:	Program Memory:	
	Data Memory:	

Conclusion

The 4 A/D channels on the PIC16C71 can be multiplexed with digital I/O, thus reducing overall pin counts and improving I/O pin usage in an analog application.

AUTHOR: Stan D'Souza, Logic Products Division

FIGURE 4 - FOUR CHANNEL VOLTMETER WITH DISPLAY AND KEYPAD



Four Channel Digital Voltmeter with Display and Keyboard

APPENDIX A

MPASM B0.50

PAGE 1

```

;*****
;This program is to demonstrate how to multiplex four 7 segment LED
;digits using a PIC16C71. The four digits will start at 0000 and
;increment at a 1 sec rate up to 9999.
;The LEDs are updated every 5 msec, for a multiplexing rate of 20 msec.
;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
;
;
;                               Stan D'Souza 5/8/93
;*****
LIST P=16C71, F=INHX8M
;
include "picreg.equ"
;
;
TempC      equ    0x0c      ;temp general purpose regs
TempD      equ    0x0d
TempE      equ    0x0e
Count      equ    0x0f      ;count
MsdTime    equ    0x10      ;most significant Timer
LsdTime    equ    0x11      ;Least significant Timer
OptionReg  equ    1
PCL        equ    2
BcdMsd     equ    26
Bcd        equ    27
;
;
org        0
0000 2805      ;goto Start      ;skip over interrupt vector
;
;
org        4
0004 281D      ;goto ServiceInterrupts
;
Start
0005 2008      call    InitPorts
0006 2012      call    InitTimers
;
loop
0007 2807      goto    loop
;
InitPorts
0008 1683      bsf    STATUS,RP0      ;select pg 1
0009 3003      movlw  3          ;make RA0-3 digital I/O
000A 0108      movwf  ADCON1      ;
000B 0205      clrf   TRISA      ;make RA0-4 outputs
000C 0206      clrf   TRISB      ;make RB0-7 outputs
000D 1283      bcf    STATUS,RP0      ;select page 0
000E 0185      clrf   PORT_A      ;make all outputs low
000F 0186      clrf   PORT_B      ;
0010 1585      bsf    PORT_A,3      ;enable MSB digit sink
0011 0008      return
;
;
;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
;end result is that we get a rtcc interrupt every 5msec.
InitTimers
0012 0190      clrf   MsdTime      ;clr timers
0013 0191      clrf   LsdTime      ;
0014 1683      bsf    STATUS,RP0      ;select pg 1
0015 3084      movlw  B'10000100'    ;assign ps to rtcc

```

Four Channel Digital Voltmeter with Display and Keyboard

```
0016 0081      movwf  OptionReg      ;ps = 32
0017 1283      bcf    STATUS,RP0     ;select pg 0
0018 3020      movlw  B'00100000'   ;enable rtcc interrupt
0019 008B      movwf  INTCON        ;
001A 3060      movlw  .96          ;preload rtcc
001B 0081      movwf  RTCC          ;start counter
001C 0009      retfie

;
ServiceInterrupts
001D 190B      btfsz  INTCON,RTIF    ;rtcc interrupt?
001E 2822      goto   ServiceRTCC   ;yes then service
001F 3020      movlw  B'00100000'   ;else clr rest
0020 008B      movwf  INTCON        ;
0021 0009      retfie

;
ServiceRTCC
0022 3060      movlw  .96          ;initialize rtcc
0023 0081      movwf  RTCC          ;
0024 110B      bcf    INTCON,RTIF    ;clr int flag
0025 2028      call   IncTimer      ;inc timer
0026 2050      call   UpdateDisplay  ;update display
0027 0009      retfie

;
;The display is incremented every 200*5msec = 1 Sec.
IncTimer
0028 0A0F      incf   Count,w        ;inc count
0029 3AC8      xorlw  .200          ;= 200?
002A 1903      btfsz  STATUS,Z      ;no then skip
002B 282E      goto   DoIncTime     ;else inc time
002C 0A8F      incf   Count        ;
002D 0008      return

DoIncTime
002E 018F      clrf   Count        ;clr count
002F 0A11      incf   LsdTime,w    ;get lsd
0030 390F      andlw  0x0F         ;mask high nibble
0031 3A0A      xorlw  0x0a        ; = 10?
0032 1903      btfsz  STATUS,Z      ;no then skip
0033 2836      goto   IncSecondLsd ;inc next lsd
0034 0A91      incf   LsdTime     ;else inc timer
0035 0008      return

IncSecondLsd
0036 0E11      swapf  LsdTime,w   ;get hi in low nibble
0037 390F      andlw  0x0F         ;mask hi nibble
0038 3E01      addlw  1           ;inc it
0039 0091      movwf  LsdTime     ;restore back
003A 0E91      swapf  LsdTime     ; /
003B 3A0A      xorlw  0x0a        ; = 10?
003C 1903      btfsz  STATUS,Z      ;no then skip
003D 283F      goto   IncThirdLsd ;else inc next lsd
003E 0008      return

IncThirdLsd
003F 0191      clrf   LsdTime     ;
0040 0A10      incf   MsdTime,w   ;get 3rd lsd
0041 390F      andlw  0x0F         ;mask hi nibble
0042 3A0A      xorlw  0x0a        ; = 10?
0043 1903      btfsz  STATUS,Z      ;no then skip
0044 2847      goto   IncMsd      ;else Msd
0045 0A90      incf   MsdTime     ;else inc timer
0046 0008      return

IncMsd
0047 0E10      swapf  MsdTime,w   ;get hi in lo nibble
0048 390F      andlw  0x0F         ;mask hi nibble
0049 3E01      addlw  1           ;inc timer
004A 0090      movwf  MsdTime     ;restore back
004B 0E90      swapf  MsdTime     ; /
004C 3A0A      xorlw  0x0a        ; = 10?
004D 1903      btfsz  STATUS,Z      ;no then skip
004E 0190      clrf   MsdTime     ;clr msd
004F 0008      return
```

Four Channel Digital Voltmeter with Display and Keyboard

```

;
;
UpdateDisplay
0050 0805      movf   PORT_A,w           ;present sink value in w
0051 0185      clrwf  PORT_A           ;disable all digits sinks
0052 390F      andlw  0x0f
0053 008C      movwf  TempC           ;save sink value in tempC
0054 160C      bsf    TempC,4         ;preset for lsd sink
0055 0C8C      rrf    TempC           ;determine next sink value
0056 1C03      btfsz  STATUS,CARRY   ;c=1?
0057 118C      bcf    TempC,3         ;no then reset LSD sink
0058 180C      btfsz  TempC,0         ;else see if Msd
0059 286B      goto   UpdateMsd       ;yes then do Msd
005A 188C      btfsz  TempC,1         ;see if 3rdLsd
005B 2866      goto   Update3rdLsd    ;yes then do 3rd Lsd
005C 190C      btfsz  TempC,2         ;see if 2nd Lsd
005D 2861      goto   Update2ndLsd    ;yes then do 2nd lsd

UpdateLsd
005E 0811      movf   LsdTime,w           ;get Lsd in w
005F 390F      andlw  0x0f           ; /
0060 286F      goto   DisplayOut        ;enable display

Update2ndLsd
0061 2080      call   Chk2LsdZero           ;msd = 0 & 2 lsd 0?
0062 1D03      btfsz  STATUS,Z           ;yes then skip
0063 0E11      swapf  LsdTime,w           ;get 2nd Lsd in w
0064 390F      andlw  0x0f           ;mask rest
0065 286F      goto   DisplayOut        ;enable display

Update3rdLsd
0066 2088      call   ChkMsdZero           ;msd = 0?
0067 1D03      btfsz  STATUS,Z           ;yes then skip
0068 0810      movf   MsdTime,w           ;get 3rd Lsd in w
0069 390F      andlw  0x0f           ;mask low nibble
006A 286F      goto   DisplayOut        ;enable display

UpdateMsd
006B 0E10      swapf  MsdTime,w           ;get Msd in w
006C 390F      andlw  0x0f           ;mask rest
006D 1903      btfsz  STATUS,Z           ;msd != 0 then skip
006E 300A      movlw  0x0a

DisplayOut
006F 2074      call   LedTable           ;get digit output
0070 0086      movwf  PORT_B           ;drive leds
0071 080C      movf   TempC,w           ;get sink value in w
0072 0085      movwf  PORT_A
0073 0008      return

;
;
LedTable
0074 0782      addwf  PCL           ;add to PC low
0075 343F      retlw  B'00111111'    ;led drive for 0
0076 3406      retlw  B'00000110'    ;led drive for 1
0077 345B      retlw  B'01011011'    ;led drive for 2
0078 344F      retlw  B'01001111'    ;led drive for 3
0079 3466      retlw  B'01100110'    ;led drive for 4
007A 346D      retlw  B'01101101'    ;led drive for 5
007B 347D      retlw  B'01111101'    ;led drive for 6
007C 3407      retlw  B'00000111'    ;led drive for 7
007D 347F      retlw  B'01111111'    ;led drive for 8
007E 3467      retlw  B'01100111'    ;led drive for 9
007F 3400      retlw  B'00000000'    ;blank led drive

;
;
Chk2LsdZero
0080 2088      call   ChkMsdZero           ;msd = 0?
0081 1D03      btfsz  STATUS,Z           ;yes then skip
0082 0008      return                   ;else return
0083 0E11      swapf  LsdTime,w           ;get 2nd lsd
0084 390F      andlw  0x0f           ;mask of LSD
0085 1D03      btfsz  STATUS,Z           ;0? then skip
0086 0008      return

```


Four Channel Digital Voltmeter with Display and Keyboard

Appendix B

MPASM B0.50

PAGE 1

```
*****
;This program is to demonstrate how to multiplex four 7 segment LED
;digits and a 4X4 keypad using a PIC16C71.
;The four digits will start as '0000' and when a key is hit
;it is displayed on the 7 segment leds as a hex value 0 to F. The last
;digit hit is always displayed on the right most led with the rest of
;the digits shifted to the left. The left most digit is deleted.
;The LEDs are updated every 20msec, the keypad is scanned at a rate of 20
msec.

;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
;
;
;                               Stan D'Souza 5/8/93
;*****
LIST P=16C71, F=INHX8M
;
include "picreg.equ"

;
000C TempC equ 0x0c ;temp general purpose regs
000D TempD equ 0x0d
000E TempE equ 0x0e
0020 PABuf equ 0x20
0021 PBBuf equ 0x21
000F Count equ 0x0f ;count
0010 MsdTime equ 0x10 ;most significant Timer
0011 LsdTime equ 0x11 ;Least significant Timer
0012 KeyFlag equ 0x12 ;flags related to key pad
0000 keyhit equ 0 ;bit 0 -> key-press on
0001 DebnceOn equ 1 ;bit 1 -> debounce on
0002 noentry equ 2 ;no key entry = 0
0003 ServKey equ 3 ;bit 3 -> service key
0013 Debnce equ 0x13 ;debounce counter
0014 NewKey equ 0x14
002F WBuffer equ 0x2f
002E StatBuffer equ 0x2e
0001 OptionReg equ 1
0002 PCL equ 2
;
;
push macro
movwf WBuffer ;save w reg in Buffer
swapf WBuffer ;swap it
swapf STATUS,w ;get status
movwf StatBuffer ;save it
endm
;
pop macro
swapf StatBuffer,w ;restore status
movwf STATUS ; /
swapf WBuffer,w ;restore W reg
endm
;
org 0
0000 280D goto Start ;skip over interrupt vector
;
org 4
;It is always a good practice to save and restore the w reg,
;and the status reg during a interrupt.
push
0004 00AF movwf WBuffer ;save w reg in Buffer
0005 0EAF swapf WBuffer ;swap it
0006 0E03 swapf STATUS,w ;get status
0007 00AE movwf StatBuffer ;save it
0008 2036 call ServiceInterrupts
```

Four Channel Digital Voltmeter with Display and Keyboard

```

                                pop
0009 0E2E                swapf  StatBuffer,w          ;restore status
000A 0083                movwf  STATUS                ; /
000B 0E2F                swapf  WBuffer,w            ;restore W reg

000C 0009                retfie

;
Start
000D 2020                call  InitPorts
000E 202A                call  InitTimers

loop
000F 1992                btfsc KeyFlag,ServKey      ;key service pending
0010 2012                call  ServiceKey           ;yes then service
0011 280F                goto  loop

;
;ServiceKey, does the software service for a keyhit. After a key service,
;the ServKey flag is reset, to denote a completed operation.
ServiceKey
0012 0814                movf  NewKey,w              ;get key value
0013 008E                movwf TempE                ;save in TempE
0014 0E10                swapf MsdTime,w           ;move MSD out
0015 39F0                andlw B'11110000'        ;clr lo nibble
0016 0090                movwf MsdTime             ;save back
0017 0E11                swapf LsdTime,w          ;get Lsd
0018 390F                andlw B'00001111'        ;mask off lsd
0019 0490                iorwf MsdTime             ;and left shift 3rd
001A 0E11                swapf LsdTime,w          ;get Lsd again
001B 39F0                andlw B'11110000'        ;mask off 2nd
001C 040E                iorwf TempE,w            ;or with new lsd
001D 0091                movwf LsdTime            ;make Lsd
001E 1192                bcf  KeyFlag,ServKey     ;reset service flag
001F 0008                return

;
InitPorts
0020 1683                bsf  STATUS,RP0          ;select pg 1
0021 3003                movlw 3                    ;make RA0-3 digital I/O
0022 0108                movwf ADCON1              ; /
0023 0205                clrf TRISA                ;make RA0-4 outputs
0024 0206                clrf TRISB                ;make RB0-7 outputs
0025 1283                bcf  STATUS,RP0          ;select page 0
0026 0185                clrf PORT_A               ;make all outputs low
0027 0186                clrf PORT_B               ; /
0028 1585                bsf  PORT_A,3             ;enable MSB digit sink
0029 0008                return

;
;
;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
;end result is that we get a rtcc interrupt every 5msec.
InitTimers
002A 0190                clrf  MsdTime             ;clr timers
002B 0191                clrf  LsdTime             ; /
002C 0192                clrf  KeyFlag            ;clr all flags
002D 1683                bsf  STATUS,RP0          ;select pg 1
002E 3084                movlw B'10000100'        ;assign ps to rtcc
002F 0081                movwf OptionReg          ;ps = 32
0030 1283                bcf  STATUS,RP0          ;select pg 0
0031 3020                movlw B'00100000'        ;enable rtcc interrupt
0032 008B                movwf INTCON              ;
0033 3060                movlw .96                 ;preload rtcc
0034 0081                movwf RTCC                ;start counter
0035 0009                retfie

;
ServiceInterrupts
0036 190B                btfsc INTCON,RTIF        ;rtcc interrupt?
0037 283B                goto  ServiceRTCC        ;yes then service
0038 018B                clrf  INTCON              ;else clr all int
0039 168B                bsf  INTCON,RTIE         ;

```


Four Channel Digital Voltmeter with Display and Keyboard

```

003A 0008          return
                    ;
                    ServiceRTCC
003B 3060          movlw   .96           ;initialize rtcc
003C 0081          movwf   RTCC
003D 110B          bcf     INTCON,RTIF       ;clr int flag
003E 1805          btfsc  PORT_A,0       ;if msb on then do
003F 2042          call    ScanKeys        ;do a quick key scan
0040 20A1          call    UpdateDisplay     ;update display
0041 0008          return

;
;
;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
;Debounce for a given keyhit is also taken care of.
;The rate of key scan is 20msec with a 4.096Mhz clock.
ScanKeys
0042 1C92          btfss  KeyFlag,DebnceOn     ;debounce on?
0043 2848          goto    Scan1           ;no then scan keypad
0044 0B93          decfsz  Debnce           ;else dec debounce time
0045 0008          return          ;not over then return
0046 1092          bcf     KeyFlag,DebnceOn     ;over, clr debounce flag
0047 0008          return          ;and return

Scan1
0048 208A          call    SavePorts        ;save port values
0049 30EF          movlw   B'11101111'       ;init TempD
004A 008D          movwf   TempD

ScanNext
004B 0806          movf    PORT_B,w           ;read to init port
004C 100B          bcf     INTCON,RBIF       ;clr flag
004D 0C8D          rrf     TempD           ;get correct column
004E 1C03          btfss  STATUS,C           ;if carry set?
004F 2862          goto    NoKey           ;no then end
0050 080D          movf    TempD,w           ;else output
0051 0086          movwf   PORT_B           ;low column scan line
0052 0000          nop
0053 1C0B          btfss  INTCON,RBIF       ;flag set?
0054 284B          goto    ScanNext        ;no then next
0055 1812          btfsc  KeyFlag,keyhit     ;last key released?
0056 2860          goto    SKreturn        ;no then exit
0057 1412          bsf     KeyFlag,keyhit     ;set new key hit
0058 0E06          swapf  PORT_B,w           ;read port
0059 008E          movwf   TempE           ;save in TempE
005A 2064          call    GetKeyValue        ;get key value 0 - F
005B 0094          movwf   NewKey           ;save as New key
005C 1592          bsf     KeyFlag,ServKey     ;set service flag
005D 1492          bsf     KeyFlag,DebnceOn     ;set flag
005E 3004          movlw   4
005F 0093          movwf   Debnce           ;load debounce time

SKreturn
0060 2097          call    RestorePorts       ;restore ports
0061 0008          return

;
NoKey
0062 1012          bcf     KeyFlag,keyhit     ;clr flag
0063 2860          goto    SKreturn

;
;GetKeyValue gets the key as per the following layout
;
;
;
;
;Row1 (RB4)      0      1      2      3
;
;Row2 (RB5)      4      5      6      7
;
;Row3 (RB6)      8      9      A      B
;
;Row4 (RB7)      C      D      E      F

```

Four Channel Digital Voltmeter with Display and Keyboard

```

;
GetKeyValue
0064 018C      clrfs TempC
0065 1D8D      btffs TempD,3      ;first column
0066 286E      goto RowValEnd
0067 0A8C      incf TempC
0068 1D0D      btffs TempD,2      ;second col.
0069 286E      goto RowValEnd
006A 0A8C      incf TempC
006B 1C8D      btffs TempD,1      ;3rd col.
006C 286E      goto RowValEnd
006D 0A8C      incf TempC      ;last col.
RowValEnd
006E 1C0E      btffs TempE,0      ;top row?
006F 2878      goto GetValCom     ;yes then get 0,1,2&3
0070 1C8E      btffs TempE,1      ;2nd row?
0071 2877      goto Get4567       ;yes the get 4,5,6&7
0072 1D0E      btffs TempE,2      ;3rd row?
0073 2875      goto Get89ab       ;yes then get 8,9,a&b
Getcdef
0074 150C      bsf TempC,2      ;set msb bits
Get89ab
0075 158C      bsf TempC,3      ; /
0076 2878      goto GetValCom     ;do common part
Get4567
0077 150C      bsf TempC,2
GetValCom
0078 080C      movf TempC,w
0079 0782      addwf PCL
007A 3400      retlw 0
007B 3401      retlw 1
007C 3402      retlw 2
007D 3403      retlw 3
007E 3404      retlw 4
007F 3405      retlw 5
0080 3406      retlw 6
0081 3407      retlw 7
0082 3408      retlw 8
0083 3409      retlw 9
0084 340A      retlw 0a
0085 340B      retlw 0b
0086 340C      retlw 0c
0087 340D      retlw 0d
0088 340E      retlw 0e
0089 340F      retlw 0f
;
;SavePorts, saves the porta and portb condition during a key scan
;operation.
SavePorts
008A 0805      movf PORT_A,w      ;Get sink value
008B 00A0      movwf PABuf       ;save in buffer
008C 0185      clrfs PORT_A      ;disable all sinks
008D 0806      movf PORT_B,w      ;get port b
008E 00A1      movwf PBBuf       ;save in buffer
008F 30FF      movlw 0xff        ;make all high
0090 0086      movwf PORT_B      ;on port b
0091 1683      bsf STATUS,RP0    ;select page 1
0092 1381      bcf OptionReg,7   ;enable pull ups
0093 30F0      movlw B'11110000' ;port b hi nibble inputs
0094 0106      movwf TRISB       ;lo nibble outputs
0095 1283      bcf STATUS,RP0    ;page 0
0096 0008      return
;
;RestorePorts, restores the condition of porta and portb after a
;key scan operation.
RestorePorts
0097 0821      movf PBBuf,w      ;get port n
0098 0086      movwf PORT_B
0099 0820      movf PABuf,w      ;get port a value

```

Four Channel Digital Voltmeter with Display and Keyboard

```

009A 0085      movwf PORT_A
009B 1683      bsf STATUS,RPO      ;select page 1
009C 1781      bsf OptionReg,7    ;disable pull ups
009D 0205      clrf TRISA         ;make port a outputs
009E 0206      clrf TRISE         ;as well as PORTB
009F 1283      bcf STATUS,RPO      ;page 0
00A0 0008      return

;
;
UpdateDisplay
00A1 0805      movf PORT_A,w      ;present sink value in w
00A2 0185      clrf PORT_A      ;disable all digits sinks
00A3 390F      andlw 0x0f
00A4 008C      movwf TempC       ;save sink value in tempC
00A5 160C      bsf TempC,4      ;preset for lsd sink
00A6 0C8C      rrf TempC        ;determine next sink value
00A7 1C03      btfss STATUS,CARRY ;c=1?
00A8 118C      bcf TempC,3      ;no then reset LSD sink
00A9 180C      btfscl TempC,0    ;else see if Msd
00AA 28B8      goto UpdateMsd    ;yes then do Msd
00AB 188C      btfscl TempC,1    ;see if 3rdLsd
00AC 28B5      goto Update3rdLsd ;yes then do 3rd Lsd
00AD 190C      btfscl TempC,2    ;see if 2nd Lsd
00AE 28B2      goto Update2ndLsd ;yes then do 2nd lsd

UpdateLsd
00AF 0811      movf LsdTime,w     ;get Lsd in w
00B0 390F      andlw 0x0f      ; /
00B1 28BA      goto DisplayOut

Update2ndLsd
00B2 0E11      swapf LsdTime,w   ;get 2nd Lsd in w
00B3 390F      andlw 0x0f      ;mask rest
00B4 28BA      goto DisplayOut   ;enable display

Update3rdLsd
00B5 0810      movf MsdTime,w    ;get 3rd Lsd in w
00B6 390F      andlw 0x0f      ;mask low nibble
00B7 28BA      goto DisplayOut   ;enable display

UpdateMsd
00B8 0E10      swapf MsdTime,w   ;get Msd in w
00B9 390F      andlw 0x0f      ;mask rest

DisplayOut
00BA 20BF      call LedTable     ;get digit output
00BB 0086      movwf PORT_B     ;drive leds
00BC 080C      movf TempC,w     ;get sink value in w
00BD 0085      movwf PORT_A
00BE 0008      return

;
;
LedTable
00BF 0782      addwf PCL         ;add to PC low
00C0 343F      retlw B'00111111' ;led drive for 0
00C1 3406      retlw B'00000110' ;led drive for 1
00C2 345B      retlw B'01011011' ;led drive for 2
00C3 344F      retlw B'01001111' ;led drive for 3
00C4 3466      retlw B'01100110' ;led drive for 4
00C5 346D      retlw B'01101101' ;led drive for 5
00C6 347D      retlw B'01111101' ;led drive for 6
00C7 3407      retlw B'00000111' ;led drive for 7
00C8 347F      retlw B'01111111' ;led drive for 8
00C9 3467      retlw B'01100111' ;led drive for 9
00CA 3477      retlw B'01110111' ;led drive for A
00CB 347C      retlw B'01111100' ;led drive for b
00CC 3439      retlw B'00111001' ;led drive for C
00CD 345E      retlw B'01011110' ;led drive for d
00CE 3479      retlw B'01111001' ;led drive for E
00CF 3471      retlw B'01110001' ;led drive for F

;
;
end
;

```

Four Channel Digital Voltmeter with Display and Keyboard

Appendix C

MPASM B0.50

PAGE 1

```
*****
;This program is to demonstrate how to multiplex four 7 segment LED
;and sample ch0 of the a/d in a PIC16C71. The a/d value is displayed
;as a 3 digit decimal value of the a/d input (0 - 255).
;The LEDs are updated every 20msec, the a/d is sampled every 20 msec.
;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
;
;
; Stan D'Souza 5/8/93
*****
LIST P=16C71, F=INHX8M
;
include "picreg.equ"

;
0026 BcdMsD      equ 26
0027 Bcd        equ 27
000C TempC      equ 0x0c      ;temp general purpose regs
000D TempD      equ 0x0d
000E TempE      equ 0x0e
0020 PABuf      equ 0x20
0021 PBBuf      equ 0x21
000F Count      equ 0x0f      ;count
0010 MsdTime    equ 0x10      ;most significant Timer
0011 LsdTime    equ 0x11      ;Least significant Timer
0012 ADFlag     equ 0x12      ;flags related to key pad
0005 ADOver     equ 5         ;bit 5 -> a/d over
002F WBuffer    equ 0x2f
002E StatBuffer equ 0x2e
0001 OptionReg  equ 1
0002 PCL        equ 2

;
push      macro
movwf    WBuffer      ;save w reg in Buffer
swapf   WBuffer      ;swap it
swapf   STATUS,w      ;get status
movwf   StatBuffer    ;save it
endm

;
pop      macro
swapf   StatBuffer,w  ;restore status
movwf   STATUS         ;
swapf   WBuffer,w     ;restore W reg
endm

;
org      0
0000 280D goto      Start      ;skip over interrupt vector
;
org      4
;It is always a good practice to save and restore the w reg,
;and the status reg during a interrupt.
push
0004 00AF movwf    WBuffer      ;save w reg in Buffer
0005 0EAF swapf   WBuffer      ;swap it
0006 0E03 swapf   STATUS,w      ;get status
0007 00AE movwf   StatBuffer    ;save it

0008 2039 call     ServiceInterrupts
;
pop
0009 0E2E swapf   StatBuffer,w  ;restore status
000A 0083 movwf   STATUS         ;


```

Four Channel Digital Voltmeter with Display and Keyboard

```

000B 0E2F          swapf  WBuffer,w          ;restore W reg
000C 0009          retfie
;
; Start
000D 2021          call   InitPorts
000E 202B          call   InitTimers
000F 2036          call   InitAd
; loop
0010 1A92          btfsz  ADFlag,ADOver      ;a/d over?
0011 2013          call   UpdateAd      ;yes then update
0012 2810          goto   loop
;
; UpdateAd
0013 1C88          btfsz  ADCON0,ADIF      ;a/d done?
0014 0008          return      ;no then leave
0015 0809          movf   ADRES,W          ;get a/d value
0016 00A1          movwf  L_byte
0017 01A0          clrfs  H_byte
0018 20AD          call   B2_BCD
0019 0824          movf   R2,W          ;get Lsd
001A 0091          movwf  LsdTime      ;save in LSD
001B 0823          movf   R1,W          ;get Msd
001C 0090          movwf  MsdTime      ;save in Msd
001D 1088          bcf   ADCON0,ADIF      ;clr interrupt flag
001E 1008          bcf   ADCON0,ADON      ;turn off a/d
001F 1292          bcf   ADFlag,ADOver      ;clr flag
0020 0008          return
;
;
;
; InitPorts
0021 1683          bsf   STATUS,RP0      ;select pg 1
0022 3003          movlw  3          ;make RA0-3 digital I/O
0023 0108          movwf  ADCON1      ;
0024 0205          clrfs  TRISA          ;make RA0-4 outputs
0025 0206          clrfs  TRISB          ;make RB0-7 outputs
0026 1283          bcf   STATUS,RP0      ;select page 0
0027 0185          clrfs  PORT_A          ;make all outputs low
0028 0186          clrfs  PORT_B          ;
0029 1585          bsf   PORT_A,3      ;enable MSB digit sink
002A 0008          return
;
;
;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
;end result is that we get a rtcc interrupt every 5msec.
; InitTimers
002B 0190          clrfs  MsdTime      ;clr timers
002C 0191          clrfs  LsdTime      ;
002D 1683          bsf   STATUS,RP0      ;select pg 1
002E 3084          movlw  B'10000100'      ;assign ps to rtcc
002F 0081          movwf  OptionReg      ;ps = 32
0030 1283          bcf   STATUS,RP0      ;select pg 0
0031 3020          movlw  B'00100000'      ;enable rtcc interrupt
0032 008B          movwf  INTCON      ;
0033 3060          movlw  .96          ;preload rtcc
0034 0081          movwf  RTCC          ;start counter
0035 0009          retfie
;
;
; InitAd
0036 30C8          movlw  B'11001000'      ;init a/d
0037 0088          movwf  ADCON0
0038 0008          return
;

```

Four Channel Digital Voltmeter with Display and Keyboard

```

;
ServiceInterrupts
0039 190B          btfsc  INTCON,RTIF          ;rtcc interrupt?
003A 283E          goto   ServiceRTCC         ;yes then service
003B 018B          clrf  INTCON
003C 168B          bsf   INTCON,RTIE
003D 0008          return

;
ServiceRTCC
003E 3060          movlw  .96                  ;initialize rtcc
003F 0081          movwf RTCC
0040 110B          bcf   INTCON,RTIF         ;clr int flag
0041 1C05          btfss PORT_A,0           ;last digit?
0042 2045          call  SampleAd           ;then sample a/d
0043 2071          call  UpdateDisplay      ;else update display
0044 0008          return

;
;
SampleAd
0045 205A          call  SavePorts
0046 204C          call  DoAd                ;do a ad conversion

AdDone
0047 1908          btfsc  ADCON0,GO         ;ad done?
0048 2847          goto  AdDone            ;no then loop
0049 1692          bsf   ADFlag,ADOver     ;set a/d over flag
004A 2067          call  RestorePorts      ;restore ports
004B 0008          return

;
;
DoAd
004C 0186          clrf  PORT_B             ;turn off leds
004D 1683          bsf   STATUS,RP0        ;select pg 1
004E 300F          movlw 0x0f              ;make port a hi-z
004F 0105          movwf TRISA             ;
0050 1283          bcf   STATUS,RP0        ;select pg 0
0051 1408          bsf   ADCON0,ADON      ;start a/d
0052 307D          movlw  .125
0053 2056          call  Wait
0054 1508          bsf   ADCON0,GO        ;start conversion
0055 0008          return

;
;
Wait
0056 008C          movwf TempC             ;store in temp

Next
0057 0B8C          decfsz TempC
0058 2857          goto  Next
0059 0008          return

;
;SavePorts, saves the porta and portb condition during a key scan
;operation.
SavePorts
005A 0805          movf  PORT_A,w          ;Get sink value
005B 00A0          movwf PABuf            ;save in buffer
005C 0185          clrf  PORT_A           ;disable all sinks
005D 0806          movf  PORT_B,w          ;get port b
005E 00A1          movwf PBBuf            ;save in buffer
005F 30FF          movlw 0xff             ;make all high
0060 0086          movwf PORT_B           ;on port b
0061 1683          bsf   STATUS,RP0        ;select page 1
0062 1381          bcf   OptionReg,7      ;enable pull ups
0063 30F0          movlw B'11110000'     ;port b hi nibble inputs
0064 0106          movwf TRISB           ;lo nibble outputs
0065 1283          bcf   STATUS,RP0        ;page 0
0066 0008          return

;
;RestorePorts, restores the condition of porta and portb after a
;key scan operation.

```

Four Channel Digital Voltmeter with Display and Keyboard

```

RestorePorts
0067 0821      movf   PBBuf,w      ;get port n
0068 0086      movwf  PORT_B
0069 0820      movf   PABuf,w      ;get port a value
006A 0085      movwf  PORT_A
006B 1683      bsf    STATUS,RPO    ;select page 1
006C 1781      bsf    OptionReg,7  ;disable pull ups
006D 0205      clrfs TRISA        ;make port a outputs
006E 0206      clrfs TRISB        ;as well as PORTB
006F 1283      bcf    STATUS,RPO    ;page 0
0070 0008      return

;
;
UpdateDisplay
0071 0805      movf   PORT_A,w      ;present sink value in w
0072 0185      clrfs PORT_A        ;disable all digits sinks
0073 390F      andlw  0x0f
0074 008C      movwf  TempC        ;save sink value in tempC
0075 160C      bsf    TempC,4      ;preset for lsd sink
0076 0C8C      rrf    TempC        ;determine next sink value
0077 1C03      btfs   STATUS,CARRY   ;c=1?
0078 118C      bcf    TempC,3      ;no then reset LSD sink
0079 180C      btfs   TempC,0      ;else see if Msd
007A 288C      goto  UpdateMsd     ;yes then do Msd
007B 188C      btfs   TempC,1      ;see if 3rdLsd
007C 2887      goto  Update3rdLsd ;yes then do 3rd Lsd
007D 190C      btfs   TempC,2      ;see if 2nd Lsd
007E 2882      goto  Update2ndLsd ;yes then do 2nd lsd

UpdateLsd
007F 0811      movf   LsdTime,w     ;get Lsd in w
0080 390F      andlw  0x0f         ; /
0081 2890      goto  DisplayOut   ;enable display

Update2ndLsd
0082 20A1      call   Chk2LsdZero  ;msd = 0 & 2 lsd 0?
0083 1D03      btfs   STATUS,Z      ;yes then skip
0084 0E11      swapf LsdTime,w     ;get 2nd Lsd in w
0085 390F      andlw  0x0f         ;mask rest
0086 2890      goto  DisplayOut   ;enable display

Update3rdLsd
0087 20A9      call   ChkMsdZero   ;msd = 0?
0088 1D03      btfs   STATUS,Z      ;yes then skip
0089 0810      movf   MsdTime,w   ;get 3rd Lsd in w
008A 390F      andlw  0x0f         ;mask low nibble
008B 2890      goto  DisplayOut   ;enable display

UpdateMsd
008C 0E10      swapf MsdTime,w     ;get Msd in w
008D 390F      andlw  0x0f         ;mask rest
008E 1903      btfs   STATUS,Z      ;msd != 0 then skip
008F 300A      movlw  0x0a

DisplayOut
0090 2095      call   LedTable     ;get digit output
0091 0086      movwf  PORT_B       ;drive leds
0092 080C      movf   TempC,w      ;get sink value in w
0093 0085      movwf  PORT_A
0094 0008      return

;
;
LedTable
0095 0782      addwf  PCL           ;add to PC low
0096 343F      retlw  B'00111111'  ;led drive for 0
0097 3406      retlw  B'00000110'  ;led drive for 1
0098 345B      retlw  B'01011011'  ;led drive for 2
0099 344F      retlw  B'01001111'  ;led drive for 3
009A 3466      retlw  B'01100110'  ;led drive for 4
009B 346D      retlw  B'01101101'  ;led drive for 5
009C 347D      retlw  B'01111101'  ;led drive for 6
009D 3407      retlw  B'00000111'  ;led drive for 7
009E 347F      retlw  B'01111111'  ;led drive for 8
009F 3467      retlw  B'01100111'  ;led drive for 9

```

Four Channel Digital Voltmeter with Display and Keyboard

```

00A0 3400                retlw      B'00000000'    ;blank led drive
;
;
;
00A1 20A9                call       ChkMsdZero
00A2 1D03                btfss     STATUS,Z      ;msd = 0?
                                ;yes then skip
00A3 0008                return    ;else return
00A4 0E11                swapf     LsdTime,w      ;get 2nd lsd
00A5 390F                andlw     0x0f           ;mask of LSD
00A6 1D03                btfss     STATUS,Z      ;0? then skip
00A7 0008                return
00A8 340A                retlw     .10           ;else return with 10
;
;
;
00A9 0810                movf     MsdTime,w      ;get Msd in w
00AA 1D03                btfss     STATUS,Z      ;= 0? skip
00AB 0008                return    ;else return
00AC 340A                retlw     .10           ;ret with 10
;
;
;
0026                    count    equ    26
0027                    temp     equ    27
;
0020                    H_byte   equ    20
0021                    L_byte   equ    21
0022                    R0       equ    22                ; RAM Assignments
0023                    R1       equ    23
0024                    R2       equ    24
;
;
;
00AD 1003                B2_BCD   bcf     STATUS,0    ; clear the carry bit
00AE 3010                movlw    .16
00AF 00A6                movwf    count
00B0 01A2                clrfs    R0
00B1 01A3                clrfs    R1
00B2 01A4                clrfs    R2
00B3 0DA1                loop16   rlf     L_byte
00B4 0DA0                rlf     H_byte
00B5 0DA4                rlf     R2
00B6 0DA3                rlf     R1
00B7 0DA2                rlf     R0
;
;
;
00B8 0BA6                decfsz   count
00B9 28BB                goto    adjDEC
00BA 3400                RETLW    0
;
;
;
00BB 3024                adjDEC   movlw    R2
00BC 0084                movwf    FSR
00BD 20C5                call    adjBCD
;
;
;
00BE 3023                movlw    R1
00BF 0084                movwf    FSR
00C0 20C5                call    adjBCD
;
;
;
00C1 3022                movlw    R0
00C2 0084                movwf    FSR
00C3 20C5                call    adjBCD
;
;
;
00C4 28B3                goto    loop16
;
;
;
00C5 3003                adjBCD   movlw    3
00C6 0700                addwf    0,W
00C7 00A7                movwf    temp
00C8 19A7                btfsc   temp,3        ; test if result > 7
00C9 0080                movwf    0
00CA 3030                movlw    30
00CB 0700                addwf    0,W
00CC 00A7                movwf    temp

```


Four Channel Digital Voltmeter with Display and Keyboard

```
00CD 1BA7          btfsc temp,7          ; test if result > 7
00CE 0080          movwf 0           ; save as MSD
00CF 3400          RETLW 0

;
;
;          end
```

Four Channel Digital Voltmeter with Display and Keyboard

APPENDIX D

MPASM B0.50

PAGE 1

```
*****
;This program is to demonstrate how to multiplex four 7 segment LED
;digits and a 4X4 keypad using a PIC16C71, along with 4 channels of A/D.
;At start the display reads the A/D value of channel 0. When a key is hit
;the A/D value of the corresponding channel's 0 to 3 is displayed.
;All key hits >= 4 are ignored.
;The LEDs are updated every 20msec, the keypad is scanned at a rate of 20 msec.
;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
;
;
;                               Stan D'Souza 5/8/93
;*****
LIST P=16C71, F=INHX8M
;
include "picreg.equ"
;
000C      TempC      equ    0x0c      ;temp general purpose regs
000D      TempD      equ    0x0d
000E      TempE      equ    0x0e
0020      PABuf      equ    0x20
0021      PBBuf      equ    0x21
000F      Count      equ    0x0f      ;count
0010      MsdTime    equ    0x10      ;most significant Timer
0011      LsdTime    equ    0x11      ;Least significant Timer
;
0012      Flag       equ    0x12      ;general purpose flag reg
0001      #define    keyhit Flag,0    ;bit 0 -> key-press on
0002      #define    DebnceOnFlag,1   ;bit 1 -> debounce on
0003      #define    noentryFlag,2    ;no key entry = 0
0004      #define    ServKey Flag,3   ;bit 3 -> service key
0005      #define    ADOver Flag,4    ;bit 4 -> a/d conv. over
;
0013      Debnce     equ    0x13      ;debounce counter
0014      NewKey     equ    0x14
;
0016      ADTABLE    equ    0x16      ;4 locations are reserved here
;from 0x16 to 0x19f
;
002F      WBuffer    equ    0x2f
002E      StatBuffer equ    0x2e
0001      OptionReg  equ    1
0002      PCL        equ    2
;
;
push      macro
movwf    WBuffer      ;save w reg in Buffer
swapf    WBuffer      ;swap it
swapf    STATUS,w     ;get status
movwf    StatBuffer   ;save it
endm
;
pop      macro
swapf    StatBuffer,w ;restore status
movwf    STATUS       ;
swapf    WBuffer,w    ;restore W reg
endm
;
org      0
0000 280D goto      Start      ;skip over interrupt vector
;
org      4
;It is always a good practice to save and restore the w reg,
;and the status reg during a interrupt.
push
0004 00AF movwf    WBuffer      ;save w reg in Buffer
0005 0EAF swapf    WBuffer      ;swap it
```

Four Channel Digital Voltmeter with Display and Keyboard

```

0006 0E03      swapf      STATUS,w      ;get status
0007 00AE      movwf      StatBuffer   ;save it

0008 204E      call       ServiceInterrupts

0009 0E2E      swapf      StatBuffer,w   ;restore status
000A 0083      movwf      STATUS       ; /
000B 0E2F      swapf      WBuffer,w    ;restore W reg

000C 0009      retfie
;
Start
000D 2038      call       InitPorts
000E 20EA      call       InitAd
000F 2042      call       InitTimers

loop
0010 1992      btfscc    ServKey       ;key service pending
0011 2015      call      ServiceKey    ;yes then service
0012 1A12      btfscc    ADOver       ;a/d pending?
0013 2026      call      ServiceAD     ;yes the service a/d
0014 2810      goto     loop

;
;ServiceKey, does the software service for a keyhit. After a key service,
;the ServKey flag is reset, to denote a completed operation.
ServiceKey
0015 0814      movf      NewKey,w      ;get key value
0016 3C03      sublw    3              ;key > 3?
0017 1C03      btfscc    STATUS,C     ;no then skip
0018 0008      return
;else ignore key
0019 3016      movlw    ADTABLE       ;get top of table
001A 0714      addwf    NewKey,w      ;add offset
001B 0084      movwf    FSR           ;init FSR
001C 0800      movf    0,w           ;get a/d value
001D 00A1      movwf    L_byte
001E 01A0      clrf    H_byte
001F 2102      call    B2_BCD
0020 0824      movf    R2,W          ;get LSD
0021 0091      movwf    LsdTime      ;save in LSD
0022 0823      movf    R1,W          ;get Msd
0023 0090      movwf    MsdTime     ;save in Msd
0024 1192      bcf     ServKey       ;reset service flag
0025 0008      return

;
;This routine essentially loads the ADRES value in the table location
;determined by the channel offset. If channel 0 then ADRES is saved
;in location ADTABLE. If channel 1 then ADRES is saved at ADTABLE + 1 and so on.
ServiceAD
0026 0808      movf    ADCON0,w      ;get adcon0
0027 008C      movwf    TempC       ;save in temp
0028 3008      movlw    B'00001000' ;select next channel
0029 0708      addwf    ADCON0,w    ; /
002A 1A88      btfscc    ADCON0,5    ;if <= ch3
002B 30C0      movlw    B'11000000' ;select ch0
002C 0088      movwf    ADCON0     ;now load adres in the table
002D 3016      movlw    ADTABLE
002E 0084      movwf    FSR        ;load FSR with top
002F 0D8C      rlf     TempC
0030 0D8C      rlf     TempC
0031 0D0C      rlf     TempC,w     ;get in w reg
0032 3903      andlw   3           ;mask off all but last 2
0033 0784      addwf    FSR        ;add offset to table
0034 0809      movf    ADRES,w     ;get a/d value
0035 0080      movwf    0         ;load indirectly
0036 1212      bcf     ADOver      ;clear flag
0037 0008      return

;
InitPorts
0038 1683      bsf     STATUS,RP0   ;select pg 1
0039 3003      movlw   3           ;make RA0-3 digital I/O
003A 0088      movwf   ADCON1     ; /

```

Four Channel Digital Voltmeter with Display and Keyboard

```

003B 0185      clrfs          TRISA          ;make RA0-4 outputs
003C 0186      clrfs          TRISB          ;make RB0-7 outputs
003D 1283      bcf           STATUS,RP0      ;select page 0
003E 0185      clrfs          PORT_A         ;make all outputs low
003F 0186      clrfs          PORT_B         ; /
0040 1585      bsf           PORT_A,3       ;enable MSB digit sink
0041 0008      return

;
;
;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
;end result is that we get a rtcc interrupt every 5msec.
InitTimers
0042 0190      clrfs          MsdTime        ;clr timers
0043 0191      clrfs          LsdTime        ; /
0044 0192      clrfs          Flag          ;clr all flags
0045 1683      bsf           STATUS,RP0      ;select pg 1
0046 3084      movlw        B'10000100'     ;assign ps to rtcc
0047 0081      movwf        OptionReg       ;ps = 32
0048 1283      bcf           STATUS,RP0      ;select pg 0
0049 3020      movlw        B'00100000'     ;enable rtcc interrupt
004A 008B      movwf        INTCON          ;
004B 3060      movlw        .96             ;preload rtcc
004C 0081      movwf        RTCC            ;start counter
004D 0009      retfie

;
ServiceInterrupts
004E 190B      btfsf        INTCON,RTIF     ;rtcc interrupt?
004F 2853      goto         ServiceRTCC     ;yes then service
0050 018B      clrfs          INTCON        ;else clr all int
0051 168B      bsf           INTCON,RTIE    ;
0052 0008      return

;
ServiceRTCC
0053 3060      movlw        .96             ;initialize rtcc
0054 0081      movwf        RTCC            ;
0055 110B      bcf           INTCON,RTIF     ;clr int flag
0056 1805      btfsf        PORT_A,0       ;if msb on then do
0057 205C      call         ScanKeys       ;do a quick key scan
0058 1985      btfsf        PORT_A,3       ;if lsb on then do
0059 20ED      call         SampleAd       ;do a/d sample
005A 20BB      call         UpdateDisplay   ;update display
005B 0008      return

;
;
;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
;Debounce for a given keyhit is also taken care of.
;The rate of key scan is 20msec with a 4.096Mhz clock.
ScanKeys
005C 1C92      btfsf        DebnceOn       ;debounce on?
005D 2862      goto         Scan1          ;no then scan keypad
005E 0B93      decfsz       Debnce          ;else dec debounce time
005F 0008      return                    ;not over then return
0060 1092      bcf           DebnceOn       ;over, clr debounce flag
0061 0008      return                    ;and return

Scan1
0062 20A4      call         SavePorts          ;save port values
0063 30EF      movlw        B'11101111'     ;init TempD
0064 008D      movwf        TempD

ScanNext
0065 0806      movf          PORT_B,w             ;read to init port
0066 100B      bcf           INTCON,RBIF     ;clr flag
0067 0C8D      rrf           TempD         ;get correct column
0068 1C03      btfsf        STATUS,C         ;if carry set?
0069 287C      goto         NoKey          ;no then end
006A 080D      movf          TempD,w             ;else output
006B 0086      movwf        PORT_B         ;low column scan line
006C 0000      nop

```

Four Channel Digital Voltmeter with Display and Keyboard

```

006D 1C0B          btfss      INTCON,RBIF          ;flag set?
006E 2865          goto      ScanNext          ;no then next
006F 1812          btfsc    keyhit            ;last key released?
0070 287A          goto      SKreturn         ;no then exit
0071 1412          bsf      keyhit            ;set new key hit
0072 0E06          swapf    PORT_B,w         ;read port
0073 008E          movwf    TempE            ;save in TempE
0074 207E          call     GetKeyValue      ;get key value 0 - F
0075 0094          movwf    NewKey           ;save as New key
0076 1592          bsf      ServKey          ;set service flag
0077 1492          bsf      DebnceOn        ;set flag
0078 3004          movlw    4
0079 0093          movwf    Debnce          ;load debounce time
                                SKreturn
007A 20B1          call     RestorePorts     ;restore ports
007B 0008          return
                                ;
                                NoKey
007C 1012          bcf      keyhit            ;clr flag
007D 287A          goto      SKreturn
                                ;
                                ;GetKeyValue gets the key as per the following layout
                                ;
                                ;          Col1      Col2      Col3      Col3
                                ;          (RB3)      (RB2)      (RB1)      (RB0)
                                ;
                                ;Row1 (RB4)    0          1          2          3
                                ;
                                ;Row2 (RB5)    4          5          6          7
                                ;
                                ;Row3 (RB6)    8          9          A          B
                                ;
                                ;Row4 (RB7)    C          D          E          F
                                ;
                                GetKeyValue
007E 018C          clrf     TempC
007F 1D8D          btfss    TempD,3          ;first column
0080 2888          goto     RowValEnd
0081 0A8C          incf    TempC
0082 1D0D          btfss    TempD,2          ;second col.
0083 2888          goto     RowValEnd
0084 0A8C          incf    TempC
0085 1C8D          btfss    TempD,1          ;3rd col.
0086 2888          goto     RowValEnd
0087 0A8C          incf    TempC            ;last col.
                                RowValEnd
0088 1C0E          btfss    TempE,0          ;top row?
0089 2892          goto     GetValCom        ;yes then get 0,1,2&3
008A 1C8E          btfss    TempE,1          ;2nd row?
008B 2891          goto     Get4567          ;yes the get 4,5,6&7
008C 1D0E          btfss    TempE,2          ;3rd row?
008D 288F          goto     Get89ab          ;yes then get 8,9,a&b
                                Getcdef
008E 150C          bsf      TempC,2          ;set msb bits
                                Get89ab
008F 158C          bsf      TempC,3          ; /
0090 2892          goto     GetValCom        ;do common part
                                Get4567
0091 150C          bsf      TempC,2
                                GetValCom
0092 080C          movf    TempC,w
0093 0782          addwf   PCL
0094 3400          retlw   0
0095 3401          retlw   1
0096 3402          retlw   2
0097 3403          retlw   3
0098 3404          retlw   4
0099 3405          retlw   5
009A 3406          retlw   6
009B 3407          retlw   7

```

Four Channel Digital Voltmeter with Display and Keyboard

```

009C 3408          retlw      8
009D 3409          retlw      9
009E 340A          retlw     0a
009F 340B          retlw     0b
00A0 340C          retlw     0c
00A1 340D          retlw     0d
00A2 340E          retlw     0e
00A3 340F          retlw     0f

;
;SavePorts, saves the porta and portb condition during a key scan
;operation.
SavePorts
00A4 0805          movf      PORT_A,w          ;Get sink value
00A5 00A0          movwf    PABuf             ;save in buffer
00A6 0185          clrf     PORT_A           ;disable all sinks
00A7 0806          movf      PORT_B,w          ;get port b
00A8 00A1          movwf    PBBuf             ;save in buffer
00A9 30FF          movlw    0xff             ;make all high
00AA 0086          movwf    PORT_B           ;on port b
00AB 1683          bsf     STATUS,RP0       ;select page 1
00AC 1381          bcf     OptionReg,7      ;enable pull ups
00AD 30F0          movlw    B'11110000'     ;port b hi nibble inputs
00AE 0086          movwf    TRISB           ;lo nibble outputs
00AF 1283          bcf     STATUS,RP0       ;page 0
00B0 0008          return

;
;RestorePorts,restores the condition of porta and portb after a key scan opera-
tion.
RestorePorts
00B1 0821          movf      PBBuf,w          ;get port b
00B2 0086          movwf    PORT_B           ;get port a value
00B3 0820          movf      PABuf,w          ;get port a value
00B4 0085          movwf    PORT_A           ;get port a value
00B5 1683          bsf     STATUS,RP0       ;select page 1
00B6 1781          bsf     OptionReg,7      ;disable pull ups
00B7 0185          clrf     TRISA           ;make port a outputs
00B8 0186          clrf     TRISB           ;as well as PORTB
00B9 1283          bcf     STATUS,RP0       ;page 0
00BA 0008          return

;
;
UpdateDisplay
00BB 0805          movf      PORT_A,w          ;present sink value in w
00BC 0185          clrf     PORT_A           ;disable all digits sinks
00BD 390F          andlw    0x0f             ;mask rest
00BE 008C          movwf    TempC            ;save sink value in tempC
00BF 160C          bsf     TempC,4           ;preset for lsd sink
00C0 0C8C          rrf     TempC            ;determine next sink value
00C1 1C03          btfss   STATUS,CARRY      ;c=1?
00C2 118C          bcf     TempC,3           ;no then reset LSD sink
00C3 180C          btfsc   TempC,0           ;else see if Msd
00C4 28D2          goto    UpdateMsd         ;yes then do Msd
00C5 188C          btfsc   TempC,1           ;see if 3rdLsd
00C6 28CF          goto    Update3rdLsd     ;yes then do 3rd Lsd
00C7 190C          btfsc   TempC,2           ;see if 2nd Lsd
00C8 28CC          goto    Update2ndLsd     ;yes then do 2nd lsd

UpdateLsd
00C9 0811          movf      LsdTime,w        ;get Lsd in w
00CA 390F          andlw    0x0f             ;
00CB 28D4          goto    DisplayOut        ;

Update2ndLsd
00CC 0E11          swapf   LsdTime,w         ;get 2nd Lsd in w
00CD 390F          andlw    0x0f             ;mask rest
00CE 28D4          goto    DisplayOut        ;enable display

Update3rdLsd
00CF 0810          movf      MsdTime,w        ;get 3rd Lsd in w
00D0 390F          andlw    0x0f             ;mask low nibble
00D1 28D4          goto    DisplayOut        ;enable display

UpdateMsd
00D2 0E10          swapf   MsdTime,w         ;get Msd in w

```

Four Channel Digital Voltmeter with Display and Keyboard

```

00D3 390F          andlw          0x0f          ;mask rest
                   DisplayOut
00D4 20D9          call           LedTable          ;get digit output
00D5 0086          movwf         PORT_B            ;drive leds
00D6 080C          movf          TempC,w           ;get sink value in w
00D7 0085          movwf         PORT_A
00D8 0008          return

;
;
LedTable
00D9 0782          addwf         PCL                ;add to PC low
00DA 343F          retlw         B'00111111'        ;led drive for 0
00DB 3406          retlw         B'00000110'        ;led drive for 1
00DC 345B          retlw         B'01011011'        ;led drive for 2
00DD 344F          retlw         B'01001111'        ;led drive for 3
00DE 3466          retlw         B'01100110'        ;led drive for 4
00DF 346D          retlw         B'01101101'        ;led drive for 5
00E0 347D          retlw         B'01111101'        ;led drive for 6
00E1 3407          retlw         B'00000111'        ;led drive for 7
00E2 347F          retlw         B'01111111'        ;led drive for 8
00E3 3467          retlw         B'01100111'        ;led drive for 9
00E4 3477          retlw         B'01110111'        ;led drive for A
00E5 347C          retlw         B'01111100'        ;led drive for b
00E6 3439          retlw         B'00111001'        ;led drive for C
00E7 345E          retlw         B'01011110'        ;led drive for d
00E8 3479          retlw         B'01111001'        ;led drive for E
00E9 3471          retlw         B'01110001'        ;led drive for F

;
;
InitAd
00EA 30C0          movlw         B'11000000'        ;internal rc for tad
00EB 0088          movwf         ADCON0            ;
; /
; note that adcon1 is set in InitPorts
00EC 0008          return

;
SampleAd
00ED 20A4          call          SavePorts
00EE 20F4          call          DoAd              ;do a ad conversion
                   AdDone
00EF 1908          btfs         ADCON0,GO          ;ad done?
00F0 28EF          goto         AdDone            ;no then loop
00F1 1612          bsf          ADOver            ;set a/d over flag
00F2 20B1          call          RestorePorts      ;restore ports
00F3 0008          return
;
;
DoAd
00F4 0186          clrf         PORT_B            ;turn off leds
00F5 1683          bsf          STATUS,RPO        ;select pg 1
00F6 300F          movlw         0x0f            ;make port a hi-Z
00F7 0085          movwf         TRISA            ;
; /
00F8 1283          bcf          STATUS,RPO        ;select pg 0
00F9 1408          bsf          ADCON0,ADON       ;start a/d
00FA 307D          movlw         .125
00FB 20FE          call          Wait
00FC 1508          bsf          ADCON0,GO         ;start conversion
00FD 0008          return
;
;
Wait
00FE 008C          movwf         TempC            ;store in temp
                   Next
00FF 0B8C          decfsz       TempC
0100 28FF          goto         Next
0101 0008          return

;
;
0026          equ          26

```

Four Channel Digital Voltmeter with Display and Keyboard

```
0027          temp      equ    27
;
0020          H_byte    equ    20
0021          L_byte    equ    21
0022          R0        equ    22          ; RAM Assignments
0023          R1        equ    23
0024          R2        equ    24
;
;
0102 1003          B2_BCD bcf    STATUS,0          ; clear the carry bit
0103 3010          movlw   .16
0104 00A6          movwf   count
0105 01A2          clrfsz  R0
0106 01A3          clrfsz  R1
0107 01A4          clrfsz  R2
0108 0DA1          loop16 rlf   L_byte
0109 0DA0          rlf     H_byte
010A 0DA4          rlf     R2
010B 0DA3          rlf     R1
010C 0DA2          rlf     R0
;
010D 0BA6          decfsz  count
010E 2910          goto    adjDEC
010F 3400          RETLW   0
;
0110 3024          adjDEC movlw   R2
0111 0084          movwf   FSR
0112 211A          call    adjBCD
;
0113 3023          movlw   R1
0114 0084          movwf   FSR
0115 211A          call    adjBCD
;
0116 3022          movlw   R0
0117 0084          movwf   FSR
0118 211A          call    adjBCD
;
0119 2908          goto    loop16
;
011A 3003          adjBCD movlw   3
011B 0700          addwf   0,W
011C 00A7          movwf   temp
011D 19A7          btfsz  temp,3          ; test if result > 7
011E 0080          movwf   0
011F 3030          movlw   30
0120 0700          addwf   0,W
0121 00A7          movwf   temp
0122 1BA7          btfsz  temp,7          ; test if result > 7
0123 0080          movwf   0          ; save as MSD
0124 3400          RETLW   0
;
;
;
;
end
;
```


SECTION 4

PIC17CXX APPLICATION NOTES

Saving and Restoring Status on Interrupt - AN534	4- 1
Implementing Table Read and Write - AN548	4- 3
Frequency and Resolution Options for PWM Outputs - AN539	4- 7
Using PWM to Generate Analog Output - AN538	4- 15
Using the PWM - AN564	4- 17
Using the Capture Module - AN545	4- 29
Serial Port Utilities - AN547	4- 61
Utility Math Routines - AN544	4- 71
Implementing IIR Digital Filters - AN540	4-121
Implementation of Fast Fourier Transforms - AN542	4-139
Tone Generation - AN543	4-161
Servo Control of a DC Brush Motor - AN532	4-197

Saving and Restoring Status on Interrupt (Implementing a Parameter Stack)

INTRODUCTION

The PIC17C42 has a 16 level deep hardware stack. The program counter is pushed into this stack on interrupts and subroutine calls. However, other key registers are not saved in the stack. Registers such as W, ALUSTA (which has carry, zero and other flag bits) and the bank select register (BSR) must be saved in an interrupt service routine. The following macros, PUSH and POP implement a parameter stack in data memory to accomplish this.

The indirect addressing register FSR0 is used to implement this parameter stack. It is assumed that FSR0 and its control bits are not used or modified elsewhere. The stack pointer (FSR0) is initialized at the highest RAM location (FFh).

```

Main_prog      SETF    FSR0            ;Initialize and dedicate FSR0 as stack pointer
                BCF    ALUSTA,FS0      ;
                BCF    ALUSTA,FS1      ;Set-up FSR0 for auto-dec
                .
                .
                .
PUSH           MACRO
                BCF    ALUSTA,FS0      ;Set-up FSR0 for auto-dec
                MOVFP  ALUSTA,IND0     ;Save ALUSTA first
                MOVFP  BSR,IND0        ;
                MOVFP  W,IND0          ;
                MOVFP  RAM_x, IND0     ;Now save general
                MOVFP  RAM_y, IND0     ;Purpose registers
                ENDM
POP            MACRO
                BSF    ALUSTA,FS0      ;Set-up for auto-inc
                INCF   FSR0            ;
                MOVFP  IND0, RAM_y     ;
                MOVFP  IND0, RAM_x     ;
                MOVFP  IND0,W          ;
                MOVFP  IND0,BSR        ;
                MOVFP  IND0,ALUSTA     ;restore ALUSTA last
                DECF   FSR0            ;Adjust stack pointer
                ENDM
                .
                .
                .
interrupt_routine
                PUSH   ;save registers
                .
                .
                .
;main body of interrupt service
                .
                .
                .
                POP   ;restore status
                RETFIE ;return

```

Saving and Restoring Status on Interrupt

While the macros are quite self-explanatory, the user should note a few subtle points.

1. `MOVFP` instruction does not affect status flags while `MOVFP` does.
2. `MOVFP` and `MOVFP` are used such that any register can be saved and restored. Note that register being saved or restored has address `f` (which can be `00h` to `FFh`) and other address is `IND0` (indirect), therefore can be any address.
3. `FSR` auto-increments or auto-decrements after the operation ('post'). Therefore in the `POP` macro pre-increment is simulated.

Using this scheme, interrupts and subroutine calls can be nested, since the stack will grow and shrink. The stack can be used to pass parameters to subroutines.

*Author: Stanley D'Souza
Logic Products Division*

Implementing Table Read and Table Write

INTRODUCTION

This application brief discusses how to read data from program memory to data memory and write data from data memory to program memory.

RETLW K Instruction

As in all PIC17CXX family parts, the simplest method used to retrieve data from program memory to data memory is to use the RETLW K instruction. For example:

```

; simple program to transfer
; table values to PortB
Main
    movlw 5,W      ;load offset
    call SimpleTableRead
    movwf PortB   ;output to PortB
    .
    .
    .
SimpleTableRead
    addwf PC      ;add offset to PC
Table retlw 0    ;return a known
                ;table value based
                ;on the OFFSET.
    .
    .
    .
    retlw 10

```

In the example above, OFFSET is loaded with the required offset to the Table and the subroutine SimpleTableRead is called. The table value is returned in the W register. In this manner program memory can be transferred to data memory.

Table Read Instruction

The PIC17C42 has an expanded instruction set which includes the TABLRD and TLRD instructions. These instructions are specifically constructed to transfer data from program memory to data memory.

The instruction syntax is: TABLRD t,i,f.

The sequence in which this instruction is executed is as follows:

- if t = 1 then the high byte of the table latch (TBLATH) is loaded in the file register f.
- else (if t = 0) the low byte of the table latch (TBLATL) is loaded in the file register f.
- next, the 16 bit data pointed to by the table pointer (TBLPTR) is loaded into the table latch.
- lastly, if i = 1 the table pointer (TBLPTR) is incremented.

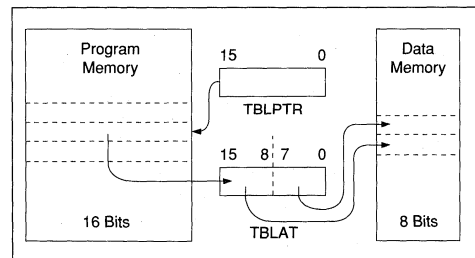
Note: The first time this instruction is executed in a sequence, the table latch will not be initialized, hence an unknown value will be loaded in the file register. This is not a problem if the user overwrites the same f register in the next subsequent instruction.

The instruction syntax is: TLRD t,f.

The sequence in which this instruction is executed is as follows:

- if t = 1 then the high byte of the table latch (TBLATH) is loaded in the file register f.
- else (if t = 0) the low byte of the table latch (TBLATH) is loaded in the file register f.

FIGURE 1 - TABLE READ



Implementing Table Read and Write in PIC17C42

Read In Line

A simple method of transferring data from program memory to data memory is to use the `tablrd` and `tlrd` instruction in sequence as shown in the example below:

```
;transfer 6 bytes of data in program memory at
0x500, to ;data memory at 0x80:
```

```
ReadInLine
    movlw 05          ;load table pointer with
                    ; 0x500

    movwf TBLPTRH   ;
                    ; /
    clrf TBLPTRL   ;
                    ; /
    tablrd 0,1,0x80 ;get 16 bit value in
                    ;table latch.

    tlrd 0,0x80     ;low byte (1st) E 80
    tablrd 1,1,0x81 ;high byte (2nd) E 81
    tlrd 0,0x82     ;3rd byte E 82
    tablrd 1,1,0x83 ;4th byte E 83
    tlrd 0,0x84     ;5th byte E 84
    tablrd 1,1,0x85 ;6th byte E 85
```

Reading a Block of Data

In instances where a block of N bytes needs to be transferred from program memory to data memory, the `tablrd` and `tlrd` instruction need to be included in a loop which checks for N transfers.

```
;transfer 'COUNT' bytes (even values only) of
;data at program memory 'MESSAGE' to data memory
;at:
```

```
;'RAM_BUFFER'
ReadBlock
    movlw high MESSAGE ;load table pointer
    movpf W,TBLPTRH   ;
                    ; /
    movlw low MESSAGE ;
                    ; /
    movpf W,TBLPTRL   ;
                    ; /
    bcf ALUSTA,5      ;enable post auto
                    ;increment of FSR0

    movlw RAM_BUFFER  ;initialize FSR0 to
                    ;RAM_BUFFER

    movfp W,FSR0      ;
                    ; /
    movlw COUNT/2     ;initialize count
    tablrd 1,1,RAM_BUFFER ;initialize table
                    ;latch

ReadBlockLoop
    tlrd 1,0x00       ;do indirect read of
                    ;high byte
    tablrd 0,1,0x00   ;do indirect read of
                    ;low byte

    decfsz W          ;check if count = 0
    goto ReadBlockLoop ;no then do next
    return           ;else end of
                    ;transfer.
```

Program	Code Size	Transfer Rate
Simple Table Read (using RETLW)	$N + 3$	6 cycles/byte
Read In-Line	$4 + N + N/2$	1.5 cycles/byte
Read Block (using loop)	$14 + N/2$	3 cycles/byte

N = Number of bytes to transfer

	Code size		
	Simple Table Read	Read In-Line	Read Block
N = 10	13	19	19
N = 20	23	34	24

Conclusion:

In cases where the number of bytes to be transferred is small, the Read In-Line offers small code size for fast transfer rate. However, as the number of bytes to be transferred increases, the Read Block offers optimum code size for a decent transfer rate.

Implementing Table Read and Write in PIC17C42

Table Write Instruction

The PIC17C42 has a `TABLWT` and a `TLWT` instruction which transfer data from data memory to program memory. Note in cases where the table pointer points to internal EPROM, the table write instruction will try to program the EPROM, hence the programming voltage must be present on the `VPP` line to successfully program the part.

The instruction syntax is: `TABLWT t,i,f`.

The sequence in which this instruction is executed is as follows:

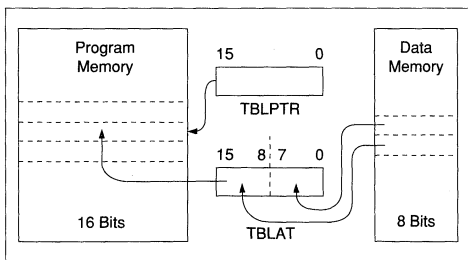
- if `t=1` then the file register `f` is loaded to the high byte of the table latch (`TBLATH`).
- else (if `t=0`) the file register `f` is loaded to the low byte of the table latch (`TBLATL`).
- next, the 16 bit data in the table latch is transferred to the program memory pointed to by the table pointer (`TBLPTR`).
- lastly, if `i=1` the table pointer (`TBLPTR`) is incremented.

The instruction syntax is: `TLWT t,f`.

The sequence in which this instruction is executed is as follows:

- if `t=1` then the file register `f` is loaded to the high byte of the table latch (`TBLATH`).
- else (if `t=0`) the file register `f` is loaded to the low byte of the table latch (`TBLATL`).

FIGURE 2 - TABLE WRITE



Write in Line

A simple method of transferring data from data memory to program memory is to use the `tablwt` and `tlwt` instruction in sequence as shown in the example below:

```

;transfer 6 bytes of data in data memory at 0x80,
to ;program memory at 0x5000:
ReadInLine
    movlw 50          ;load table pointer with
                    ;0x5000
    movwf TBLPTRH    ;
    clrf TBLPTRL    ;
    tlwt 1,0x80     ;high byte E table latch.
    tablwt 0,1,0x81 ;low byte E table latch;
                    ;latch E prog. mem.
    tlwt 1,0x82     ;3rd and 4th byte E prog.
                    ;mem.
    tablwt 0,1,0x83 ;
    tlwt 1,0x84     ;5th and 6th byte E prog.
                    ;mem.
    tablwt 0,1,0x85 ;

```

Writing a Block of Data

In instances where a block of `N` bytes needs to be transferred from data memory to program memory, the `tablwt` and `tlwt` instruction need to be included in a loop which checks for `N` transfers.

;transfer 'COUNT' bytes (even values only) of data at ;program memory at 'RAM_BUFFER' to program memory ;at 'MESSAGE'

```

WriteBlock
    movlw high MESSAGE ;load table pointer
    movwf W,TBLPTRH   ;
    movlw low MESSAGE ;
    movwf W,TBLPTRL   ;
    bcf ALUSTA,5      ;enable post auto
                    ;increment of FSR0
    movlw RAM_BUFFER  ;initialize FSR0 to
                    ;RAM_BUFFER
    movfp W,FSR0      ;
    movlw COUNT/2     ;initialize count
WriteBlockLoop
    tlwt 1,0x00       ;high byte E table
                    ;latch
    tablwt 0,1,0x00   ;low byte E table
                    ;latch;
                    ;table latch E prog.
                    ;mem.
    decfsz W          ;check if count = 0
    goto WriteBlockLoop ;no then do next
    return            ;else end of transfer.

```

Author: Stanley D'Souza
Logic Products Division

Implementing Table Read and Write in PIC17C42

NOTES:

Frequency and Resolution Options for PWM Outputs

INTRODUCTION

The PIC17C42 is equipped with two high frequency Pulse Width Modulation (PWM) outputs. In a pulse width modulated signal the period of the signal is (usually) kept fixed, while the duty cycle is varied. In this application note, we will discuss options in selecting their frequency and resolution.

This application brief assumes that internal clock is used for the time-base, which is typically the preferred set-up. Also, throughout this application brief, PWM1 output is used in examples, timer1 is assumed to be the time-base.

Definition of terms:

Period of a PWM output is the duration after which the PWM pattern will repeat itself.

Frequency of a PWM output is = 1/Period.

Resolution of a PWM output is the granularity with which the duty cycle can be modulated.

In the case of the PIC17C42, when using PWM1 with timer1 as time-base the:

$$\text{PWM1 period} = [(PR1) + 1] \times 4\text{tosc}$$

$$\text{PWM1 duty cycle} = (DC1) \times \text{tosc}$$

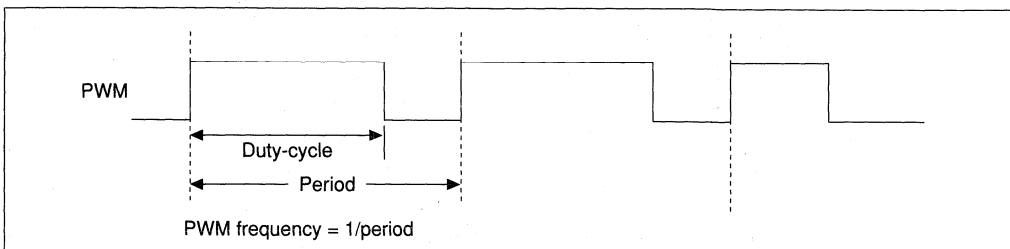
where PR1 = period register for timer1

DC1 = PW1DCH, PW2DCL concatenated (10-bit value)

tosc = oscillator period

At 16 MHz oscillator frequency, tosc = 62.5 ns. The user can control the frequency of the PWM output by altering the 'period' value of the time-base. For example, if period is chosen to be 100 tosc (PR1 = 18h), then PWM frequency is 1/(100 x 62.5) ns = 160 KHz. Note however that duty cycle resolution is a little less than 7-bits.

FIGURE 1 - PWM OUTPUT



Useful and Common PWM Modes

While a variety of period values can be selected, the following modes would be most commonly used:

10-Bit Mode: In this mode PWM duty cycle has full 10-bit resolution (maximum offered by the PIC17C42). The period register PR1 is set at FFh. PWM period is $1024\text{tosc} = 64 \mu\text{s}$. PWM frequency is 15.625 KHz. The user must write both PW1DCH and PW1DCL to update PWM output. See Appendix A for an example that code modules 10-bit resolution PWM output (PWM10.LST).

8-Bit Hi-Resolution Mode: In this mode, the user has only an 8-bit quantity to write to the duty-cycle register. Period register is set at 3Fh (63 decimal), such that PWM period is 256 tosc. To write the 8-bit duty-cycle value, first the 8-bit is right shifted two bits. The upper six bits are written to PW1DCH and the lower two bits are written to PW1DCL as follows:

```

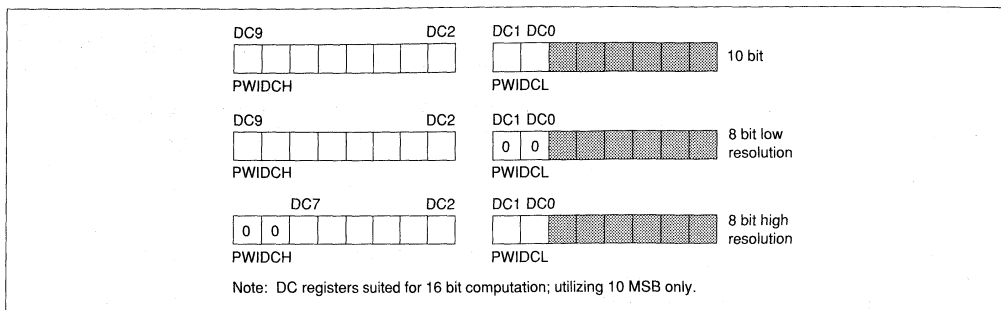
;8-bit duty-cycle value is in W reg
CLRF  TEMP          ;
RRCF  WREG          ;
RRCF  TEMP          ;
RRCF  WREG          ;
RRCF  TEMP          ;Shift right twice
ANDLW 00111111b    ;Mask off two-high bits
MOVVF  WREG,PW1DCH ;Write duty-cycle values
MOVVF  TEMP,PW1DCL ;

```

Note that in 8-bit, hi-resolution mode, maximum PWM frequency is attained. For example, at 16 MHz clock, PWM period = 256 tosc = 16 μs ; PWM frequency = 62.5 KHz. See appendix B for an example code that generates 8-bit low high resolution PWM output (PWM8HI.LST).

PWM Frequency and Resolution

FIGURE 2 - VARIOUS PWM MODES



8-Bit Low Resolution Mode

In this mode, the user still has only an 8-bit quantity to write to duty cycle register. However, the desired frequency of the PWM output is less, due to the nature of the application. For example, if the PWM output is being used to drive a motor through a power stage, the power transistors (or devices) due to their switching time will prefer PWM frequency not to exceed certain frequency. In the previous section, we derived an 8-bit resolution PWM output at 62.5 KHz.

To attain a low-resolution PWM output, the PW1DCL is always kept at zero. The 8-bit value is written to PW1DCH. The period (PR1) is set at FFh, i.e. 256 Tcy equals 1024 tosc (15.625 KHz). See Appendix C for an example code that produces 8-bit low resolution PWM output (PWM8LO.LST).

Choosing Resolution and Frequency of PWM Output

Actually, the resolution and the frequency of the PWM output is selectable within certain limits. The user will need to first define the requirements based on the application. There may be an upper limit to the frequency if the PWM is being used to drive motors. On the

other hand, if the PWM is being filtered to generate an analog signal, higher frequency may be desirable. In any case, the lowest frequency achievable (using internal clock for the timer) is (OSC freq/1024). At 16 MHz oscillator input, the lowest PWM frequency possible is 15.625 KHz. At resolutions less than 10-bit higher frequencies are possible (see Figure 3). For example if 7-bit resolution is chosen, then the PWM frequencies can be 15-625 KHz, 31.25 KHz, 62.5 KHz or 125 KHz. The reader will note that its how the 7-bits are placed within the 10-bit possible duty cycle value.

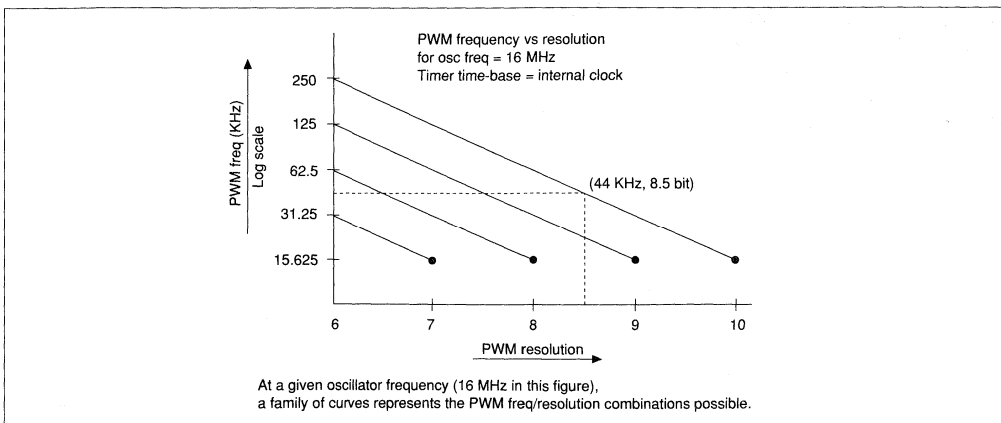
Conversely, if a certain frequency is desired, such as 44 KHz, then referring to Figure 1, resolution can be 8.5-bit or 7.5-bit or 6.5-bit etc.

Summary

The frequency and resolution of the PWM outputs of the PIC17C42 can be traded off against each other to best suit the application. The oscillator frequency can also be varied to adjust PWM frequency, if necessary. External clock should be used as timer time-base to generate very low frequency PWM output.

Authors: Stanley D'Souza
Sumit Mitra
Logic Products Division

FIGURE 3 - PWM FREQ VS RESOLUTION



PWM Frequency and Resolution

APPENDIX A: PWM10.LST

MPASM B0.54

PAGE 1

PULSE WIDTH MODULATION 10 BIT RESOLUTION

```
TITLE "PULSE WIDTH MODULATION 10 BIT RESOLUTION"
LIST P=17C42, C=80, T=ON

include "17c42.h"

0021 PWM_HI equ 0x21
0020 PWM_LO equ 0x20
0022 TEMP equ 0x22

;The user would generate a 16 bit value which is saved in r
;locations PWM_HI and PWM_LO byte. In 10 bit mode, the prog
;transfers these values directly to the Duty Cycle (DC) reg
;generate the required 10 bit PWM.
;
;
;This is a short program to demonstrate how to generate PWM
;10 bit resolution. Since a 10Mhz crystal was used in the t
;The max. period = 1024x100nS = 102.4 uS or 9.8 KHz. This p
;keeps the period constant and varies the duty cycle (which
;to the most significant 10 bits of the 16 bit value PWM_LO
;This program is interrupt driven, i.e. the update to the D
;is done in the rtcc interrupt, which then enables the pwm
;The period update is done during the pwm interrupt. The pw
;ramps up from 0% to 100% duty cycle and then repeats. The
;sweep takes approx. 52 secs.
;
;
ORG 0
goto start

;
ORG 0x10 ;vector for rtcc interrupt
rtcc_int
goto service_rtcc ;service rtcc

;
ORG 0x0020 ;vector for pwm interrupt
pwm_int
goto service_pwm ;service pwm only

;
ORG 0x0030

;initialize internal hardware to generate the output
;for 10 bit resolution pwm.
init_pwm10
0030 B802 movlb 2
0031 2910 clrfs tmr1 ;clear timer 1
;used to "drive" pwm1

0032 2B14 setf prl ;set period=9.8 khz
0033 B803 movlb 3
0034 7221 movfp PWM_HI,pwldch ;load duty cyl. hi byte
0035 7020 movfp PWM_LO,pwldcl ;load duty cycle lo byte
0036 2916 clrfs tcon1 ;tmr1 inc. internally
;as 8 bit counter

0037 B01B movlw 00010001B ;start tmr1 and
0038 4A17 movpf wreg,tcon2 ;enable pwm1
0039 B801 movlb 1
003A 2917 clrfs pie ;clr all int. enables
003B 2916 clrfs pir ;clear all interrupts
003C 8307 bsf _peie ;except peripheral int.
003D 0005 retfie

;
;
```

PWM Frequency and Resolution

```

; everytime a new value is written to the PWM_HI, PWM_LO reg
; tmr1 interrupts is enabled. The DC value are written just
; the "pwm interrupt" is enabled. Here the new period regist
; updated. In this example, period is kept constant at 0xff
service_pwm
    ;if the period changed, write new value here.
003E B802        movlb 2           ;select bank 2
003F 2B14        setf prl           ;period <- 0xff
0040 B801        movlb 1           ;disable tmr1 int
0041 8C17        bcf _tmlie        ; /
0042 0005        retfie

;
; This part of the program is basically used to simulate a
; which would be used to drive the pwm output.
;
; the rtcc is set up to interrupt every 52 ms.
init_rtcc
0043 B00B        movlw 00100000b    ;set up rtcc timer
0044 650A        movfp wreg,rtccsta ; /
0045 290B        clrfs rtcc1       ;clear rtcc
0046 290C        clrfs rtccch      ; /
0047 B080        movlw 0x80        ;start pwm at 50%
0048 0121        movwf PWM_HI      ; /
0049 2920        clrfs PWM_LO      ; /
004A 8107        bsf _rtcie        ;enable rtcc int.
004B 0002        return

;
; Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
; Only the 10 most significant bits are incremented. Once th
;
service_rtcc
004C 8D07        bcf _rtcir        ;reset int flag
                ;do a pseudo inc of the 10 bit PWM_HI, PWM_LO.
004D 8804        bcf _carry        ;clear carry
004E B00B        movlw 01000000b    ;load lsb for 10 bit
004F 0F20        addwf PWM_LO,1     ;add to LSB
0050 9804        btfscc _carry     ;carry?
0051 1521        incf PWM_HI        ;yes then inc PWM_HI
                ;now load the values into the Duty Cycle Registers

0052 B803        movlb 3           ;bank 3
0053 7020        movfp PWM_LO,pwldcl ;load lo value
0054 7221        movfp PWM_HI,pwldch ;load hi value
0055 B801        movlb 1           ;
0056 8417        bsf _tmlie        ;enable tmr1 int
0057 0005        retfie

;
;
start
0058 8406        bsf _glintd       ;disable interrupts
0059 E043        call init_rtcc     ;initialize the RTCC tmr
                ;for test purposes
005A E030        call init_pwm10    ;initialize pwm
005B C05B        loop goto loop     ;spin wheels
;

END

Errors : 0
Warnings : 0
```

PWM Frequency and Resolution

Appendix B: PWM8HI.LST

MPASM B0.54

PAGE 1

PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION

```
TITLE "PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION"
LIST P=17C42, C=80, T=ON

include "17c42.h"

0021 PWM_HI equ 0x21
0020 PWM_LO equ 0x20
0022 TEMP equ 0x22

;The user would generate a 16 bit value which is saved in r
;locations PWM_HI and PWM_LO byte. In 8 bit hi-res mode, th
;transfers the 8 bit values to the lo Duty Cycle (DC) regis
;generate the required 8 bit hi-res PWM.
;
;
;This is a short program to demonstrate how to generate PWM
;8 bit resolution. Since a 10Mhz crystal was used in the te
;The max. period = 256x100nS = 25.6uS or 39KHz. This progra
;keeps the period constant and varies the duty cycle (which
;to the most significant 10 bits of the 16 bit value PWM_LO
;This program is interrupt driven, i.e. the update to the D
;is done in the rtcc interrupt, which then enables the pwm
;The period update is done during the pwm interrupt. The pw
;ramps up from 0% to 100% duty cycle and then repeats. The
;sweep takes approx. 13.3 secs.
;
;
; ORG 0
0000 C063 goto start
;
; ORG 0x10 ;vector for rtcc interrupt

rtcc_int
0010 C054 goto service_rtcc ;service rtcc
;
; ORG 0x0020 ;vector for pwm interrupt

pwm_int
0020 C046 goto service_pwm ;service pwm only
;
; ORG 0x0030
;
;initialize internal hardware to generate the pwm output
init_pwm8hi
0030 B802 movlb 2
0031 2910 clrfl tmr1 ;clear timer 1
;used to "drive" pwml
0032 B062 movlw 62 ;set period=39khz
0033 0114 movwfl pr1 ; /
0034 B803 movlb 3
0035 2922 clrfl TEMP ;TEMP = mask for pwldcl
0036 6A21 movwfl PWM_HI,wreg ;get duty cyl. hi byte
0037 190A rrcfl wreg ;rotate hi through carry
0038 1922 rrcfl TEMP ;rotate into lo byte
0039 190A rrcfl wreg ;repeat for 2nd lsb
003A 1922 rrcfl TEMP ; /
003B B53F andlw b'00111111' ;mask hi bits
003C 4012 movwfl W,pwldch ;save in high
003D 7022 movwfl TEMP,pwldcl ;save in low
003E 2916 clrfl tcon1 ;tmr1 inc. internally
;as 8 bit counter
003F B011 movlw b'00010001' ;start tmr1 and
0040 4017 movwfl W,tcon2 ;enable pwml
0041 B801 movlb 1
```

PWM Frequency and Resolution

```
0042 2917          clrfs    pie           ;clr all int. enables
0043 2916          clrfs    pir           ;clear all interrupts
0044 8307          bsfs     _peie        ;except peripheral int.
0045 0005          retfies

;
;
;everytime a new value is written to the PWM_HI, PWM_LO reg
;tmr1 interrupts is enabled. The DC value are written just
;the "pwm interrupt" is enabled. Here the new period regist
;updated. In this example, period is kept constant at 62 Tc
service_pwm
;if the period changed, write new value here.
0046 B802          movlb   2             ;select bank 2
0047 B062          movlw   62           ;period = 62 Tcyl.
0048 0114          movwf  prl           ; /
0049 B801          movlb   1             ;disable tmr1 int
004A 8C17          bcf     _tm1ie        ; /
004B 0005          retfies

;
;This part of the program is basically used to simulate a
;which would be used to drive the pwm output.
;
;the rtcc is set up to interrupt every 52 mS.
init_rtcc
004C B020          movlw   b'00100000'    ;set up rtcc timer
004D 650A          movfpl wreg,rtccsta    ; /
004E 290B          clrfs    rtcc1        ;clear rtcc
004F 290C          clrfs    rtcc2        ; /
0050 B031          movlw   31           ;init pwm at 50%
0051 0121          movwf  PWM_HI        ;save in high
0052 8107          bsfs     _rtcie        ;enable rtcc int.
0053 0002          return

;
;Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
;Only the 8 most significant bits are incremented.
;
service_rtcc
0054 8D07          bcf     _rtcir        ;reset int flag
;do a pseudo inc of the 8 bit PWM_HI.
0055 1521          incf   PWM_HI        ;inc PWM HI
;now load the values into the Duty Cycle

0056 B803          movlb   3             ;bank 3
0057 2922          clrfs    TEMP         ;TEMP = mask for pwldcl
0058 6A21          movfpl PWM_HI,wreg    ;get duty cyl. hi byte
0059 190A          rrcf   wreg         ;rotate hi through carry
005A 1922          rrcf   TEMP         ;rotate into lo byte
005B 190A          rrcf   wreg         ;repeat for 2nd lsb
005C 1922          rrcf   TEMP         ; /
005D B53F          andlw  b'00111111'    ;mask hi bits
005E 4A12          movfpl wreg,pwldch    ;save in high
005F 7022          movfpl TEMP,pwldcl   ;save in low
0060 B801          movlb   1             ;
0061 8417          bsfs     _tm1ie        ;enable tmr1 int
0062 0005          retfies

;
;
start
0063 8406          bsfs     _glintd       ;disable interrupts
0064 E04C          call   init_rtcc      ;initialize the RTCC tmr
;for test purposes
0065 E030          call   init_pwm8hi    ;initialize 8 bit pwm
0066 C066          loop   goto loop    ;spin wheels.

;
END

Errors : 0
Warnings : 0
```

PWM Frequency and Resolution

Appendix C: PWM8LO.LST

MPASM B0.54

PAGE 1

PULSE WIDTH MODULATION 8 BIT LOW RESOLUTION

TITLE "PULSE WIDTH MODULATION 8 BIT LOW RESOLUTION"
LIST P=17C42, T=ON, C=80

```
include "17c42.h"

0021          PWM_HI equ    0x21
0020          PWM_LO equ    0x20
0022          TEMP  equ    0x22

;The user would generate a 16 bit value which is saved in r
;locations PWM_HI and PWM_LO byte. In 8 bit lo-res mode, th
;transfers the 8 hi-byte value directly to the PW1DCH regis
;
;
;This is a short program to demonstrate how to generate PWM
;8 bit low resolution. Since a 5.068Mhz crystal was used in
;The max. period = 1024x100nS = 102.4 uS or 9.8 KHz. This p
;keeps the period constant and varies the duty cycle (which
;to the most significant 8 bits of the 16 bit value PWM_LO&
;This program is interrupt driven, i.e. the update to the D
;is done in the rtcc interrupt, which then enables the pwm
;The period update is done during the pwm interrupt. The pw
;ramps up from 0% to 100% duty cycle and then repeats. The
;sweep takes approx. 52 secs.
;
;
;          ORG    0
0000 C053      goto    start

;          ORG    0x10          ;vector for rtcc interrupt

rtcc_int
0010 C04C      goto    service_rtcc    ;service rtcc

;          ORG    0x0020        ;vector for pwm interrupt

pwm_int
0020 C03E      goto    service_pwm     ;service pwm only

;          ORG    0x0030

;initialize internal hardware to generate the output
;for 8 bit low resolution pwm
init_pwm8lo
0030 B802      movlb   2
0031 2910      clrfs  tmr1          ;clear timer 1
                                ;used to "drive" pwm1

0032 2B14      setf    pr1          ;set period=9.8 khz
0033 B803      movlb   3
0034 7221      movfpc PWM_HI,pw1dch ;load duty cyl. hi byte
0035 2910      clrfs  pw1dcl       ;clear lo byte
0036 2916      clrfs  tcon1        ;tmr1 inc. internally
                                ;as 8 bit counter

0037 B011      movlw   b'00010001'  ;start tmr1 and
0038 4A17      movfpc wreg,tcon2    ;enable pwm1
0039 B801      movlb   1
003A 2917      clrfs  pie          ;clr all int. enables
003B 2916      clrfs  pir          ;clear all interrupts
003C 8307      bsf    _peie        ;except peripheral int.
003D 0005      retfde

;
;
```

PWM Frequency and Resolution

```
; everytime a new value is written to the PWM_HI, PWM_LO reg
; tmr1 interrupts is enabled. The DC value are written just
; the "pwm interrupt" is enabled. Here the new period regist
; updated. In this example, period is kept constant at 0xff
service_pwm
    ; if the period changed, write new value here.
003E B802        movlb 2           ;select bank 2
003F 2B14        setf prl           ;period <- 0xff
0040 B801        movlb 1           ;disable tmr1 int
0041 8C17        bcf _tmr1ie      ; /
0042 0005        retfie

;
; This part of the program is basically used to simulate a
; which would be used to drive the pwm output.
;
; the rtcc is set up to interrupt every 52 mS.
init_rtcc
0043 B020        movlw b'00100000'   ;set up rtcc timer
0044 6500        movfp W,rtccsta    ; /
0045 290B        clrfs rtcc1       ;clear rtcc
0046 290C        clrfs rtccch      ; /
0047 B080        movlw 0x80        ;begin PWM at 50% dc
0048 0121        movwf PWM_HI      ; /
0049 2920        clrfs PWM_LO      ; /
004A 8107        bsf _rtccie       ;enable rtcc int.
004B 0002        return

;
; Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
; Only the 8 most significant bits are incremented.
;
service_rtcc
004C 8D07        bcf _rtccir       ;reset int flag
                                        ;do a inc of the 8 bit PWM_HI
004D 1521        incf PWM_HI
                                        ;now load the values into the Duty Cycle
004E B803        movlb 3
004F 7221        movfp PWM_HI,pwldch ;load hi byte
0050 B801        movlb 1
0051 8417        bsf _tmr1ie      ;enable tmr1 int
0052 0005        retfie

;
;
start
0053 8406        bsf _glintd       ;disable interrupts
0054 E043        call init_rtcc    ;initailize the RTCC tmr
                                        ;for test purposes
0055 E030        call init_pwm8lo  ;initialize pwm
0056 C056        loop goto loop    ;spin wheels.

;

END

Errors : 0
Warnings : 0
```

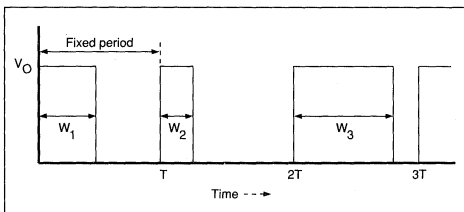

Using PWM to Generate Analog Output

Pulse Width Modulated (PWM) waveforms which are basically digital waveforms, can be used as cheap Digital-to-Analog (D/A) converters with few external components. A wide variety of microcontroller applications exists that need analog output but do not require high resolution D/A converters. Some speech applications (talk back units, speech synthesis systems in toys, etc.) also do not require high resolution D/A converters. For these applications, Pulse Width Modulated outputs may be used to convert to analog outputs.

Conversion of PWM waveforms to analog signals involves the use of analog low-pass filters. This brief application note describes the design criteria of the analog filters necessary and the requirements of the PWM frequency. Later in this application note, a simple RC low-pass filter is designed to convert pulse-width modulated speech signals of 4 KHz bandwidth.

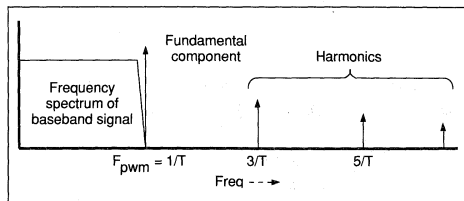
In a typical PWM signal, the base frequency is fixed, but the pulse width is a variable. The pulse width is directly proportional to the amplitude of the original unmodulated signal. In other words, in a PWM signal, the frequency of the waveform is a constant while the duty cycle varies (from 0 % to 100 %) according to the amplitude of the original signal. A typical PWM signal is shown in Figure 1.

FIGURE 1 - A TYPICAL PWM WAVEFORM



A Fourier analysis of a typical PWM signal (like the one depicted in Figure 1) shows that there is a strong peak at frequency $F_n = 1/T$. Other strong harmonics also exist at $F = K/T$, where K is an integer. These peaks are unwanted noise and should be eliminated. This requires that the PWM signal be low-pass filtered, thus eliminating these inherent noise components (see Figure 2).

FIGURE 2 - FREQUENCY SPECTRUM OF A PWM SIGNAL

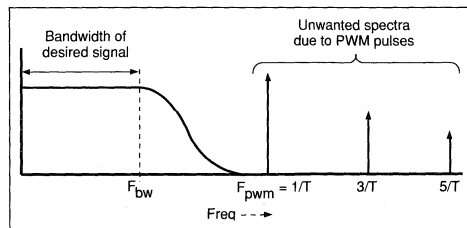


The band-width of the desired signal should thus be

$$fbw \ll (f_{pwm} = 1/T)$$

If fbw is selected such that $fbw = f_{pwm}$, then the external low-pass filter should be a brick-wall type filter. Brick-wall type analog filters are very difficult and expensive to build. So, for practical purpose, the external low-pass filter should be as shown in Figure 3.

FIGURE 3 - EXTERNAL LOW-PASS FILTER



This means, $fbw \ll f_{pwm}$

$$\text{or } f_{pwm} \gg fbw$$

$$\Rightarrow f_{pwm} = K * fbw \quad (1)$$

where, K is a constant such that $K \gg 1$

The value of K should be chosen depending upon by how many dB the inherent fundamental noise component of PWM be rejected. An example follows :

Example : It is required to design a simple RC low-pass filter to obtain an analog output from a pulse width modulated speech signal of bandwidth 4 KHz.

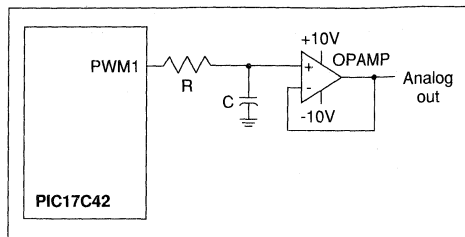
From eqn (1), choosing arbitrarily $K = 5$,

$$f_{pwm} = K * fbw = 5 * 4 \text{ KHz} = 20 \text{ KHz.}$$



Using PWM to Generate Analog Output

FIGURE 4 - RC FILTER CONNECTED TO PWM1 OF PIC17C42



Choosing, the -3 dB point at 4 KHz, and using the relation $RC = 1/(2\pi f)$, we get $R = 4\text{ K}\Omega$, if C is chosen as $0.01\ \mu\text{F}$:

$$R = 4.0\text{ K}\Omega$$
$$C = 0.01\ \mu\text{F}$$

Since the PWM frequency is selected as 20 KHz, the fundamental noise peak to be filtered is at 20 KHz. Now, lets calculate by how many dB the main peak of PWM signal is cut-off at 20 KHz :

$$(\text{dB})20\text{KHz} = -10\log[1 + (2\pi f.RC)^2] = -14\text{ dB.}$$

For many applications, this rejection of -14 dB will not suffice. Therefore instead of a simple RC low-pass filter, a higher order active low-pass filter may be necessary. Or if the microcontroller is capable of modulating at higher PWM frequencies, the rejection of noise will be more.

For example, using 8-bit resolution, the PIC17C42 can generate PWM frequency of 62.5 KHz.

At this frequency the attenuation of the PWM frequency is:

$$(\text{dB})62.5\text{KHz} = -10\log[1 + (2\pi f.RC)^2] = -24\text{ dB.}$$

The higher frequency of the PIC17C42 PWM outputs makes it easier to generate analog output.

Author: Amar Palacherla
Logic Products Division

Using the PWM

INTRODUCTION

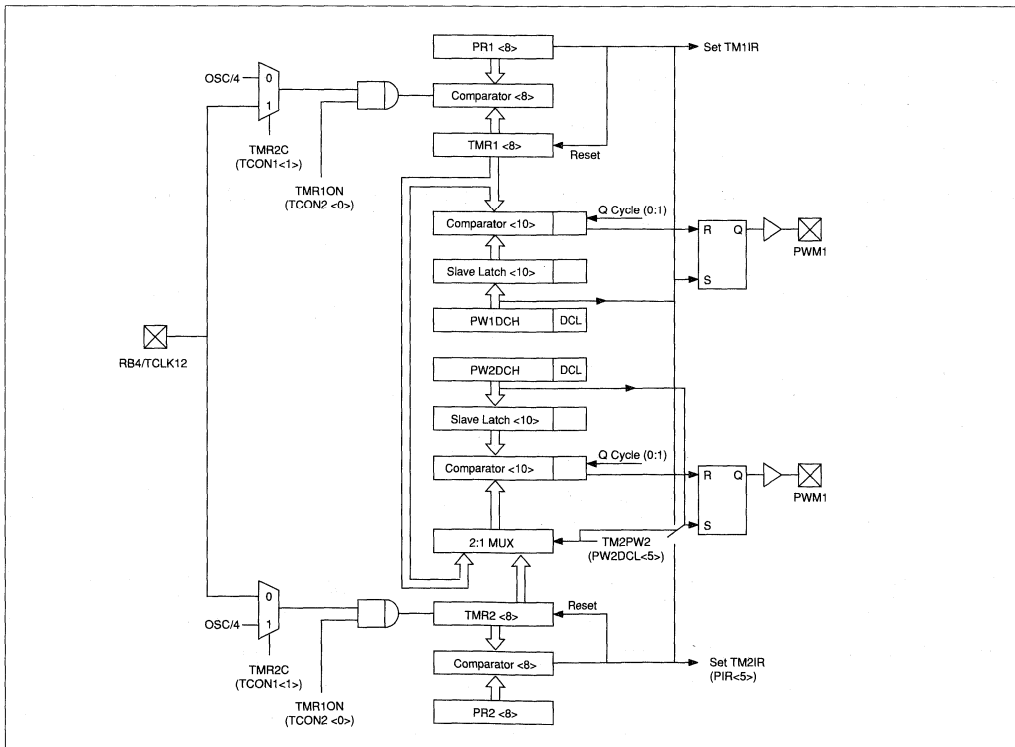
The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application report provides examples which illustrate the uses of Pulse Width Modulation (PWM) using the PIC17C42 Timer1 or Timer2 modules. These examples may be modified to suit the specific needs of your application.

The listing file for the Variable Period / Variable Duty Cycle example can be found in Appendix A. The source files can be found on the Microchip BBS. On directions on how to access the Microchip BBS please refer to DS30128, which can also be found in the Microchip Embedded Control Handbook (Literature Number DS00092).

This Application Note describes the operation of the PWM. They include the following topics:

1. Simple PWM Operation
2. Variable Period / Variable Duty Cycle PWM
3. External Clock for Timer Timebase
(ramifications/issues)

FIGURE 1 - TIMER1 AND TIMER2 BLOCK DIAGRAM WITH PWM MODE



PWM Operation

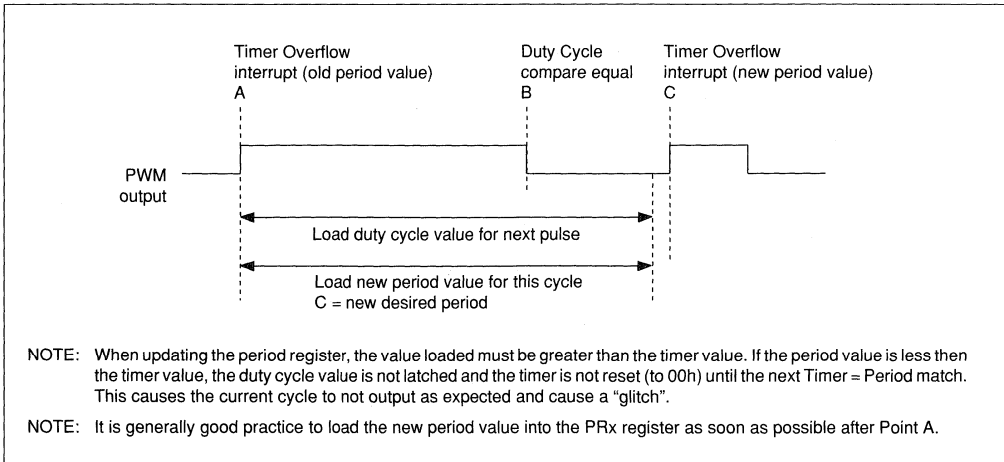
The control registers that are utilized by Timer1 and Timer2 are show in Table 1. Shaded Boxes are control bits that are not used by the Timer1 nor Timer2 module.

TABLE 1 - REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE

Name	BANK	ADDR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
INTSTA		0x06	PEIR	RTXIR	RTCIR	INTIR	PEIE	RTXIE	RTCIE	INTIE
CPUSTA		0x07	-	-	STKAV	GLINTD	TO	PD	-	-
TMR1	2	0x10	Timer1 Register							
TMR2	2	0x11	Timer2 Register							
PR1	2	0x14	Timer1 Period Register							
PR2	2	0x15	Timer2 Period Register							
PIR	1	0x16	IRB	TM3IR	TM2IR	TM1IR	CA2IR	CA1IR	TBMT	RBFL
PIE	1	0x17	IEB	TM3IE	TM2IE	TM1IE	CA2IE	CA1IE	TXIE	RCIE
PW1DCL	3	0x10	PWM1 Duty Cycle Low Register							
PW2DCL	3	0x11	PWM2 Duty Cycle Low Register							
PW1DCH	3	0x12	PWM1 Duty Cycle High Register							
PW2DCH	3	0x13	PWM2 Duty Cycle High Register							
TCON1	3	0x16	CA2ED1	CA2ED0	CA1ED1	CA1ED0	16/8	TMR3C	TMR2C	TMR1C
TCON2	3	0x17	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/PR3	TMR3ON	TMR2ON	TMR1ON

Care must be taken when loading values into the PWM registers. These registers are the duty cycle registers (PWxDCH:PWxDCL) and the period register (PRx). Figure 2 shows the proper update timing of these values.

FIGURE 2 - TIMING FOR UPDATING THE DUTY CYCLE REGISTERS AND PERIOD REGISTER



SIMPLE PWM OPERATION

Simple PWM operation is where the period of the PWM output remains constant, and only the duty cycle is modified. The PWM can operate in either of two modes:

- Hi-resolution mode—the PWxDCL register is modified
- Standard resolution mode - the PWxDCL register is not modified

When operating in the high-resolution mode, only the PWxDCH register is ever modified. Since this takes only a single cycle, this can be done at any time. Also since the period is remaining constant this may be done without any PWM interrupt software overhead.

When operating in the high-resolution mode both the PWxDCH:PWxDCL register pair is modified. Since this is a multicycle update, care needs to be taken that the "new" PWM duty cycle value is not latched until the update is complete. If the duty cycle is latched before this update is complete, the duty cycle will display a "glitch". If the PWxDCH is written first, the maximum error is 3 Q-cycles (187.5 ns @ 16 MHz). If the PWxDCL is written first, the maximum error is also 3 Q-cycles (187.5 ns @ 16 MHz), with the PWxDCH delayed by one PWM period. This may be acceptable for some applications. If this is not acceptable for your application then a subroutine can be written to ensure that these duty cycle writes are not done when the timer will equal the period. One implementation of this subroutine (PWM_UD) is used in the Variable Period / Variable Duty Cycle PWM example. This is discussed in the following section, with the listing in Appendix A.

Additional code examples can be found in application note AN539 in the Embedded Control Handbook.

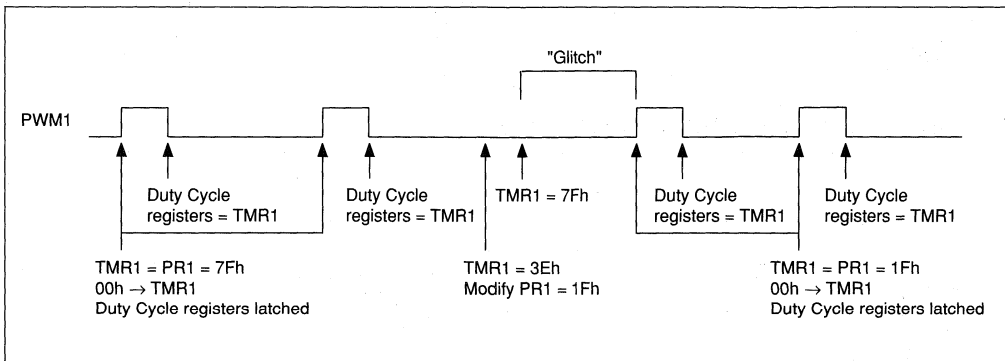
VARIABLE PERIOD / VARIABLE DUTY CYCLE PWM

In a variable period / variable duty cycle PWM both the duty cycle of the PWM as well as the frequency (period) of the PWM are modified.

The 17C42's hardware double buffers the duty cycle registers, but the period registers are not double buffered. What this means is that you can modify the duty cycle registers, but the value will only be latched when the timer register equals period register. Since the period register is not buffered, as the period register is modified this becomes the "new" period. This means that care must be taken when modifying the period register. The most common problem would be to modify the period register resulting in a "glitch" to occur. This "glitch" occurs when the period register is modified with a value that is less than the present timer value. The timer does not have a match with the old period value, and continues to count until the timer register equals period register.

Figure 3, shows an example where the period (PR1) register = 7Fh. Then the period is modified to a smaller value (PR1 = 1Fh) without checking that the value in Timer1 (TMR1) register = 3Eh. Since the new period (PR1) value is less than the present timer (TMR1) value, a glitch has occurred.

FIGURE 3 - MODIFYING PERIOD REGISTER "GLITCH"



PWM Operation

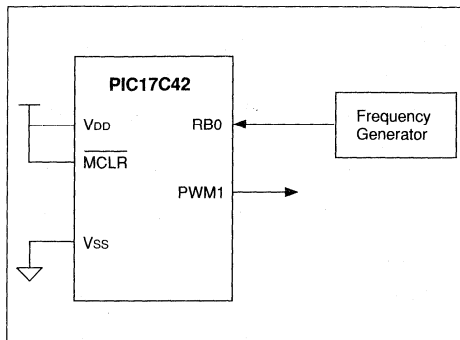
Care must be taken when writing a 10-bit duty cycle value. Since this requires two register writes, the Timer equals period could occur between these two writes, which would give a duty cycle that was not as expected. The cases are as follows:

1. If the duty cycle low (DCL) register is written, and then the Timer equals period. The old DCH register and the new DCL register becomes the duty cycle.
2. If the duty cycle high (DCH) register is written, and then the Timer equals period. The new DCH register and the old DCL register becomes the duty cycle

At the following occurrence of the timer equaling the period, the second register written would be updated. The subroutine PWM_UD (Appendix A) ensures that these duty cycle writes are not done when the timer will equal the period.

A software example of a variable period / variable duty cycle is shown in appendix A. In this example the period is double buffered in software, and the new period value is loaded in the timer overflow interrupt service routine. When the new duty cycle needs to be loaded. The device connections are shown in Figure 4. This program has two PWM settings (period / duty cycle combinations) that are switched between depending on the level on pin RB0. A frequency generator was used to give a low frequency signal on the RB0 pin. Figure 5 shows an example of the input and output waveforms.

FIGURE 4 - APPLICATION HARDWARE SETUP

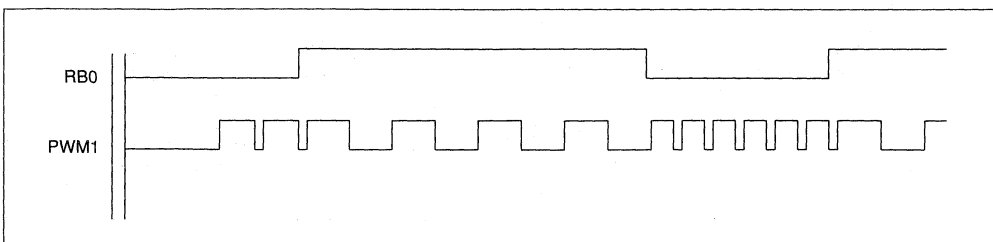


The program listing in appendix A implements this example, Figure 8 is the flowchart of the program. This example may be modified to suit the particular needs of your application. The Table 3 is a summary of the requirements for this program (@ 16 MHz):

TABLE 3 - PROGRAM REQUIREMENTS

Code Size:	52 Words
RAM used:	11 Bytes
Interrupt Service Routine time	3.0 usec
Subroutine time	4.5 usec 6.0 usec
Maximum PWM frequency:	200 KHz
PWM Accuracy:	62.5 nsec

FIGURE 5: EXAMPLE APPLICATION WAVEFORMS.



EXTERNAL CLOCK FOR TIMER TIMEBASE

The counters used for the time base of the PWM outputs can be software selected to operate from an external clock source. This allows a lower frequency PWM to be achieved. Doing this brings up new issues that must be understood for the application.

One of these issues is clock synchronization. All external clocks must be synchronized to the internal operating speed of the microcontroller, as shown in figure 9. When this synchronization occurs the PWM output is not truly operating from the external clock, but actually the internal synchronized clock. This leads to a "jitter" of the output to the clock. This jitter is caused from the delta time between the external clock and the synchronized clock not being constant. The synchronization errors are:

$$\text{Duty cycle error} = \pm T_{cy}$$

$$\text{Period error} = \pm T_{cy}$$

If you needed to run the PWM at a low frequency, and also want to reduce the "jitter" from the use of an external asynchronous clock, a PWM output could be used as the synchronous clock source. When the clock is synchronized to the device the clock error is always constant, so there is no jitter. Figure 7 shows this example.

FIGURE 7 - PWM OUTPUT TO GENERATE A SYNCHRONOUS CLOCK

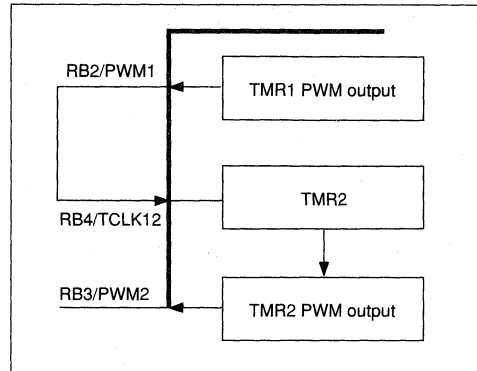
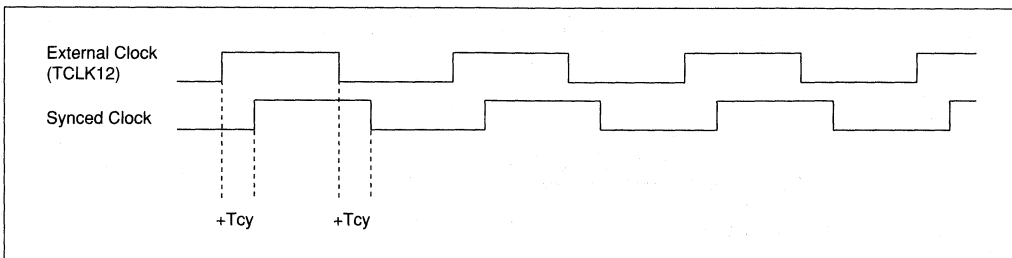


FIGURE 6 - EXTERNAL CLOCK SYNCHRONIZATION



PWM Operation

Another use is where precise timing of updates need to be done, but not at the frequency of the PWM output. In this discussion, TMR1 is used as the time-base of a constant frequency PWM output. TMR1 uses the internal clock of the device and TMR2 uses the external clock input. TMR2 will get the clock input from the PWM2 output.

The PWM output is a constant frequency variable duty cycle output. The PW1DCH:PW1DCL register pair contain the variable duty cycle value of PWM1 output. The PW2DCH:PW2DCL register pair is set for a fixed duty cycle (50%) for the PWM2 output.

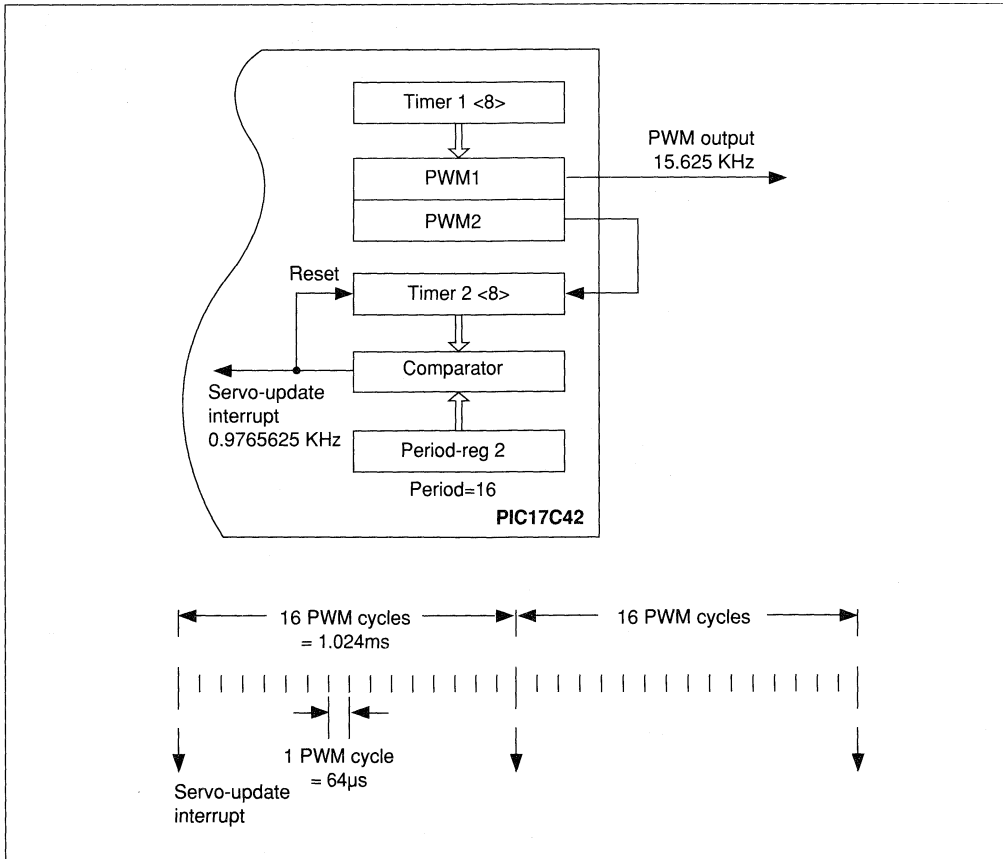
The PWM outputs could be programmed to have a frequency of 20 KHz, so to reduce audible noise. The PWM2 signal is connected to the RB4/TCLK12, as shown in Figure 11. The PR2 register could be loaded with 14h (20), to give a interrupt every 1 KHz. This interrupt can then trigger tasks, such as updating the duty cycle of PWM1. This is useful in motor control as well as other applications where the update rate is less than the PWM frequency.

CONCLUSION

The PIC17C42s PWM features offer a high performance solution at a lower system cost than previously available. The versatility of PWM's make the PIC17C42 ideal for motor control applications (see AN532) and many industrial control applications.

*Author: Mark Palmer
Logic Products Division*

FIGURE 11 - SAMPLING SCHEME



PWM Operation

APPENDIX A: LISTING FILE

MPASM B0.54

PAGE 1

```
LIST P=17C42, T=ON, C=120, N=0

; This is the basic outline for a program that generates a
; variable PWM output. The PWM's period and duty cycle can
; be varied. The new period (NEW_PR1) and the new duty cycle
; (NEW_DC1 and NEW_DC1Q) are loaded by the user program.
; The peripheral interrupt routine loads the new period value
; (frequency) into the PR1 register. A subroutine (PWM_UD)
; is also used to ensure that the 10-bit duty cycle registers
; are updated in the same PWM cycle, i.e. the timer match does not
; occur between two duty cycle register writes.
;
; The duty cycle value gets latched on the overflow (Period match)
; of the timer. The period value gets modified as soon as the period
; register is changed. Therefore care must be taken in updating
; the period register. In cases where the period value is modified
; to a smaller value, we must ensure that the Timer counter is less
; then this value when the period register is updated (TMR1 < new PR1).
; If TMR1 is greater then PR1, the counter will count to FFh, rollover
; to 00h, and only cause the overflow interrupt when it then reaches
; the period value. This would give a wrong PWM output.
;
; In this example the event which cause the PWM to be updated
; is an asynchronous event. A low frequency signal was placed on
; port pin RBO.
; For a high level the PWM registers are updated as follows:
; PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
; For a low level the PWM registers are updated as follows:
; PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h
;
; Do the EQUate table
;
0020 NEW_DC1 EQU 0x20 ; New PWM1 duty cycle value
0021 NEW_DC1Q EQU 0x21 ;
0022 NEW_PR1 EQU 0x22 ; New PWM1 period value
0025 PWM_WIN EQU 0x25 ; Register for the PWM window cycle
count
0026 CALC_PR EQU 0x26 ; Calculated period value
0027 FLAG_REG EQU 0x27 ; Register for flag bits
;
001A DC1H EQU 0x1A ; PWM registers for high time
001B DC1QH EQU 0x1B
001C PR1H EQU 0x1C
;
001D DC1L EQU 0x1D ; PWM registers for low time
001E DC1QL EQU 0x1E
001F PR1L EQU 0x1F
;
;
07FF END_OF_PROG_MEM EQU 0x07FF
;
0004 ALUSTA EQU 0x04
0006 CPUSTA EQU 0x06
0007 INTSTA EQU 0x07
000A W EQU 0x0A
;
0011 DDRB EQU 0x11 ; Bank 0
0012 PORTB EQU 0x12
;
0016 PIR EQU 0x16 ; Bank 1
0017 PIE EQU 0x17
```

PWM Operation

```

;
0010      TMR1      EQU      0x10      ; Bank 2
0011      TMR2      EQU      0x11
0012      TMR3l     EQU      0x12
0013      TMR3h     EQU      0x13
0014      PR1       EQU      0x14
0015      PR2       EQU      0x15
0016      PR3L      EQU      0x16
0017      PR3h      EQU      0x17

;
0010      PW1DCL     EQU      0x10      ; Bank 3
0011      PW2DCL     EQU      0x11
0012      PW1DCH     EQU      0x12
0013      PW2DCH     EQU      0x13
0016      TCON1      EQU      0x16
0017      TCON2      EQU      0x17

0000 C02B      ORG      0x0000      ; Origin for the RESET vector
; On reset, go to the start of
; the program
; Origin for the external RA0/INT
; interrupt vector
0008 C07C      GOTO     EXT_INT      ; Goto the ext. interrupt
; on RA0/INT routine
; Origin for the RTCC
; overflow interrupt vector
0010 C07D      GOTO     RTCCINT      ; Goto the RTCC overflow interrupt
; routine
; Origin for the external
; RA1/RT interrupt vector
0018 C07E      GOTO     RT_INT       ; Goto the ext. interrupt on
; RA1/RT routine
; Origin for the interrupt vector
; of any enabled peripheral

;
; The interrupt routine for any peripheral interrupt, This routine
; only deals with Timer1 interrupt.
;
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routine)
;
; case1 - only T1 overflow          = 12 cycles
; case2 - Other                    = Infinite Loop
;
;
0020 B801      PER_INT      MOVLB    1          ; Select register Bank 1
0021 9416      BTFS      PIR,4          ; Did Timer1 overflow?
0022 C022      ERROR       GOTO     ERROR      ; Not a Timer1 overflow.
; No other interrupts should
; be enabled, so error.

;
; Once the enabled Timer1 overflow occurs, the period register
; is loaded. This PWM waveform will remain until the PWM duty
; cycle and / or period is updated. Until such update, there is no
; S/W overhead from T1 interrupts (T1 interrupts can be disabled).
;
; NOTE: If PW1DCH >= PR1, then the duty cycle of this PWM output
; is 100%.
;
; NOTE: The new Period register (PR1) value, must always be greater
; than the value in the Timer1 register (TMR1). If a PR1 value
; is loaded that is less than the TMR1 value, the timer will
; continue to count until it reaches the PR1 value. I.E. TMR1
; will overflow at FFh and the count to the new PR1 value.
; Minimum PR1 value is 0Ah, due to time to load new values and
; execute the peripheral interrupt service routine.
;
0023 8C16      T1OVFL      BCF      PIR,4          ; Clear Overflow interrupt flag
0024 B802      MOVLB      2          ; Bank2

```

PWM Operation

```

0025 7422          MOVFP  NEW_PRI,PR1      ; Load this period value
0026 B801          MOVLB  1                ; Bank 0
0027 8C17          BCF    PIE, 4          ; Disable T1 interrupt
                                           ; (until transition on PORTB0)
0028 B800          MOVLB  0                ; Bank 0
0029 3F12          BTG    PORTB, 7        ; Transition PortB 7 pin (H->L,
002A 0005          RETFIE                  ; Return from Interrupt

;
; This is the start of the program.
;
002B 8406          START      BSF    CPUSTA,4      ; Disable ALL interrupts via the
                                           ; Global Interrupt Disable
                                           ; (GLINTD) bit.
                                           ;
MAIN
002C B803          MOVLB  3                ; Place Main program here
002D 2817          CLRFB  TCON2,0          ; Select register Bank 3
002E B070          MOVLW  0x070           ; Stop the timers, Single Capture
002F 0116          MOVWF  TCON1           ; Initialize TCON1 so that
                                           ; T1 (8-bit), T2 (8-bit),
                                           ; and T3 run off the internal
                                           ; system clock. Timer3 uses
                                           ; period register
0030 B00D          MOVLW  0x0D           ; Load the PWM window cycle value
0031 0125          MOVWF  PWM_WIN         ;

;
0032 B800          MOVLB  0                ; Select register Bank 0
0033 2B11          SETFB  DDRB, 1         ; Port B is an input
0034 2912          CLRFB  PORTB, 1        ; Set output values to 0 (for
0035 8F11          BCF    DDRB, 7        ; PORTB7 is an output. Used to
0036 2927          CLRFB  FLAG_REG, 1     ; Clear the Flag registers

;
; Load registers with the PWM values that we will switch between. One set
; for the time PORTB0 is high and another set for when low.
;
; For a high level the PWM registers are updated as follows:
; PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
; At 16Mhz this gives a period of 31.75 us, and a duty cycle of 16.625 us
; For a low level the PWM registers are updated as follows:
; PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h
; At 16Mhz this gives a period of 7.75 us, and a duty cycle of 6.00 us
;

0037 B803          MOVLB  3                ; Bank 3
0038 B03F          MOVLW  0x3F           ; The Duty Cycle initial value is
0039 4A1A          MOVFP  W, DC1H         ; 50% of the initial period
003A B040          MOVLW  0x40           ;
003B 4A1B          MOVFP  W, DC1QH       ; Duty Cycle low = 01
003C B007          MOVLW  0x07           ; The Duty Cycle initial value is
003D 4A1D          MOVFP  W, DC1L        ; 25% of the initial period
003E B080          MOVLW  0x80           ;
003F 4A1E          MOVFP  W, DC1QL       ; Duty Cycle low = 10

;
0040 B802          MOVLB  2                ; Bank 2
0041 B07F          MOVLW  0x7F           ;
0042 4A1C          MOVFP  W, PR1H        ; The initial period value is 50%
                                           ; of full scale (for High)
0043 B01F          MOVLW  0x1F           ;
0044 4A1F          MOVFP  W, PR1L        ; The initial period value is
                                           ; of full scale (for Low)

;
; Default PWM values should be set, and the timer should be started
; and the interrupts enabled.
;
0045 B0F0          MOVLW  0xF0           ; Load the Period register
0046 0114          MOVWF  PR1            ;
0047 B803          MOVLB  3                ; Select register Bank 3
0048 B0C0          MOVLW  0xC0           ; Load the T1 duty cycle register
0049 0112          MOVWF  PW1DCH         ;

```

PWM Operation

```

004A 0110      MOVWF  PW1DCL      ; effectively loaded with 0
004B B031      MOVLW  0x31      ;** Enable PWM1 and PWM2 outputs
004C 0117      MOVWF  TCON2      ;** and turn on Timer1.
004D 8307      BSF    INTSTA,3    ; Turn on Peripheral Interrupts
004E B801      MOVLB  1      ; Select register Bank 1
004F B010      MOVLW  0x10      ; Enable Timer1 overflow
0050 0117      MOVWF  PIE      ; Interrupts (when GLINTD = 0)
0051 8C06      BCF    CPUSTA,4    ; Enable ALL interrupts
0052 B800      MOVLB  0      ; Bank 0

;
; Only need to update PWM values on the first occurrence of a new level on
; Else loop waiting for level to change.
;
0053 8827      HIGH1ST  BCF    FLAG_REG, 0      ; First time in loop (this
cycle) = True
0054 9012      HIGHCYC  BTFSS  PORTB, 0      ; Is PortB0 low
0055 C05F      GOTO    LOW1ST      ; PORTB0 = L
0056 9827      BTFSC  FLAG_REG, 0      ; Is this the First High time
(this cycle)?
0057 C054      GOTO    HIGHCYC      ; Loop looking for low signal on
PortB0
0058 8027      BSF    FLAG_REG, 0      ; Set First time in loop (this
cycle) = False

;
; Here is where we update the PWM values (period and Duty cycle) for high
;
0059 B803      MOVLB  3      ; Bank 3
005A 5A20      MOVWF  DC1H, NEW_DC1      ;
005B 5B21      MOVWF  DC1QH, NEW_DC1Q      ;
005C 5C22      MOVWF  PR1H, NEW_PR1      ;
005D E06B      CALL   PWM1_UD      ;
005E C054      GOTO    HIGHCYC      ; Loop looking for low signal on

;
;
005F 8827      LOW1ST  BCF    FLAG_REG, 0      ; First time in loop (this
0060 9812      LOWCYC  BTFSC  PORTB, 0      ; Is PortB0 high
0061 C053      GOTO    HIGH1ST      ; PORTB0 = H
0062 9827      BTFSC  FLAG_REG, 0      ; Is this the First Low time
0063 C060      GOTO    LOWCYC      ; Loop looking for high signal
on PortB0
0064 8027      BSF    FLAG_REG, 0      ; First time in loop (this

;
; Here is where we update the PWM values (period and Duty cycle) for low
;
0065 B803      MOVLB  3      ; Bank 3
0066 5D20      MOVWF  DC1L, NEW_DC1      ;
0067 5E21      MOVWF  DC1QL, NEW_DC1Q      ;
0068 5F22      MOVWF  PR1L, NEW_PR1      ;
0069 E06B      CALL   PWM1_UD      ;
006A C060      GOTO    LOWCYC      ; Loop looking for high signal

;
; This code segment ensure that all PWM values (period and duty cycle)
; are updated at the same time. This is done by ensuring that the Timer
; is at least PWM_WIN (0Dh) cycles before the PR1 value (PR1 - PWM_WIN >
; If not a "glitch" could occur in the PWM waveform. When only the 1st duty
; cycle register is latched for this PWM cycle, and the following PWM period
; will latch the 2nd duty cycle register.
;
006B 8406      PWM1_UD  BSF    CPUSTA, 4      ; Disable Global Interrupts
006C B802      MOVLB  2      ; Bank 2
006D 6A10      MOVWF  TMR1, W      ; Load W reg. with Timer1 value
006E 0414      SUBWF  PR1, 0      ; PR1 - TMR1 -> W reg.
006F 3025      CPFSLT  PWM_WIN      ; Check if Timer1 is about to
0070 C06B      GOTO    PWM1_UD      ; Overflow would have occurred
; PWM updates, Delay a few
0071 B803      MOVLB  3      ; Bank 3

```

PWM Operation

```
0072 6A20      MOVFP  NEW_DC1, W      ; Your New PWM MSB
0073 0112      MOVWF  PW1DCH          ; Loaded in duty cycle buffer
0074 6A21      MOVFP  NEW_DC1Q, W   ; Your New PWM LSB
0075 0110      MOVWF  PW1DCL          ; Loaded in duty cycle buffer
0076 B801      MOVLB  1              ; Back to Bank 1
0077 8C16      BCF    PIR, 4          ; Clear T1 Overflow interrupt
0078 8417      BSF    PIE, 4          ; Enable T1 int
0079 8C06      BCF    CPUSTA, 4       ; Enable Global Interrupts
007A B800      MOVLB  0              ; Bank 0
007B 0002      RETURN          ;** this does not need to be
                                ;** as a subroutine.
;
; Other Interrupt routines. (Not utilized in this example)
;
007C 0005      EXT_INT  RETFIE          ; RAO/INT interrupt routine
                                ; (NOT used in this program)
007D 0005      RTCCINT  RETFIE          ; RTCC overflow interrupt routine
                                ; (NOT used in this program)
007E 0005      RT_INT   RETFIE          ; RAL/RT interrupt routine
                                ; (NOT used in this program)
;
007F C02B      SRESET   GOTO  START     ; If program became lost, goto
                                ; START and reinitialize.
;
;
; When the executed address is NOT in the program range, the
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.
;
                                ORG    END_OF_PROG_MEM ;
07FF C07F      GOTO    SRESET          ; The program has lost it's mind,
                                ; do a system reset
                                END

Errors : 0
Warnings : 0
```

PWM Operation

NOTES:

Using the Capture Module

INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application report provides examples which illustrate the uses of input capture using the PIC17C42 Timer3 module. These examples may be modified to suit the specific needs of an application.

This Application Note has 4 examples that utilize the Timer3 input capture. They are:

1. Frequency Counter (Period Measurement)
2. Frequency Counter (Period Measurement) using a Free Running Timer
3. Pulse Width Measurement
4. Frequency Counter (Period Measurement) with Input Prescaler

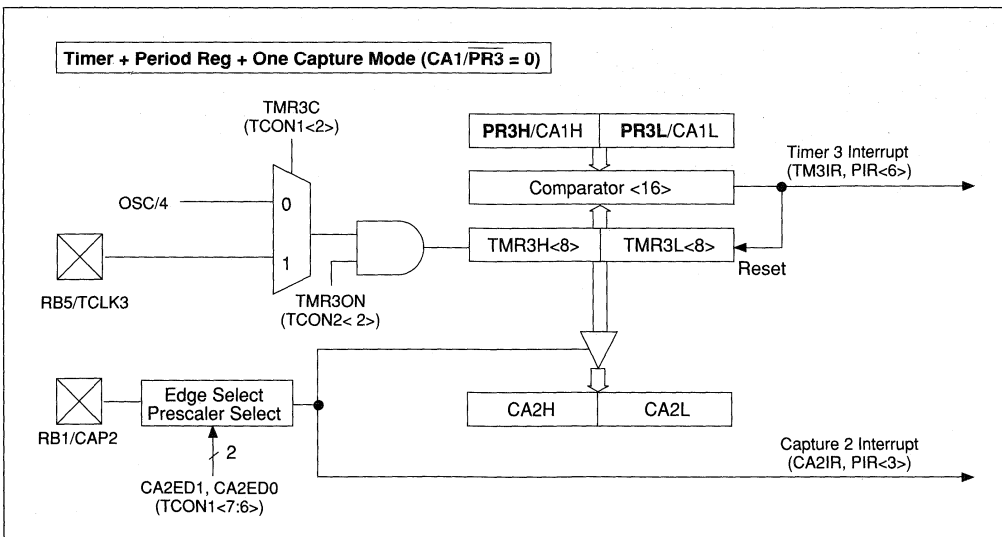
TIMER3 DESCRIPTION

Timer3 is a 16-bit counter that has two modes of operation which can be software selected. The CA1/PR3 bit (TCON2<3>) selects the mode of operation. The two modes are:

1. Timer3 with Period Register and Single Capture Register (see Figure 1)
 2. Timer3 and Dual Capture Registers (see Figure 2)
- Timer3 is the time-base for capture operations.

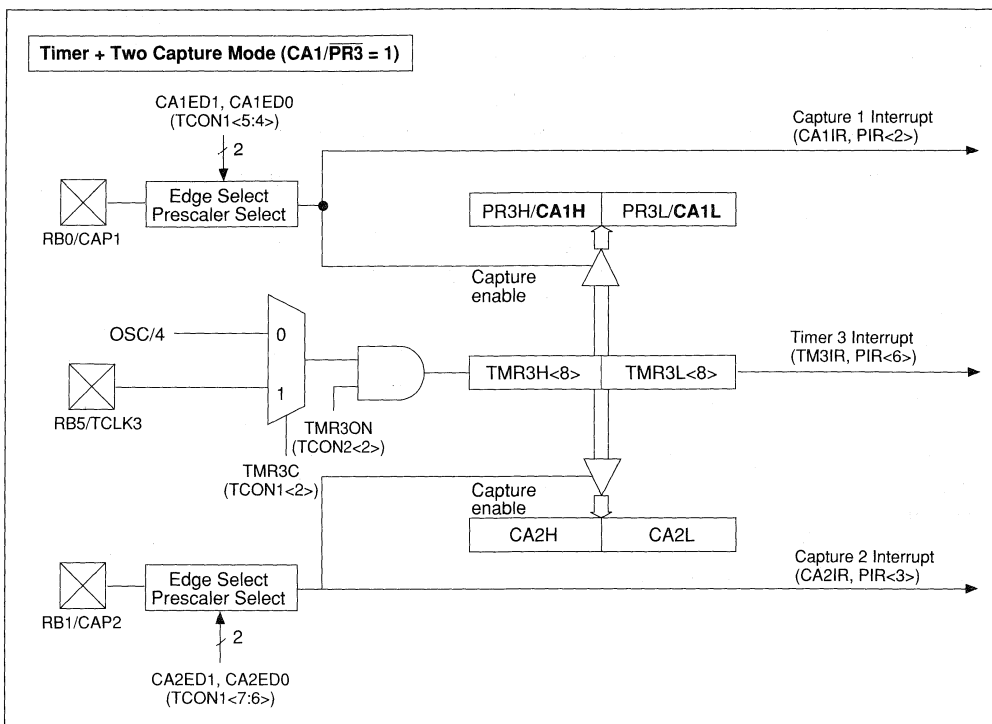


FIGURE 1 - TIMER3 WITH PERIOD REGISTER AND SINGLE CAPTURE REGISTER



Capture Module

FIGURE 2 - TIMER3 AND DUAL CAPTURE REGISTERS



The period register allows the time base of Timer3 to be something other than the 2^{16} counter overflow value, which corresponds to FFFFh (65536) cycles. This is accomplished by loading the desired period value into the PR3H/CA1H:PR3L/CA1L register pair. The overflow time can be calculated by this equation:

$$T_{\text{ofl}} = T_{\text{clk}} * (\text{value in PR3H/CA1H:PR3L/CA1L register pair} + 1).$$

Where Tclk is either the internal system clock (Tcy) or the external clock cycle time. Table 1 shows time-out period for different period values at different frequencies. The values in the register are the closest approximation for the period value. All examples in this application report uses a Timer3 overflow value of FFFFh.

TABLE 1 - TIMER3 OVERFLOW TIMES

Overflow Time	Period Register					
	@ 16 MHz (250 ns)	@ 10 MHz (400 ns)	@ 8 MHz (500 ns)	@ 5 MHz (800 μs)	@ 2 MHz (2.0 μs)	@ 32 KHz (125 μs)
8.192 S	N.A.	N.A.	N.A.	N.A.	N.A.	0xFFFF
131.072 mS	N.A.	N.A.	N.A.	N.A.	0xFFFF	0x0418
52.428 mS	N.A.	N.A.	N.A.	0xFFFF	0x6666	0x01A3
32.7675 mS	N.A.	N.A.	0xFFFF	0x9FFF	0x3FFF	0x0106
26.214 mS	N.A.	0xFFFF	0xCE20	0x80D4	0x3388	0x00D3
16.384 mS	0xFFFF	0xA000	0x8000	0x5000	0x2000	0x0083
10.0 mS	0x9C40	0x61A8	0x4E20	0x30D4	0x1388	0x0050
4.0 mS	0x3E80	0x2710	0x1F40	0x1388	0x07D0	0x0020
1.0 mS	0x0FA0	0x09C4	0x07D0	0x04E2	0x01F4	0x0008
600 μS	0x0960	0x05DC	0x04B0	0x02EE	0x012C	0x0005
100 μS	0x0190	0x00FA	0x00C8	0x007D	0x0032	N.A.

The uses of an input capture are all for time based measurements. These include:

- Frequency measurement
- Duty cycle and pulse width measurements

The PIC17C42 has two pins (RB0/CAP1 and RB1/CAP2) which can be used for capturing the Timer3 value, when a specified edge occurs. The input capture can be specified to occur on one of the following four events:

- Falling Edge
- Rising Edge
- 4th Rising Edge
- 16th Rising Edge

These are specified, for both capture pins, by the register TCON1<7:4>.

This flexibility allows an interface without the need of additional hardware to change polarity or specify an input prescaler.

Capture Module

The control registers that are utilized for by Timer3 are shown in Table 2. Shaded Boxes are control bits that are not used by the Timer3 module, the Peripheral Interrupt enable and flag bits, and the Global Interrupt enable bit.

TABLE 2 - REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE:

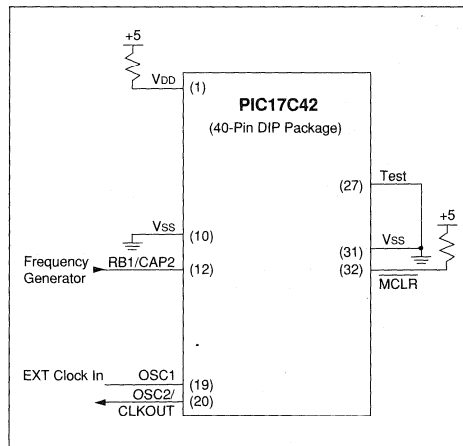
Name	BANK	ADDR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
INTSTA	All	6h	PEIR	RTXIR	RTCIR	INTIR	PEIE	RTXIE	RTCIE	INTIE
CPUSTA	All	7h	-	-	STKAV	GLINTD	TO	PD	-	-
TMR3L	2	12h	Timer3 LSB Register							
TMR3H	2	13h	Timer3 MSB Register							
CA2L	3	14h	Timer3 Capture2 LSB Register							
CA2H	3	15h	Timer3 Capture2 MSB Register							
PIR	1	16h	IRB	TM3IR	TM2IR	TM1IR	CA2IR	CA1IR	TBMT	RBFL
PIE	1	17h	IEB	TM3IE	TM2IE	TM1IE	CA2IE	CA1IE	TXIE	RCIE
PR3L/ CA1L	2	16h	Timer3 - Period MSB Register/Capture1 MSB Register							
PR3H/ CA1H	2	17h	Timer3 - Period MSB Register/ Capture1 MSB Register							
TCON1	3	16h	CA2ED1	CA2ED0	CA1ED1	CA1ED0	16/8	TMR3C	TMR2C	TMR1C
TCON2	3	17h	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/PR3	TMR3ON	TMR2ON	TMR1ON

This Application Note has 4 examples that utilize the Timer3 input capture. They are:

1. Frequency Counter (Period Measurement)
2. Frequency Counter (Period Measurement) using a Free Running Timer
3. Pulse Width Measurement
4. Frequency Counter (Period Measurement) with Input Prescaler

All these examples can be run from a simple setup, this is shown in Figure 3.

FIGURE 3 - APPLICATION HARDWARE SETUP



A discussion of each application with the operation of the software and application issues. The source listings for these are in appendices A-D.

PERIOD MEASUREMENT (FREQUENCY COUNTER)

Period measurement is simply done by setting the counter to 0000h, then starting the counter on the 1st rising edge. On the following rising edge, the capture 2 register is loaded with the Timer3 value and the Timer3 (TMR3) register is cleared. If the period is greater than the overflow rate of Timer3, the timer overflows, causing an interrupt. With a TMR3 overflow, an interrupt occurs and the overflow counter may need to be incremented. The overflow counter should be incremented if:

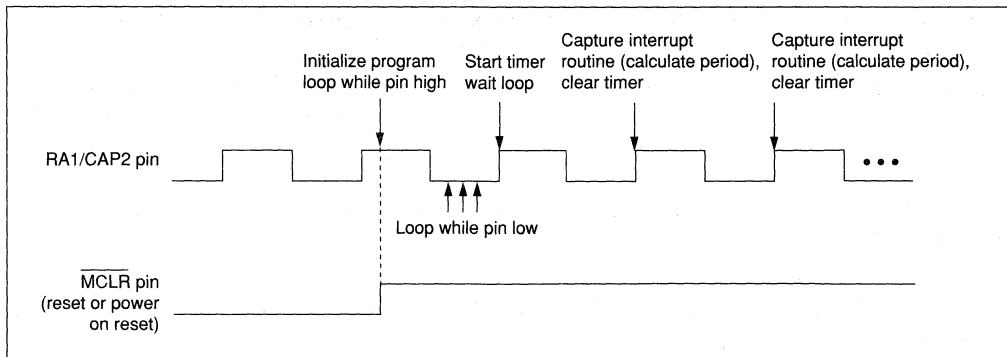
1. The TMR3 overflow is the only interrupt source.
2. Both the TMR3 overflow and capture2 interrupts occurred at near the same time, but the TMR3 overflow occurred first (Most Significant Byte of the Capture2 register = 00h).

Once a capture2 has occurred, the capture registers are moved to data RAM, the capture2 interrupt flag is cleared and the TMR3 register is loaded with an offset value. This offset value is the number of cycles from the time the interrupt routine is entered to when the TMR3 register is reloaded. In this example a data RAM location is used as an overflow counter. This gives in effect a 24-bit timer. The software flow for this routine is shown in Figure 4.

The program listing in Appendix A implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	30 Words
RAM used:	4 Bytes
Maximum frequency that can be measured:	130 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (± 250 nsec)

FIGURE 4 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2



Capture Module

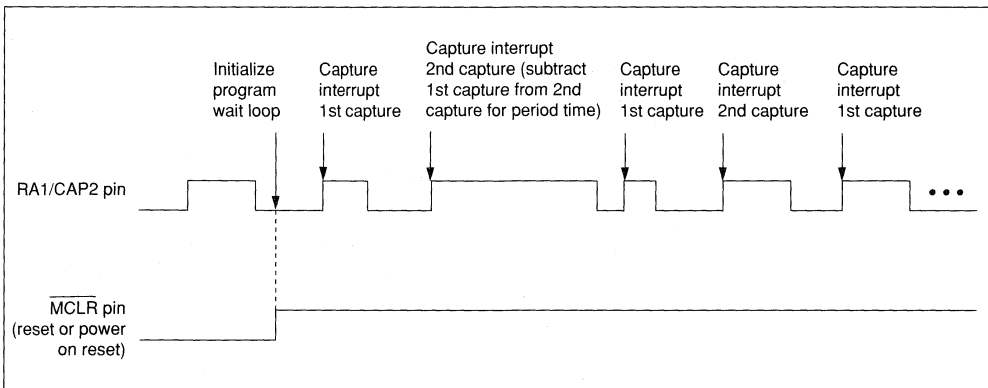
PERIOD MEASUREMENT (FREQUENCY COUNTER) USING A FREE RUNNING TIMER

In many applications the timer (TMR3) would need to be used for multiple tasks, and could not be reset (modified) by any one of these tasks. This is called a free running timer. To do period measurement in a free running timer application, the program needs to store each capture in a data RAM location pair (word). The 1st capture in data RAM locations input capture2A (IC2AH:IC2AL) and the 2nd capture in data RAM locations input capture2B (IC2BH:IC2BL). Once the two captures have occurred, the values in these two words is subtracted. Since this is a free running timer, the value in input capture2B may be less than the value in input capture2A. This is if the 1st capture occurs, then the Timer3 overflows, and then the 2nd capture occurs. So an overflow counter should only be incremented if the Timer3 overflow occurs after a capture1 but before the capture2 occurs. With the use of an overflow counter this becomes an effective 24-bit period counter. The software flow for this routine is shown in Figure 5.

The program listing in Appendix B implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	41 Words
RAM used:	76 Bytes
Maximum frequency that can be measured:	80 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (± 250 nsec)

FIGURE 5 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2



PULSE WIDTH MEASUREMENT USING A FREE RUNNING TIMER

Applications that require the measurement of a pulse width can also be easily handled. The PIC17C42 can be programmed to measure either the low or the high pulse. The software example in Appendix C measures the High pulse time. The program is initialized to capture on the rising edge of the RA1/CAP2 pin. After this event occurs, the capture mode is switched to the falling edge of the RA1/CAP2 pin. When the capture edge is modified (rising to falling, or falling to rising) a capture interrupt is generated. This "false" interrupt request must be cleared before leaving the interrupt service routine, or the program will immediately re-enter the interrupt service routine due to this "false" request. When the falling edge of the RA1/CAP2 pin occurs, the difference of the two capture values is calculated. The flow for this is shown in Figure 6.

Due to the software overhead of the Peripheral Interrupt routine the following are the limitations on the input signal on RA1/CAP2 pin. This does not include any software overhead that may be required in the main routine, or if additional peripheral interrupt features need to be included. This is shown in Table 3. If you assume that the input is a square wave (high time = low time), one needs to take the worst case time of the two minimum pulse times (11 μ S) times two, to determine the period. The maximum continuous input frequency would then be approximately 45.5 KHz. For a single pulse measurement, minimum pulse width is 4.5 μ s.

The program listing in Appendix C implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	51 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	71 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (\pm 250 nsec)

FIGURE 6 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2

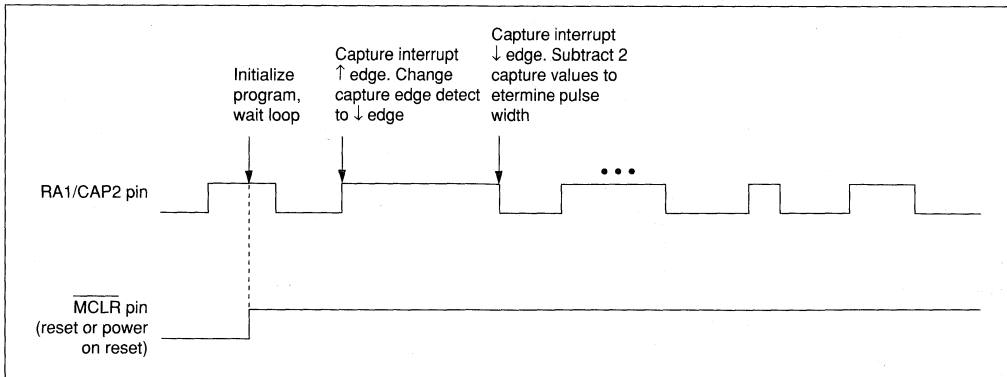


TABLE 3 - PERIPHERAL INTERRUPT ROUTINE

EVENT		# of Cycles	Time @ 16 MHz
1st CAPTURE	Capture1 Only	18	4.5 μ S
	Capture1 and Timer Overflow	30	7.5 μ S
2nd CAPTURE	Capture Only	35	8.75 μ S
	Capture and Timer Overflow	41	10.25 μ S
Minimum Pulse High	Capture1 and Timer Overflow + INT Latency	33	8.25 μ S
Minimum Pulse Low	Capture2 and Timer Overflow + INT Latency	44	11 μ S
Minimum Period (square wave)	2 * (Minimum Pulse Low)	88	22 μ S

Capture Module

PERIOD MEASUREMENT (FREE RUNNING TIMER) WITH A PRESCALER

Occasionally the application may require a prescaler on the input signal. This may be due to application requirements, such as:

- Require higher resolution measurement of the input signal
- Reduce interrupt service overhead
- The input frequency is higher than interrupt service routine

The software selectable prescaler of the PIC17C42 allows the designer to easily implement this in their system without the cost of additional hardware. Care must be taken in determining if this option is appropriate. For example, if the input frequency is not stable (excessive frequency change per period) then the prescaler will give a less accurate capture value than the individual measurements.

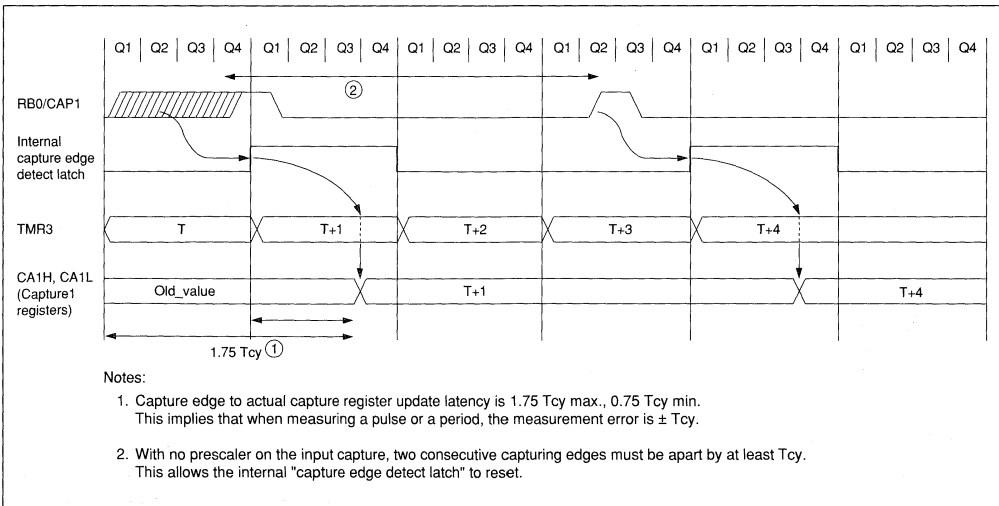
In cases where the resolution of the input frequency is important, the Prescaler can be used to reduce the input capture error. There are two components to input capture:

1. Resolution Error
2. Input Synchronization Error

These two errors add together to form the total capture error. Resolution error is dependent on the rate at which the timer is incremented, remember the timer may be based on an external clock (must be slower than T_{cy}). The input synchronization error is dependent on the system clock speed (T_{cy}), and will be less than T_{cy} .

It is easy to see that when a capture occurs the synchronization error (T_{sync}) can be up to $1 T_{cy}$ (see Figure 7). This error is constant regardless of the number of edges that occur before the capture is taken. So a capture on the 1st edge gives a synchronization error per sample up to T_{cy} . While a capture taken on the 16th edge gives a synchronization error per sample only up to $T_{cy} / 16$, by achieving a smaller percentage of error, the captured value becomes more accurate.

FIGURE 7 - SYNCHRONIZATION ERROR WITH NO CAPTURE PRESCALER



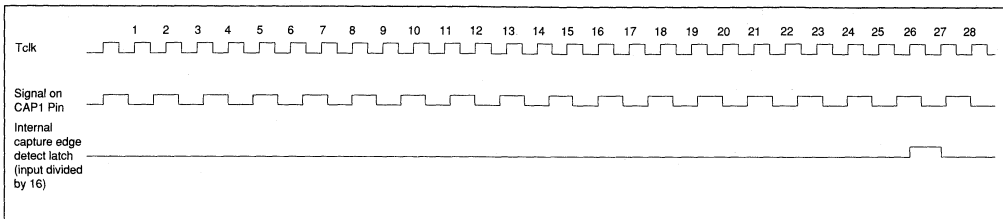
Capture Module

Another scenario is when the signal on the input capture pin is different than the system cycle time (T_{cy}), for example $1.6 T_{cy}$. If you tried to capture this you would have a capture value of 1. If you set the prescaler to actually capture on the 16th edge you would have $16 * 1.6 T_{cy} = 25.6 T_{cy}$, which would be latched on the 26th T_{cy} (see Figure 8). This $0.4 T_{cy}$ error is over 16 samples, which therefore gives an effective error/sample of $0.025 T_{cy}$.

The program listing in Appendix D implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	41 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	80 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	$\pm 0.4 T_{cy}$ (± 16.625 nsec)

FIGURE 8 - INPUT CAPTURE DIVIDED BY 16 PRESCALE EXAMPLE



Author: Mark Palmer
Logic Products Division

Capture Module

APPENDIX A - PERIOD MEASUREMENT EXAMPLE CODE

MPASM B0.54

PAGE 1

```
; This is the basic outline for a program that can determine the
; frequency of an input, via input capture. This routine uses an
; 8-bit register to count the times that timer3 overflowed. At the
; Max crystal frequency of 16 MHz, this gives an overflow time of
; (2**16) (256 + 1) (250 nS) > 4.21 sec or a frequency < 0.25 Hz. If
; measurement of longer time intervals is required, the overflow
; counter could be extended to 16 (or more) bits.
;
; Do the EQUate table
;
LIST      P=17C42, C=80, T=ON
0020      IC2OF          EQU      0x20          ; T3 overflow data
0021      IC2H          EQU      0x21          ; T3 IC2 MSB data r
0022      IC2L          EQU      0x22          ; T3 IC2 LSB data r
0023      T3OFLCNTR    EQU      0x23          ; Temperay T3 overf

;
07FF      END_OF_PROG_MEM EQU      0x07FF
;

0006      CPUSTA        EQU      0x06
0007      INTSTA        EQU      0x07
000A      W              EQU      0x0A
;
0012      PORTB         EQU      0x12          ; Bank 0
;
0016      PIR           EQU      0x16          ; Bank 1
0017      PIE           EQU      0x17
;
0012      TMR3L         EQU      0x12          ; Bank 2
0013      TMR3H         EQU      0x13
;
0014      CA2L          EQU      0x14          ; Bank 3
0015      CA2H          EQU      0x15
0016      TCON1         EQU      0x16
0017      TCON2         EQU      0x17
;

ORG      0x0000          ; Origin fo

0000 C028          GOTO      START          ; On reset,
; the program
ORG      0x0008          ; Origin fo
; interrupt vector
0008 C05E          GOTO      EXT_INT        ; Goto the
; on RA0/INT routine
ORG      0x0010          ; Origin fo
; overflow interrupt vector
0010 C05F          GOTO      RTCCINT        ; Goto the
; routine
ORG      0x0018          ; Origin fo
; RAl/RT interrupt vector
0018 C060          GOTO      RT_INT        ; Goto the
```


Capture Module

```
                                ; RAl/RT routine
                                ; Origin fo
                                ; of any enabled peripheral
                                ; Goto the
                                ; peripheral routine
                                ; Origin fo
                                ; program memory
                                ; Disable A
                                ; Global Interrupt Disable
                                ; (GLINTD) bit.
                                ;
                                ; Place Main program here
                                ; Select re
0020 C040          GOTO    PER_INT
                                ;
0028 8406          START   BSF    CPUSTA,4
                                ;
                                ;
                                ;
0029 B803          MAIN    MOVLB  3
                                ;
002A 2817          CLRFB  TCON2,0
                                ; Stop the
002B 8317          BSFB   TCON2,3
                                ; Dual in
002C B070          MOVLW  0x070
                                ; Initalize
002D 0116          MOVWF  TCON1
                                ; T1 (8-b
                                ;
                                ; and T3 run off the internal
                                ; system clock. Capture2
                                ; captures on the rising edge.
                                ;
                                ; Initialize Timer 3, so we can count clock cycles. For the
                                ; measurement Timer3 is not started exactly at the RAl/CAP2
                                ; transition. Therefore an offset is required due to softwa
                                ; overhead (3 cycle).
                                ;
002E B802          MOVLB  2
                                ; Select re
002F 280A          CLRFB  W,0
                                ; Clear the
0030 0113          MOVWF  TMR3H
                                ; Timer3 MS
0031 B003          MOVLW  0x03
                                ; Timer3 LS
0032 0112          MOVWF  TMR3L
                                ;
                                ;
                                ; We need to wait for an event to happen to start timer 3.
                                ; be when the RAl/CAP2 pin to does a Low to High transition
                                ; signifies the start of the first interval to measure. Thi
                                ; by looping while the RAl/CAP2 pin is high, and then loopi
                                ; it is low. When that 2nd loop is exited, there has been a
                                ; high transition and the timer should be started and inter
                                ; enabled.
                                ;
0033 B800          MOVLB  0
                                ; Select re
0034 9912          RAlHIGH BTFSB PORTB,1
                                ; Is the CA
0035 C034          GOTO    RAlHIGH
                                ; Loop here
0036 9112          RAlLOW  BTFSB PORTB,1
                                ; Is the CA
```

Capture Module

```
0037 C036          GOTO    RAILOW          ; Loop here
0038 B803          MOVLB   3              ; Select re
0039 8217          BSF     TCON2,2        ; Turn on t
003A 8307          BSF     INTSTA,3       ; Turn on P
003B B801          MOVLB   1              ; Select re
003C B048          MOVLW  0x48           ; Enable Ca
003D 0117          MOVWF  PIE            ; Interr
003E 8C06          WAIT          BCF     CPUSTA,4    ; Enable AL
003F C03E          GOTO    WAIT          ; Loop here
                                ; Interrupt
                                ;
                                ; The interrupt routine for any peripheral interrupt, This
                                ; only deals with Timer3 (T3) interrupts.
                                ;
0040 B801          PER_INT  MOVLB   1              ; Select re
0041 9E16          BTFSC  PIR,6            ; Did T3 ov
                                ; If not skip next
0042 C057          GOTO    T3OVFL          ; Inc overf
0043 9316          CK_CAP  BTFSS  PIR,3            ; Did the R
                                ; interrupt?
0044 0005          RETFIE                    ; No RB1/CA
                                ; Return from
                                ;
                                ; If Both a timer 3 overflow and CAP2 interrupt occurred nea
                                ; same time. Need to determine which one occurred first. Ther
                                ; two possibilities when the PIR register has both the time
                                ; overflow interrupt and input capture 2 interrupt bits set
                                ; 1. the input capture occurred and then timer 3 overfl
                                ; 2. timer 3 overflowed and then the input capture occ
                                ; in case 2. the MSB of the capture register would be 0h.
                                ;
0045 B803          CAPTURE  MOVLB   3              ; Select re
0046 5422          MOVPF  CA2L,IC2L          ; Move the
0047 5521          MOVPF  CA2H,IC2H          ; tempora
                                ; (to pervent
0048 B802          MOVLB   2              ; Select re
0049 B00A          MOVLW  0x000A          ; Reload th
004A 0112          MOVWF  TMR3L          ; 10, sin
                                ; to reach this
                                ; timer3 interrupt
```

Capture Module

```
004B 280A          CLRWF  W,0          ; Clear the
004C 0113          MOVWF  TMR3H        ; Timer3 MS
                                ;
004D B801          MOVLB  1          ; Select re
004E 3121          CPFSEQ  IC2H        ; If the hi
                                ; register = 0,
004F C051          GOTO   SDEC        ; (ie Capture
                                ; Skip the
0050 0723          DECF   T3OFLCNTR,1  ; Capture o
                                ; overflow, so
0051 6A23          SDEC          MOVFP  T3OFLCNTR,W  ; overflow counter.
                                ; Store the
0052 4A20          MOVFP  W,IC2OF      ; overflo
0053 2823          CLRWF  T3OFLCNTR,0  ; Clear the
                                ; counts how
                                ; overflows.
;
; Clear the Capture2 and T3 Overflow interrupt flags. T3 Ov
; should be cleared again since the overflow may have occur
; while we were in this (PER_INT) interrupt routine. This w
; be between the compare of the T3 overflow interrupt flag
; the resetting of the timer 3 counter, which is about 6 cy
;
0054 8B16          BCF   PIR,3          ; Clear Cap
0055 8E16          BCF   PIR,6          ; Clear Tim
                                ; flag
0056 0005          RETFIE         ; Return fr
                                ;
;
; When Timer 3 has overflowed, the overflow counter should
; incremented. Then the Timer 3 overflow flag, the source o
; the interrupt, should be cleared. then return to see if a
; capture also occurred.
;
; NOTE: Timer 3 has a process dependent silicon problem whi
; may give 2 overflow interrupts for each "real" over
; interrupt. The first ("false") interrupt will occur
; the timer 0xFEFF to 0xFF00 transition. The second
; interrupt then occurs at the "real" overflow. A soft
; fix is included in this routine (T3OVFL). These add
; lines are designated by ;** as a comment. Once this
; been corrected, these lines of code can be removed.
```

Capture Module

```

;          The routine lable must remain.
;
0057 8E16      T3OVFL      BCF      PIR,6          ; Clear int
0058 B802                        MOVLB   02          ;** Switch
0059 9F13                        BTFSCL TMR3H,7      ;** Is Time
005A 0005                        RETFIE          ;** the "false"
;** Return
;** interrupt.
005B B801                        MOVLB   01          ;** "real" overflow.
;** Return
005C 1523                        INCF   T3OFLCNTR,1      ; increment
005D C043                        GOTO   CK_CAP          ; return to
; interrupt also occurred
;
; Other Interrupt routines. (Not utilized in this example)
;
005E 0005      EXT_INT      RETFIE          ; RAO/INT i
005F 0005      RTCCINT      RETFIE          ; (NOT used in
; RTCC over
0060 0005      RT_INT       RETFIE          ; (NOT used in
; RAI/RT in
; (NOT used in
0061 C028      SRESET       GOTO   START      ; If progra
; START and reinitialize.
;
;
; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.
; 2
07FF C061      ORG      END_OF_PROG_MEM      ;
GOTO SRESET          ; The progr
; do a system reset
END
```

```
Errors : 0
Warnings : 0
```

APPENDIX B - PERIOD MEASUREMENT WITH FREE RUNNING TIMER EXAMPLE CODE

MPASM B0.54

PAGE 1

```

; This is the basic outline for a program that can determine the
; frequency of an input, via input capture. This routine uses an
; 8-bit register to count the times that timer3 overflowed. At the
; Max crystal frequency of 16 MHz, this gives an overflow time of
; (2**16) (256 + 1) (250 nS) > 4.21 sec or a frequency < 0.25 Hz. If
; measurement of longer time intervals is required, the overflow
; counter could be extended to 16 (or more) bits.
;
; Timer 3 in this example is a free running timer. The input
; capture is generated on the RB1/CAP2 pin. There is a flag
; that specifies if this is the 1st or 2nd capture.
; The first capture is the start of the period measurement. The
; second capture value gives the end of the period. In this type
; of measurement if the 2nd capture value < the 1st capture value
; then the overflow counter should be decremented.
;
; Do the EQUate table
;
LIST      P=17C42, C=80, T=ON
0020      IC2OF          EQU    0x20          ; T3 overflow regis
0021      IC2BH          EQU    0x21          ; T3 ICA2 MSB regis
0022      IC2BL          EQU    0x22          ; T3 ICA2 LSB regis
0023      IC2AH          EQU    0x23          ; T3 ICB2 MSB regis
0024      IC2AL          EQU    0x24          ; T3 ICB2 LSB regis
0025      T3OFLCNTR     EQU    0x25          ; Temperay T3 overf
;
0026      FLAG_REG      EQU    0x26          ; Register that has
;
;      FLAG_REG bit   7   6   5   4   3   2   1   0
;                  -   -   -   -   -   -   -   - UFL CAP1
;      CAP1 = 0, 1st Capture
;            = 1, 2nd Capture
;
;      UFL = 0, No Underflow
;            = 1, Underflow during subtract
;
07FF      END_OF_PROG_MEM EQU    0x07FF
;
0004      ALUSTA        EQU    0x04
0006      CPUSTA        EQU    0x06
0007      INTSTA        EQU    0x07
000A      W              EQU    0x0A
;
0012      PORTB         EQU    0x12          ; Bank 0
;
0016      PIR           EQU    0x16          ; Bank 1
0017      PIE           EQU    0x17
;
0012      TMR3L         EQU    0x12          ; Bank 2
0013      TMR3H         EQU    0x13
0016      T3PRL        EQU    0x16
0017      T3PRH        EQU    0x17
;

```

Capture Module

```
0014          CA2L          EQU    0x14          ; Bank 3
0015          CA2H          EQU    0x15
0016          TCON1        EQU    0x16
0017          TCON2        EQU    0x17
                                EQU    0x0000          ; Origin fo

0000 C028          GOTO    START          ; On reset,
                                ; the program
                                ORG    0x0008          ; Origin fo
                                ; interrupt vector
0008 C068          GOTO    EXT_INT        ; Goto the
                                ; on RA0/INT routine
                                ORG    0x0010          ; Origin fo
                                ; overflow interrupt vector
0010 C069          GOTO    RTCCINT       ; Goto the
                                ; routine
                                ORG    0x0018          ; Origin fo
                                ; RAL/RT interrupt vector
0018 C06A          GOTO    RT_INT        ; Goto the
                                ; RAL/RT routine
                                ORG    0x0020          ; Origin fo
                                ; of any enabled peripheral
0020 C03E          GOTO    PER_INT       ; Goto the
                                ; peripheral routine
                                ORG    0x0028          ; Origin fo
                                ; program memory
0028 8406          START    BSF    CPUSTA,4 ; Disable A
                                ; Global Interrupt Disable
                                ; (GLINTD) bit.
                                ;
                                ; Place Main program here
0029 B803          MAIN    MOVLB  3      ; Select re
002A 2817          CLRFB  TCON2,0      ; Stop the
002B B070          MOVLW  0x070        ; Initalize
002C 0116          MOVWF  TCON1        ; T1 (8-b
                                ; and T3 run off the internal
                                ; system clock. Capture2
                                ; captures on the rising edge.
;
; Initialize Timer 3, load the timer with the number of cycl
; the device executes (from RESET) before the timer is turn
; Therefore the offset is required due to software overhead
;
002D B802          MOVLB  2            ; Select re
002E 280A          CLRFB  W,0          ; Clear the
002F 0126          MOVWF  FLAG_REG     ; Initalize
0030 0113          MOVWF  TMR3H        ; Timer3 MS
```

Capture Module

```
0031 B013          MOV LW    0x13          ; Timer3 LS
0032 0112          MOV WF    TMR3L          ;
;
; Load the Timer 3 period register with 0xFFFF, which will
; interrupt on the overflow of Timer3
;
0033 B0FF          MOV LW    0xFF          ;
0034 0117          MOV WF    T3PRH          ;
0035 0116          MOV WF    T3PRL          ;
;
; the timer should be started and interrupts enabled.
;
0036 B803          MOV LB    3              ; Select re
0037 8217          BSF      TCON2,2        ; Turn on t
0038 8307          BSF      INTSTA,3       ; Turn on P
0039 B801          MOV LB    1              ; Select re
003A B048          MOV LW    0x48          ; Enable Ca
003B 0117          MOV WF    PIE           ; Interr
;
; This is where you would do the things you wanted to do.
; this example will only loop waiting for the interrupts.
;
003C 8C06          WAIT      BCF      CPUSTA,4 ; Enable AL
003D C03C          GOTO     WAIT           ; Loop here
;
; Interrupt
;
; The interrupt routine for any peripheral interrupt, This
; only deals with Timer3 (T3) interrupts.
;
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routi
;
; case1 - only T3 overflow           = 12 cycles
; case2 - 1st capture                = 14 cycles
; case3 - 2nd capture                = 30 cycles
; case4 - T3 overflow and 1st capture = 34 cycles
; case5 - T3 overflow and 2nd capture = 50 cycles
;
;
003E B801          PER_INT MOV LB    1          ; Select re
003F 9E16          BTFSC   PIR,6           ; Did T3 ov
;
; If not skip next Instruction
; Inc overf
0040 C055          GOTO     T3OVFL
0041 9316          CK_CAP BTFSS   PIR,3       ; Did the R
;
; interrupt?
0042 0005          RETFIE                    ; No RB1/CA
;
; Return from Interrupt
;
; This portion of the code takes the 1st capture and stores
```

Capture Module

```

; value in register pair IC2AH:IC2AL. When the 2nd capture
; is take, its value is stored in register pair IC2BH:IC2BL
; A 16-bit subtract is performed, with the final 24-bit res
; being stored in IC2OF:IC2BH:IC2BL. This value will no lon
; be correct after the next capture occurs (IC2BH:IC2BL wil
; change), so the main routine must utilize this value befo
; it changes.
;
0043 8B16      CAPTURE      BCF      PIR,3          ; Clear Cap
0044 B803                      MOVLB   3              ; Select re
0045 9826                      BTFSC  FLAG_REG,0       ; 1st or 2n
0046 C04B                      GOTO   CAP2            ; It was th
0047 5424      CAP1         MOVPF  CA2L,IC2AL       ; Move the
0048 5523                      MOVPF  CA2H,IC2AH       ; tempora
0049 8026                      BSF    FLAG_REG,0       ; Have 1st
004A 0005                      RETFIE                    ; Return fr
004B 5422      CAP2         MOVPF  CA2L,IC2BL       ; Move the
004C 5521                      MOVPF  CA2H,IC2BH       ; tempora
; (to pervent
;
004D E061                      CALL   SUB16          ; Call rout
; 2 16-bit numbers.
; Underflow
004E 9926                      BTFSC  FLAG_REG,1       ; Underflow
004F 0725                      DECF   T3OFLCNTR,1     ; Since und
; overflow counter.
; Clear the
0050 2926                      CLRF  FLAG_REG,1       ; Clear the
; underflow and
; Store the
0051 6A25                      MOVFP  T3OFLCNTR,W     ; Store the
0052 4A20                      MOVPF  W,IC2OF         ; overflo
0053 2825                      CLRF  T3OFLCNTR,0     ; Clear the
; counts how
; overflows.
; Return fr
0054 0005                      RETFIE                    ; Return fr
;
; When Timer 3 has overflowed, the overflow counter only sh
; be incremented when the overflow occurs after a capture 1
; but before the capture 2. The 4 possible cases when enter
; the T3OVFL section of the PER_INT routine are as follows:
; Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
```


Capture Module

```

;          for Capture 1 to occur). Do Not increment count
;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
;          for Capture 2 to occur). Increment counter
;   Case 3: T3 Overflow happened after Capture. Do Not
;          increment overflow counter
;   Case 4: T3 Overflow occurred before Capture 2 and FLAG_R
;
;          (waiting for Capture 2 to occur). Increment cou

;
0055 8E16      T3OVFL      BCF      PIR,6          ; Clear Ove
0056 9316          BTFSS     PIR,3          ; Did the R
;          ; cause an interrupt?
0057 C05E          GOTO      FRO           ; No, Check
;          ; and 2nd capture
0058 B803          MOVLB     3              ; Bank 3
0059 280A          CLRF      W,0           ; W = 0
005A 3115          CPFSEQ    CA2H         ; if CA2H =
005B C05E          GOTO      FRO           ; first,
;          ; bit 0
005C B801          MOVLB     1              ; Back to b
005D C043          GOTO      CAPTURE       ; Capture h
;          ; Increment
;          ; and do capture
005E 9826      FRO          BTFSC     FLAG_REG,0 ; Between C
005F 1525          INCF      T3OFLCNTR,1 ; Yes, Inc.
0060 0005          RETFIE                    ; Return fr
;
0061 6A24      SUB16      MOVFP     IC2AL,W     ; Do the 16
0062 0522          SUBWF     IC2BL,1         ;
0063 6A23          MOVFP     IC2AH,W     ;
0064 0321          SUBWFB    IC2BH,1         ;
0065 9004          BTFSS     ALUSTA,0        ; Is the re
0066 8126          BSF       FLAG_REG,1     ; neg., Set
0067 0002          RETURN                    ; Return fr
;
; Other Interrupt routines. (Not utilized in this example)
;
0068 0005      EXT_INT    RETFIE                    ; RA0/INT i
;          ; (NOT used in
0069 0005      RTCCINT    RETFIE                    ; RTCC over
;          ; (NOT used in
006A 0005      RT_INT     RETFIE                    ; RA1/RT in
;          ; (NOT used in
;
006B C028      SRESET    GOTO      START          ; If progra

```

Capture Module

```
                                ; START and reinitialize.
;
;
; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.
;
                                ORG     END_OF_PROG_MEM ;
07FF C06B                       GOTO    SRESET   ; The progr
                                ; do a system reset
END
```

```
Errors : 0
Warnings : 0
```

APPENDIX C - PULSE WIDTH MEASUREMENT EXAMPLE CODE

MPASM B0.54

PAGE 1

```

; This is the basic outline for a program that can determine the
; Pulse Width of an input, via input capture. This routine uses an
; 8-bit register to count the times that timer3 overflowed. At the
; Max crystal frequency of 16 MHz, this gives an overflow time of
; (2*16)(256 + 1)(250 nS) > 4.21 sec or a frequency < 0.25 Hz. If
; measurement of longer time intervals is required, the overflow
; counter could be extended to 16 (or more) bits.
;
; Do the EQUate table
;
LIST      P=17C42, C=80, T=ON
0020      IC2OF          EQU      0x20          ; T3 overflow regis
0021      IC2BH          EQU      0x21          ; T3 ICA2 MSB regis
0022      IC2BL          EQU      0x22          ; T3 ICA2 LSB regis
0023      IC2AH          EQU      0x23          ; T3 ICB2 MSB regis
0024      IC2AL          EQU      0x24          ; T3 ICB2 LSB regis
0025      T3OFLCNTR     EQU      0x25          ; Temperay T3 overf
;
0026      FLAG_REG      EQU      0x26          ; Register that has
;
;      FLAG_REG bit   7   6   5   4   3   2   1   0
;                  -   -   -   -   -   -   -   UFL CAP1
;      CAP1 = 0, 1st Capture
;             = 1, 2nd Capture
;
;      UFL = 0, No Underflow
;             = 1, Underflow during subtract
;
07FF      END_OF_PROG_MEM EQU      0x07FF
;
0004      ALUSTA        EQU      0x04
0006      CPUSTA        EQU      0x06
0007      INTSTA        EQU      0x07
000A      W              EQU      0x0A
;
0012      PORTB         EQU      0x12          ; Bank 0
;
0016      PIR           EQU      0x16          ; Bank 1
0017      PIE           EQU      0x17
;
0012      TMR3L         EQU      0x12          ; Bank 2
0013      TMR3H         EQU      0x13
0016      T3PRL        EQU      0x16
0017      T3PRH        EQU      0x17
;
0014      CA2L          EQU      0x14          ; Bank 3
0015      CA2H          EQU      0x15
0016      TCON1        EQU      0x16
0017      TCON2        EQU      0x17
;
ORG      0x0000          ; Origin fo
0000 C028              GOTO      START        ; On reset,
;
;      the program

```

Capture Module

```
                                ORG    0x0008    ; Origin fo
                                ; interrupt vector
0008 C072    GOTO   EXT_INT    ; Goto the
                                ; on RA0/INT routine
                                ; Origin fo
                                ORG    0x0010
                                ; overflow interrupt vector
0010 C073    GOTO   RTCCINT    ; Goto the
                                ; routine
                                ; Origin fo
                                ORG    0x0018
                                ; RA1/RT interrupt vector
0018 C074    GOTO   RT_INT    ; Goto the
                                ; RA1/RT routine
                                ; Origin fo
                                ORG    0x0020
                                ; of any enabled peripheral
0020 C03E    GOTO   PER_INT    ; Goto the
                                ; peripheral routine
                                ; Origin fo
                                ORG    0x0028
                                ; program memory
0028 8406    START   BSF     CPUSTA,4    ; Disable A
                                ; Global Interrupt Disable
                                ; (GLINTD) bit.
                                ;
                                ; Place Main program here
                                ; Select re
0029 B803    MAIN   MOVLB   3
002A 2817    CLRF   TCON2,0    ; Stop the
002B B070    MOVLW  0x070    ; Initalize
002C 0116    MOVWF  TCON1    ; T1 (8-b
                                ; and T3 run off the internal
                                ; system clock. Capture2
                                ; captures on the rising edge.
                                ;
                                ; Initialize Timer 3, load the timer with the number of cycl
                                ; the device executes (from RESET) before the timer is turn
                                ; Therefore the offset is required due to software overhead
                                ;
002D B802    MOVLB   2    ; Select re
002E 280A    CLRF   W,0    ; Clear the
002F 0126    MOVWF  FLAG_REG    ; Initalize
0030 0113    MOVWF  TMR3H    ; Timer3 MS
0031 B013    MOVLW  0x13    ; Timer3 LS
0032 0112    MOVWF  TMR3L    ;
                                ;
                                ; Load the Timer 3 period register with 0xFFFF, which will
                                ; interrupt on the overflow of Timer3
                                ;
0033 B0FF    MOVLW  0xFF    ;
```

Capture Module

```
0034 0117          MOVWF  T3PRH          ;
0035 0116          MOVWF  T3PRL          ;
;
; the timer should be started and interrupts enabled.
;
0036 B803          MOVLB  3              ; Select re
0037 8217          BSF    TCON2,2        ; Turn on t
0038 8307          BSF    INTSTA,3       ; Turn on P
0039 B801          MOVLB  1              ; Select re
003A B048          MOVLW  0x48           ; Enable Ca
003B 0117          MOVWF  PIE            ; Interr
;
; This is where you would do the things you wanted to do.
;
; this example will only loop waiting for the interrupts.
;
003C 8C06          WAIT    BCF    CPUSTA,4      ; Enable AL
003D C03C          GOTO   WAIT            ; Loop here
;
; Interrupt
;
; The interrupt routine for any peripheral interrupt, This
; only deals with Timer3 (T3) interrupts.
;
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routi
;
; case1 - only T3 overflow           = 12 cycles
; case2 - 1st capture                = 20 cycles
; case3 - 2nd capture                = 34 cycles
; case4 - T3 overflow and 1st capture = 32 cycles
; case5 - T3 overflow and 2nd capture = 44 cycles
;
;
003E B801          PER_INT MOVLB  1          ; Select re
003F 9E16          BTFSC  PIR,6           ; Did T3 ov
;
; If not skip
; Inc overfl
0040 C05A          GOTO   T3OVFL
0041 9316          CK_CAP BTFSS  PIR,3      ; Did the R
;
; interrupt?
; No RB1/CA
0042 0005          RETFIE                ; No RB1/CA
;
; Return from
;
; This portion of the code takes the 1st capture and stores
; value in register pair IC2AH:IC2AL. When the 2nd capture
; is take, its value is stored in register pair IC2BH:IC2BL
; A 16-bit subtract is performed, with the final 24-bit res
; being stored in IC2OF:IC2BH:IC2BL. This value will no lon
; be correct after the next capture occurs (IC2BH:IC2BL wil
```

Capture Module

```
        ; change), so the main routine must utilize this value befo
        ; it changes.
        ;
0043 B803    CAPTURE      MOVLB   3                ; Select re
0044 9826                BTFSC   FLAG_REG,0        ; Capture o
0045 C04B                GOTO    FALLING           ; It was th
0046 5424    RISING      MOVPPF  CA2L,IC2AL        ; Move the
0047 5523                MOVPPF  CA2H,IC2AH        ; tempora
0048 8026                BSF     FLAG_REG,0        ; Set flag
0049 8E16                BCF     TCON1,6           ; Change ed
                                ; to falling
004A C055                GOTO    FALSE_C           ;** With th
                                ;** edge, we have
004B 5422    FALLING     MOVPPF  CA2L,IC2BL        ; Move the
004C 5521                MOVPPF  CA2H,IC2BH        ; tempora
                                ; (to pervent
                                ;
004D E06B                CALL    SUB16            ; Call rout
                                ; 2 16-bit numbers.
004E 9926                BTFSC   FLAG_REG,1        ; Underflow
004F 0725                DECF    T3OFLCNR,1        ; Since und
                                ; overflow counter.
0050 2926                CLRF   FLAG_REG,1        ; Clear the
                                ; underflow and
0051 6A25                MOVPPF  T3OFLCNR,W        ; Store the
0052 4A20                MOVPPF  W,IC2OF          ; overflo
0053 2825                CLRF   T3OFLCNR,0        ; Clear the
                                ; counts how many
                                ; overflows.
0054 8616                BSF     TCON1,6           ; Change ed
                                ; to rising
        ;
        ; Note when you change the edge of the capture, an addition
        ; is generated. The capture register must be read before th
        ; flag is cleared.
        ;
0055 550A    FALSE_C     MOVPPF  CA2H,W            ; Dummy rea
0056 540A                MOVPPF  CA2L,W            ;
                                ;
0057 B801                MOVLB   1                ; Select re
0058 8B16                BCF     PIR,3             ; Clear Cap
0059 0005                RETFIE                    ; Return fr
```

Capture Module

```

; T3 overflow or falling edge
; capture
;
;
; When Timer 3 has overflowed, the overflow counter only sh
; be incremented when the overflow occurs after a capture 1
; but before the capture 2. The 6 possible cases when enter
; the T3OVFL section of the PER_INT routine are as follows:
;
; Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
; for Capture 1 to occur). Do Not increment count
; Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
; for Capture 2 to occur). Increment counter
; Case 3: T3 Overflow, Then Capture1 happened. Do Not
; increment overflow counter
; Case 4: T3 Overflow, Then Capture 2 happened
; Increment counter
; Case 5: Capture1, Then T3 Overflow happened
; Increment counter
; Case 6: Capture2, Then T3 Overflow happened. Do Not
; Increment counter
;
005A 8E16      T3OVFL      BCF      PIR,6      ; Clear Overflow in
005B 9316      BTFS      PIR,3      ; Did the RB1/CAP2
; cause an interrupt?
005C C068      GOTO      FR0      ; No, only overflow
005D B803      MOVLB     3      ; Bank 3
005E 280A      CLRF      W,0      ; W = 0
005F 9826      BTFSC     FLAG_REG,0 ; T3 overflow with
; or Capture 2?
0060 C064      GOTO      OF_C1     ; Overflow with Cap
0061 3115      OF_C2      CPFSEQ   CA2H     ; if CA2H = 0, over
; first
0062 1525      INCF      T3OFLCNTR,1 ; Increment counter
0063 C043      GOTO      CAPTURE   ; Do capture routin
0064 3115      OF_C1      CPFSEQ   CA2H     ; if CA2H = 0, over
; first
0065 C043      GOTO      CAPTURE   ; Capture happened
; Increment overflow counter
; and do capture routine
0066 1525      INCF      T3OFLCNTR,1 ; Yes, Inc. the ove
0067 C043      GOTO      CAPTURE   ; Do capture routin
;
; Only increment overflow counter if between 1st and 2nd ca
;
0068 9826      FR0      BTFSC     FLAG_REG,0 ; Between Capture 1
0069 1525      INCF      T3OFLCNTR,1 ; Yes, Inc. the ove

```

Capture Module

```
006A 0005          RETFIE          ; Return from overf

;
006B 6A24          SUB16          MOVFP  IC2AL,W      ; Do the 16-bit sub
006C 0522          SUBWF  IC2BL,1      ;
006D 6A23          MOVFP  IC2AH,W      ;
006E 0321          SUBWFB IC2BH,1      ;
006F 9004          BTFSS  ALUSTA,0    ; Is the result pos
0070 8126          BSF     FLAG_REG,1  ; neg., Set the und
0071 0002          RETURN         ; Return from the s

;
; Other Interrupt routines. (Not utilized in this example)

;
0072 0005          EXT_INT          RETFIE          ; RA0/INT interrupt
; (NOT used in this program)
0073 0005          RTCCINT          RETFIE          ; RTCC overflow int
; (NOT used in this program)
0074 0005          RT_INT           RETFIE          ; RA1/RT interrupt
; (NOT used in this program)
;
0075 C028          SRESET          GOTO   START      ; If program became
; START and reinitialize.

;
; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.
;
07FF C075          ORG     END_OF_PROG_MEM ;
                  GOTO   SRESET      ; The progr
; do a system reset
                  END
```

```
Errors   :    0
Warnings :    0
```


APPENDIX D - PERIOD MEASUREMENT WITH PRESCALER EXAMPLE CODE

MPASM B0.54

PAGE 1

```

; This is the basic outline for a program that can determine the
; frequency of an input, via input capture. The input capture has
; been selected to capture on the 16th rising edge. This is useful
; for high frequency inputs, where an interrupt on each rising edge
; would not be able to be serviced (at that rate). This particular
; example can support an input signal with a period of approximately
; 625 nS. Without the divide by 16 selected, this is approximately
; 10 us. This period time increases (frequency decreases) as the
; overhead in the main routine increases.
;
; This routine uses an 8-bit register to count the times that timer3
; overflowed. At the Max crystal frequency of 16 MHz, this gives an
; overflow time of (16)(2**8 + 1)(2**16)(250 nS) > 67.37 sec. If
; measurement of longer time intervals is required, the overflow
; counter could be extended to 16 (or more) bits.
;
; Timer 3 in this example is a free running timer. The input
; capture is generated on the RB1/CAP2 pin. There is a flag
; that specifies if this is the 1st or 2nd capture.
; The first capture is the start of the period measurement. The
; second capture value gives the end of the period. In this type
; of measurement If the 2nd capture value < the 1st capture value
; then the overflow counter should be decremented.
;
; Do the EQUate table
;
LIST          P=17C42, T=ON, C=80
0020          IC2OF          EQU    0x20          ; T3 overflow regis
0021          IC2BH          EQU    0x21          ; T3 ICA2 MSB regis
0022          IC2BL          EQU    0x22          ; T3 ICA2 LSB regis
0023          IC2AH          EQU    0x23          ; T3 ICB2 MSB regis
0024          IC2AL          EQU    0x24          ; T3 ICB2 LSB regis
0025          T3OFLCNTR     EQU    0x25          ; Temperay T3 overf
;
0026          FLAG_REG      EQU    0x26          ; Register that has
;
;          FLAG_REG bit   7   6   5   4   3   2   1   0
;          -   -   -   -   -   -   -   -   UFL CAP1
;          CAP1 = 0, 1st Capture
;                = 1, 2nd Capture
;
;          UFL = 0, No Underflow
;                = 1, Underflow during subtract
;
07FF          END_OF_PROG_MEM EQU 0x07FF
;
;
0004          ALUSTA        EQU    0x04
0006          CPUSTA        EQU    0x06
0007          INTSTA        EQU    0x07
000A          W             EQU    0x0A
;
0012          PORTB         EQU    0x12          ; Bank 0

```

Capture Module

```

;
0016      PIR      EQU    0x16      ; Bank 1
0017      PIE      EQU    0x17
;
0012      TMR3L    EQU    0x12      ; Bank 2
0013      TMR3H    EQU    0x13
0016      T3PRL    EQU    0x16
0017      T3PRH    EQU    0x17
;
0014      CA2L     EQU    0x14      ; Bank 3
0015      CA2H     EQU    0x15
0016      TCON1    EQU    0x16
0017      TCON2    EQU    0x17

;
ORG      0x0000      ; Origin fo

0000 C028      GOTO   START      ; On reset,
;
; the program
ORG      0x0008      ; Origin fo
;
; interrupt vector
0008 C068      GOTO   EXT_INT    ; Goto the
;
; on RA0/INT routine
ORG      0x0010      ; Origin fo
;
; overflow interrupt vector
0010 C069      GOTO   RTCCINT    ; Goto the
;
; routine
ORG      0x0018      ; Origin fo
;
; RAL/RT interrupt vector
0018 C06A      GOTO   RT_INT     ; Goto the
;
; RAL/RT routine
ORG      0x0020      ; Origin fo
;
; of any enabled peripheral
0020 C03E      GOTO   PER_INT    ; Goto the
;
; peripheral routine
ORG      0x0028      ; Origin fo
;
; program memory
0028 8406      START  BSF        CPUSTA,4 ; Disable A
;
; Global Interrupt Disable
; (GLINTD) bit.
;
; Place Mai
MAIN
0029 B803      MOVLE  3          ; Select re
002A 2817      CLRFB  TCON2,0    ; Stop the
002B B0F0      MOVLW  0x0F0     ; Initalize
002C 0116      MOVWF  TCON1     ; T1 (8-b
;
; and T3 run off the internal
; system clock. Capture2
; captures on the 16th rising edge.
;
; Initialize Timer 3, load the timer with the number of cycl
; the device executes (from RESET) before the timer is turn

```

Capture Module

```
; Therefore the offset is required due to software overhead
;
002D B802          MOVLB  2          ; Select re
002E 280A          CLRFB  W,0        ; Clear the
002F 0126          MOVWF  FLAG_REG   ; Initialize
0030 0113          MOVWF  TMR3H     ; Timer3 MS
0031 B000          MOVLW  0x00      ; Timer3 LS
0032 0112          MOVWF  TMR3L     ;
;
; Load the Timer 3 period register with 0xFFFF, which will
; interrupt on the overflow of Timer3
;
0033 B0FF          MOVLW  0xFF      ;
0034 0117          MOVWF  T3PRH     ;
0035 0116          MOVWF  T3PRL     ;
;
; the timer should be started and interrupts enabled.
;
0036 B803          MOVLB  3          ; Select re
0037 8217          BSF    TCON2,2    ; Turn on t
0038 8307          BSF    INTSTA,3   ; Turn on P
0039 B801          MOVLB  1          ; Select re
003A B048          MOVLW  0x48      ; Enable Ca
003B 0117          MOVWF  PIE        ; Interr
;
; This is where you would do the things you wanted to do.
;
; this example will only loop waiting for the interrupts.
;
003C 8C06          WAIT    BCF    CPUSTA,4 ; Enable AL
003D C03C          GOTO   WAIT        ; Loop here
;
; Interrupt
;
; The interrupt routine for any peripheral interrupt, This
; only deals with Timer3 (T3) interrupts.
;
; Time required to execute interrupt routine. Not including
; interrupt latency (time to enter into the interrupt routi
;
; case1 - only T3 overflow          = 12 cycles
; case2 - 1st capture              = 14 cycles
; case3 - 2nd capture              = 30 cycles
; case4 - T3 overflow and 1st capture = 34 cycles
; case5 - T3 overflow and 2nd capture = 50 cycles
;
;
003E B801          PER_INT MOVLB  1          ; Select re
003F 9E16          BTFSC  PIR,6        ; Did T3 ov
; If not skip next In-
```

Capture Module

```
struction
0040 C055          GOTO    T3OVFL          ; Inc overf

0041 9316          CK_CAP  BTFSS  PIR,3          ; Did the R

                                ; interrupt?
0042 0005          RETFIE          ; No RB1/CA

                                ; Return from Interrupt
;
; This portion of the code takes the 1st capture and stores
; value in register pair IC2AH:IC2AL. When the 2nd capture
; is take, its value is stored in register pair IC2BH:IC2BL
; A 16-bit subtract is performed, with the final 24-bit res
; being stored in IC2OF:IC2BH:IC2BL. This value will no lon
; be correct after the next capture occurs (IC2BH:IC2BL wil
; change), so the main routine must utilize this value befo
; it changes.
;
0043 8B16          CAPTURE  BCF    PIR,3          ; Clear Cap
0044 B803          MOVLB   3          ; Select re
0045 9826          BTFSC  FLAG_REG,0          ; 1st or 2n
0046 C04B          GOTO    CAP2          ; It was th
0047 5424          CAP1    MOVPF  CA2L,IC2AL          ; Move the
0048 5523          MOVPF  CA2H,IC2AH          ; tempora
0049 8026          BSF    FLAG_REG,0          ; Have 1st
004A 0005          RETFIE          ; Return fr
;
004B 5422          CAP2    MOVPF  CA2L,IC2BL          ; Move the
004C 5521          MOVPF  CA2H,IC2BH          ; tempora
; (to pervent
;
004D E061          CALL   SUB16          ; Call rout
;
004E 9926          BTFSC  FLAG_REG,1          ; 2 16-bit numbers.
; Underflow
004F 0725          DECF   T3OFLCNTR,1          ; Since und
; overflow counter.
0050 2926          CLRF   FLAG_REG,1          ; Clear the
; underflow and
0051 6A25          MOVFP  T3OFLCNTR,W          ; Store the
0052 4A20          MOVPF  W,IC2OF          ; overflo
0053 2825          CLRF   T3OFLCNTR,0          ; Clear the
; counts how many
; overflows.
0054 0005          RETFIE          ; Return fr
```

Capture Module

```
;
; When Timer 3 has overflowed, the overflow counter only sh
; be incremented when the overflow occurs after a capture 1
; but before the capture 2. The 4 possible cases when enter
; the T3OVFL section of the PER_INT routine are as follows:
; Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
;           for Capture 1 to occur). Do Not increment count
; Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
;           for Capture 2 to occur). Increment counter
; Case 3: T3 Overflow happened after Capture. Do Not
;           increment overflow counter
; Case 4: T3 Overflow occurred before Capture 2 and FLAG_R
;           (waiting for Capture 2 to occur). Increment cou

;
0055 8E16      T3OVFL      BCF      PIR,6           ; Clear Ove
0056 9316      BTFSS     PIR,3           ; Did the R
;           cause an interrupt?
0057 C05E      GOTO     FR0              ; No, Check
;           ; and 2nd capture
0058 B803      MOVLB     3               ; Bank 3
0059 280A      CLRWF    W,0             ; W = 0
005A 3115      CPFSEQ   CA2H           ; if CA2H =
005B C05E      GOTO     FR0              ; first,
;           ; bit 0
005C B801      MOVLB     1               ; Back to b
005D C043      GOTO     CAPTURE          ; Capture h
;           ; Increment overflow
;           ; and do capture routine
005E 9826      FR0       BTFSC     FLAG_REG,0 ; Between C
005F 1525      INCF     T3OFLCNTR,1     ; Yes, Inc.
0060 0005      RETFIE                    ; Return fr
;
0061 6A24      SUB16     MOVFP     IC2AL,W   ; Do the 16
0062 0522      SUBWF    IC2BL,1           ;
0063 6A23      MOVFP     IC2AH,W           ;
0064 0321      SUBWFB   IC2BH,1           ;
0065 9004      BTFSS    ALUSTA,0        ; Is the re
0066 8126      BSF      FLAG_REG,1       ; neg., Set
0067 0002      RETURN                    ; Return fr
;
; Other Interrupt routines. (Not utilized in this example)
;
0068 0005      EXT_INT   RETFIE          ; RA0/INT i
;           ; (NOT used in this
```

Capture Module

```
0069 0005      RTCCINT      RETFIE                ; RTCC over
                                           ; (NOT used in this
006A 0005      RT_INT      RETFIE                ; RA1/RT in
                                           ; (NOT used in this program)
                                           ;
006B C028      SRESET      GOTO   START          ; If progra
                                           ; START and reinitialize.
;
;
; When the executed address is NOT in the program range, th
; 16-bit address should contain all 1's (a CALL 0x1FFF). At
; this location you could branch to a routine to recover or
; shut down from the invalid program execution.
;
007F C06B      ORG      END_OF_PROG_MEM          ;
GOTO      SRESET          ; The progr
                                           ; do a system reset

END
```

```
Errors : 0
Warnings : 0
```



Serial Port Utilities

INTRODUCTION

PIC17C42 has an on chip high speed Universal Synchronous Asynchronous Receiver Transmitter (USART). The serial port can be configured to operate either in full-duplex asynchronous mode or half duplex synchronous mode. The serial port has a dedicated 8-bit baud rate generator. Either 8- or 9-bits can be transmitted/received.

This application note provides information on using the serial port, parity generation, serial port expansion, RS-232 interface, I/O port expansion using synchronous mode of serial port.

SERIAL PORT USAGE

Brief code to setup serial port, receive and transmit data is given below. Small sections of code for both asynchronous and synchronous mode is given.

EXAMPLE 1

```
SPBRG   :   25           :   9600 baud @ 16 Mhz input clock
TXSTA   :   00100000 (20h) :   8-bit transmission, async mode
RCSTA   :   10010000 (90h) :   8-bit reception, enable serial port, enable reception

;*****
;       Sample Code For Asynchronous Mode Serial Port Setup
;*****

#define ClkFreq      16000000           ; input clock frequency = 16 Mhz
#define baud(X)      ((10*ClkFreq/(64*X))+5)/10 - 1
#define TXSTA_INIT  0xB0
#define RCTSA_INIT  0x90

#include          "17C42.h"           ; file containing the Register Definitions

Setup_Async_Mode
    movlb    0           ; SPBRG, TXSTA & RCSTA are in bank 0
    movlw   baud(9600)   ; equals 25 for 9600 baud
    movwf  SPBRG        ; baud rate generator is reset & initialized
    movlw  TXSTA_INIT   ; 8-bit transmission, async mode
    movwf  TXSTA
    movlw  RCSTA_INIT   ; 8-bit reception, enable serial port,
    movwf  RCSTA        ; enable reception

    return

;*****
```

Asynchronous Mode Setup

Asynchronous mode set-up requires selection of 8/9-bits of data transfer, baud rate, setting the baud rate generator and configuring the TXSTA & RCSTA control registers. The baud rate generator is set by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{\text{Input_Clk_Freq}}{64 * \text{Baud_Rate}} - 1$$

For example, to select a baud rate of 9600 bits/sec with input clock frequency of 16 Mhz, SPBRG is computed from the above equation to be 25. Once the Baud Rate Generator is setup, it is necessary to configure the TXSTA & RCSTA control registers as follows (please refer to the data sheet) :



Serial Port Utilities

Synchronous Mode Setup

Synchronous mode setup requires selection of 8/9-bits of data transfer, bit rate, setting the baud rate generator and configuring the TXSTA & RCSTA control registers. The baud rate generator is set by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{\text{Input_Clk_Freq}}{4 * \text{Baud_Rate}} - 1$$

For example, to select a bit rate of 1Mhz with input clock frequency of 16 Mhz, SPBRG is computed from the above equation to be 3. Once the Baud Rate Generator is setup, it is necessary to configure the TXSTA & RCSTA control registers as follows (please refer to the data sheet) :

EXAMPLE 2

```
SPBRG : 3 : 1 Mbits/sec @ 16 Mhz input clock
TXSTA : 10110000 (B0h) : 8-bit transmission, Sync mode (MASTER)
RCSTA : 10010000 (90h) : 8-bit reception, enable serial port, continuous
reception

;*****
; Sample Code For Synchronous Mode (MASTER) Serial Port Setup
;*****

#define ClkFreq 16000000 ; input clock frequency = 16 Mhz
#define baud(X) ((10*ClkFreq/(4*X))+5)/10 - 1
#define TXSTA_INIT 0xB0
#define RCSTA_INIT 0x90

#include "17C42.h" ; file containing the Register Definitions

Setup_Sync_Master_Mode
    movlb 0 ; SPBRG, TXSTA & RCSTA are in bank 0
    movlw baud(1000000) ; equals 3 for 1 Mbits/sec
    movwf SPBRG ; baud rate generator is reset & initialized
    movlw TXSTA_INIT
    movwf TXSTA ; 8-bit transmission, async mode
    movlw RCSTA_INIT
    movwf RCSTA ; 8-bit reception, enable serial port,
; enable reception

    return
;*****
```


Receiving Data (Software Polling)

The sample code provides a way to read the received serial data by software polling (with no serial port interrupts). This applies to both asynchronous and synchronous mode. Software polling is done by checking the RBFL bit (PIR<0>). If this bit is set it means that a word has been received (8 bits are in RCREG and the 9th bit in RCSTA<0>).

EXAMPLE 3

```
;*****  
;  
;           Return The 8-bit received Data By Software Polling  
;           The received data is returned in location SerInData  
;*****  
Get_Serial_Data_Poll  
    movlb 1           ; PIR is in bank 1  
PollRcv   btfnss    PIR,0       ; chech the RBFL bit  
          goto     PollRcv      ; loop until char received, assume WDT is off  
          movlb 0           ; RCREG is in bank 0  
          movfp   RCREG,SerInData  
          return          ; Received 8-bits are in SerInData  
;*****
```

Transmitting Data (Software Polling)

The sample code provides a way to transmit serial data by software polling (no serial port interrupts). Software polling is done by checking the TBMT bit (PIR<0> in bank 1) to be one, indicating the transfer of TXREG to the serial shift register.

EXAMPLE 4

```
;*****  
;  
;           Transmit 8-bit Data By Software Polling  
;           The data to be transmitted is in location SerOutData  
;*****  
Send_Serial_Data_Poll  
    movlb 1           ; PIR is in bank 1  
PollTxmt  btfnss    PIR,1       ; chech the TBMT bit of PIR register in bank1  
          goto     PollTxmt      ; loop until char received, assume WDT is off  
          movlb 0           ; RCREG is in bank 0  
          movfp   SerOutData,TXREG  
          return          ; Received 8-bits are in SerInData  
;*****
```



Serial Port Utilities

Transmitting & Receiving A Block Of Data (Interrupt Driven)

A general purpose routine which is interrupt driven that transmits and receives a block of data is provided. The reception or transmission of the block is ended when an end of block character is detected. As an example, the

end of block is identified by a 0. The block of data to be transmitted is stored in the program memory and `TABLRD` instruction is used to transfer this example data to the file registers and serial port. The user may modify this code to a more general purpose routine that suits his application.

EXAMPLE 5

MPASM B0.54

PAGE 1

```

;          TITLE   `Serial Interface Routines
;          LIST P=17C42, C=80. I=ON, R=DEC

;This is a short program to demonstrate how to transmit and
;serial data using the PIC17C42.
;
;A message will be transmitted and routed right back to the
;and read. The read information will be saved in an interna
;
;          include "p17reg.h"

0080          TX_BUFFER    equ    0x80
00B0          RX_BUFFER    equ    0xB0
0020          RXPTR        equ    0x20
0021          TXPTR        equ    0x21
0022          SERFLAG      equ    0x22
0023          RTINUM       equ    0x23
0001          RXDONE       equ    1
0000          TXDONE       equ    0
0002          HILOB        equ    2
;
;          ORG    0
0000 C072          goto    start
;
;          ORG    0x0010          ;vector for rtcc interrupt

;rtcc_int          ;not used here
;
;          ORG    0x0020          ;vector for peripheral inte

perf_int
0020 C04D          goto    service_perf  ;service the interrupts
;
;          ORG    0x0030

;initialize the serial port: baud rate interrupts etc.
init_serial
0030 2922          clrf    SERFLAG          ;clear all flags
0031 B800          movlb   0                ;select bank 0
0032 B007          movlw   0x07            ;select 9600 baud
0033 770A          movfp   W,SPBRG        ; /
0034 B090          movlw   0x90            ;set up serial pins
0035 730A          movfp   W,RCSTA        ; /
```

Serial Port Utilities

```

0036 2915          clrf    TXSTA          ;setup transmit status
0037 B801          movlb   1              ;select bank 1
0038 2916          clrf    PIR           ;clear all interrupts
0039 2917          clrf    PIE           ;clear all enables
003A 8017          bsf     PIE,RCIE       ;enable receive interrupt
003B B0B0          movlw   RX_BUFFER      ;set pointer to rx buffer
003C 4A20          movpf   W,RXPTR          ; /
003D 2907          clrf    INTSTA        ;clear all interrupts
003E 8307          bsf     INTSTA,PEIE     ;enable peripheral ints
003F 0005          retfie

;
;start transmission of first two bytes
start_xmit
0040 B800          movlb   0              ;select bank 0
0041 8515          bsf     TXSTA,TXEN      ;enable transmit
0042 AB0A          tablrld 1,1,W          ;load latch
0043 A216          tlrld  1,TXREG        ;load high byte
0044 B801          movlb   1              ;select bank 1
empty_chk
0045 9116          btfsz  PIR,TBMT        ;TXBUF empty?
0046 C045          goto   empty_chk      ;no then keep checking
0047 B800          movlb   0              ;select bank 0
0048 A916          tablrld 0,1,TXREG     ;load lo byte
0049 B801          movlb   1              ;select bank 1
004A 8117          bsf     PIE,TXIE       ;enable transmit interrupts

004B 8222          bsf     SERFLAG,HILOB   ;set up next for high byte

004C 0002          return

;
;
;
service_perf
;check for transmit or receive interrupts only
004D 9816          btfsz  PIR,RBFL        ;RX buffer full?
004E C062          goto   service_recv   ;yes then service
004F 9116          btfsz  PIR,TBMT        ;TX buffer empty?
0050 C060          goto   exit_perf      ;no, ignore other int.
service_xmt
0051 9822          btfsz  SERFLAG,TXDONE  ;all done?
0052 C060          goto   exit_perf      ;yes then quit
0053 9A22          btfsz  SERFLAG,HILOB   ;if clr, do low byte
0054 C057          goto   rd_hi         ;else read high byte
0055 A90A          tablrld 0,1,W          ;read lo
0056 C058          goto   sx_cont       ;continue
rd_hi
0057 A20A          tlrld  1,W            ;read high byte
sx_cont
0058 3A22          btg    SERFLAG,HILOB   ;toggle flag
0059 B800          movlb   0              ;bsr=0
005A 4A16          movpf   W,TXREG        ;load tx reg
005B 330A          tstfsz W              ;last byte?
005C C060          goto   exit_perf      ;no then cont
end_xmt
005D B801          movlb   1              ;select bank 1
005E 8917          bcf    PIE,TXIE       ;disable tx interrupt
005F 8022          bsf     SERFLAG,TXDONE  ;set done flag
exit_perf
0060 8F07          bcf    INTSTA,PEIR    ;clear peripheral int
0061 0005          retfie

;
service_recv
0062 9922          btfsz  SERFLAG,RXDONE  ;RX complete?
0063 C060          goto   exit_perf      ;exit int
0064 6120          movfp  RXPTR,FSR0     ;get pointer
0065 B800          movlb   0              ;select bank 0
0066 6014          movfp  RCREG,F0       ;load received value
0067 290A          clrf   W              ;clr W
0068 3200          cpdfsgt F0          ;value = 0?

```

Serial Port Utilities

```
0069 C06D          goto    end_recv    ;yes then end
006A 1501          incf    FSR0        ;inc pointer
006B 4120          movpf   FSR0,RXPTR  ;save pointer
006C C060          goto    exit_perf   ;return from int
                    end_recv
006D 8122          bsf     SERFLAG,RXDONE ;set flag
006E 2907          clrf   INTSTA      ;clear all int
006F B801          movlb  1           ;select bank 1
0070 8817          bcf    PIE,RCIE    ;disable rx interrupts
0071 C060          goto    exit_perf   ;return
;
start
0072 2909          clrf   FSR1        ;assign FSR1 as S.P.
0073 0709          decf   FSR1        ; /
0074 B020          movlw  0x20       ;clear ram space
0075 610A          movfp  W,FSR0      ;do indirect addressing
                    start1
0076 2900          clrf   F0         ;clear ram
0077 1F01          incfsz FSR0       ;inc and skip if done
0078 C076          goto   start1     ;
0079 E030          call  init_serial ;initialize serial port
Warning: MESSAGE not a single byte quantity
007A B000          movlw  MESSAGE    ;load table pointer
007B 4A0D          movpf  W,TBLPTRL  ; /
007C B001          movlw  page MESSAGE ; /
007D 4A0E          movpf  W,TBLPTRH  ; /
007E E040          call  start_xmit  ;start transmission
                    chk_end
007F 9122          btfss  SERFLAG,RXDONE ;receive all?
0080 C07F          goto   chk_end    ;no then keep checking
;
loop
0081 C081          loop   goto    loop    ;spin wheel
;
                    ORG    0x100
MESSAGE
0100 5468 6520 636F DATA  "The code is: Tea for the Tillerman"
0103 6465 2069 733A
0106 2054 6561 2066
0109 6F72 2074 6865
010C 2054 696C 6C65
010F 726D 616E
0111 0000          DATA  0
;
;
                    END
```

```
Errors      : 0
Warnings    : 0
```

PARITY GENERATION

Since the serial port of PIC17C42 does not have an on chip parity generator, parity is generated using software. It takes only 10 program memory words and executes in 10 instruction cycles to generate parity. Since the serial port of PIC17C42 can operate in a 9-bit mode, the parity bit can be generated in software and transmitted as the 9th bit or be compared with the received 9th bit.

In case of transmission, set TX8/9 to 1 (< 6> of TXSTA) to enable 9-bit transmission and write the computed parity bit to TXD8 (TXSTA<0>). The 9th bit (parity bit) must be written prior to writing the 8 data bits to TXREG.

In case of reception, first of all enable 9-bit reception by setting RC8/9 to 1 (RCSTA<6>). Upon successful reception, the 9 bit is received in RCD8 (RCSTA<0>). Parity of the 8 bits of received data is computed using the routine listed below and compared with the 9-bit received.

EXAMPLE 6

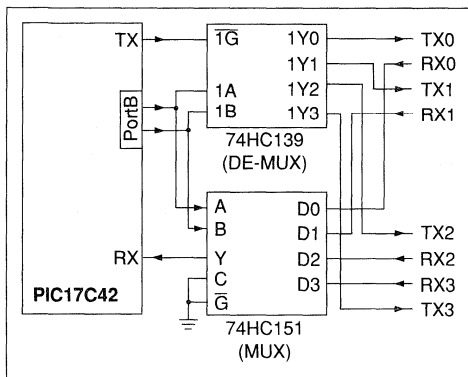
```
*****  
; Generate Parity Bit for the 8 bit register 'data'  
; The parity bit is stored in Bit 0 of 'parity'  
;  
*****  
  
#define ODD_PARITY FALSE  
  
    swapf    data,w  
    xorwf   data,w  
    movwf   parity  
    rrrncf  parity  
    rrrncf  parity  
    xorwf   parity,w  
    andlw   0x03  
    addlw   0x01  
    rrrncf  wreg  
    movwf   parity  
#if ODD_PARITY  
    btg    parity,0  
#endif
```

Serial Port Utilities

SERIAL PORT EXPANSION

The PIC17C42 has only one serial port. For applications that require the PIC17C42 to communicate with multiple serial ports, a scheme the multiplexes and demultiplexes the RX and TX pins is provided below. This method is suited only if no more than one UART is needed at any one time. This is the case in many applications where the microcontroller drives several output devices serially. Figure 1 shown below suggests a way to expand the on-chip serial port to 4 serial ports. To use the scheme as shown in Figure 1, The PIC17C42 must select the desired serial port by appropriately setting the two pins of PORT-B. The same scheme may be used to further expand the serial ports by using more I/O Ports.

FIGURE 1 - MULTIPLEXING THE ON-CHIP UART



RS-232 INTERFACE

Two circuits are provided to interface the CMOS levels of PIC17C42 to RS-232 levels. Figure 2 provides interface to MAX232 (MAXIM's RS-232 Driver/Receiver) with a single +5V power supply. Figure 3 provides a low cost 2 chip solution for RS-232 level translation using a single +5V supply (Note that V- of MC14C88 is connected to DTR of RS-232 Interface. By asserting DTR to low, V- gets the negative voltage from the RS-232 line). An alternative single chip low cost solution is provided in Figure 4. However 3 voltage sources (+5, +12, -12) are necessary.

FIGURE 2 - RS-232 INTERFACE TO MAX232

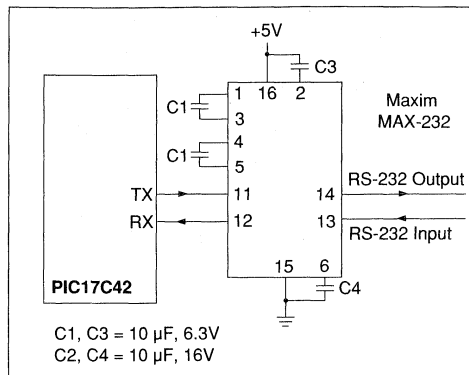


FIGURE 3 - LOW COST 2 CHIP SOLUTION USING SINGLE POWER SOURCE

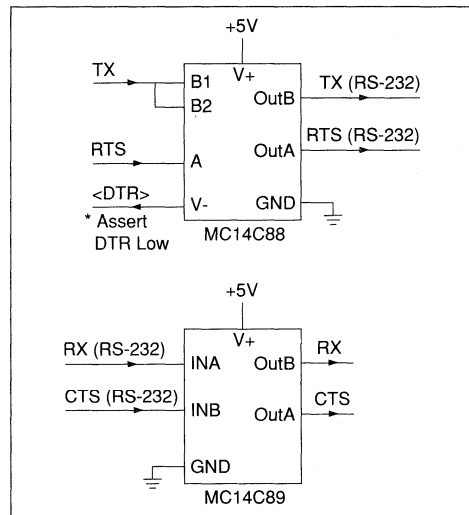


FIGURE 4 - LOW COST SINGLE CHIP SOLUTION USING 3 POWER SOURCES

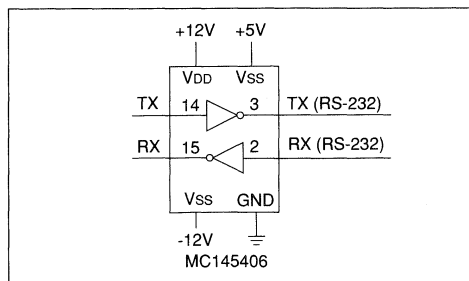


Table 1 provides the summary of RS-232 and V.28 Electrical Specifications.

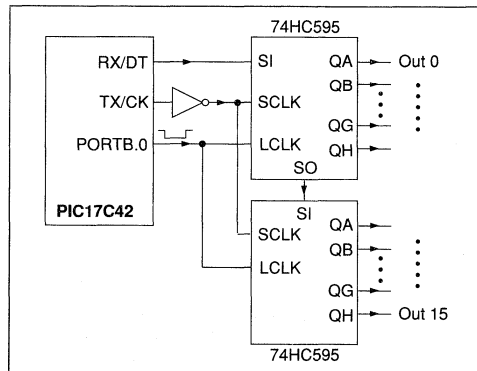
TABLE 1 - SUMMARY OF RS-232C AND V.28 ELECTRICAL SPECIFICATIONS

Parameter	Specification	Comments
Driver Output Voltage		
0 level	+5V to +15V	With 3-7kΩ load with 3-7kΩ load No load
1 level	-5V to -15V	
Max. output	±25V	
Receiver Input Thresholds (Data and clock signals)		
0 level	+3V to +25V	
1 level	-3V to -25V	
Receiver Thresholds RTS, DSR, DTR		
On level	+3V to +25V	Detects power Off condition at driver
Off level	Open circuit or -3V to -25V	
Receiver input resistance	3kΩ to 7kΩ	
Driver output resistance		
Power off condition	300Ω Min.	Vout < ±2V
Driver slew rate	30V/μs max.	3kΩ < RL < 7kΩ; 0pF < CL < 2500pF
Signalling rate	Up to 20k bits/sec.	
Cable length		
	50'/15m. Recommended max. length	Longer cables permissible, if CLoad ≤ 2500pF

I/O PORT EXPANSION USING SYNCHRONOUS MODE

Although the PIC17C42 has 33 I/O pins, most of these are multiplexed with other peripheral functions. In case more I/O ports are needed, the scheme provided below expands the I/O port using the synchronous mode of serial port by serially shifting the data. Figure 5 shows a scheme to expand the output ports to 16-bits using 2 standard logic chips (74HC595). The PIC17C42's serial port is configured in synchronous mode and set to be the MASTER. Thus serial data is available on DT (pin 22) and the clock is available on CK (pin 21). The following code will transmit 16-bits serially and clock all the 16-bits at the same time.

FIGURE 5 - OUTPUT PORT EXPANSION USING SYNCHRONOUS MODE



EXAMPLE 1

```

InitSerialPortTxmt
    movlb    0
    clrf    SPBRG        ; set to highest baud rate = CLKOUT = CLKIN/4
    movlw   0x80
    movwf   RCSTA        ; enable serial port
    movlw   0xB0
    movwf   TXSTA        ; 8 bit synchronous master mode
    bcf    DDRB,0        ; set bit 0 of PortB as output, to be used as Latch Clk
    return

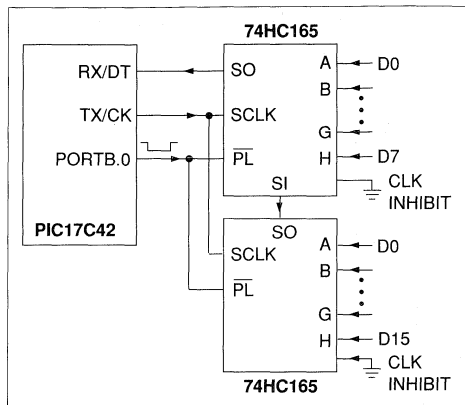
SendSerialData        ; shift out DataLo & DataHi serially
    movlb    0
    movfp   DataLo, TXREG
    nop
    ; wait for TXREG transfer : if slower baud rate is
    ; used check for TBMT reset
    movfp   DataHi, TXREG
wait    btfss    TXSTA, TRMT    ; wait until all 16 bits shifted out
    goto    wait
    bcf    PortB,0
    bsf    PortB,0        ; clock in the serial data to parallel output of HC595
    return
    
```



Serial Port Utilities

A similar scheme as shown above may be implemented in the reverse way to expand Input Ports. For this it is necessary to have a Parallel In and serial out device. Using a standard logic chip (74HC165), the scheme is shown in Figure 6. In order to read the 16-bit input data, the serial port of PIC17C42 is configured to be in Synchronous Mode Reception (MASTER mode). An I/O port (PortB<0>) is used to parallel load the 16 inputs for reading serially. A sample code to read the 16 inputs is shown below.

FIGURE 6 - INPUT PORT EXPANSION USING SYNCHRONOUS MODE



Author: Amar Palacherla
Logic Products Division

EXAMPLE 2

```

InitSerialPortRcv
    movlb    0
    clrf    SPBRG           ; set to highest baud rate = CLKOUT = CLKIN/4
    movlw   0x80
    movwf   RCSTA          ; enable serial port
    movlw   0x90
    movwf   TXSTA          ; 8-bit synchronous master mode
    bcf     DDRB,0         ; bit 0 of PortB is output, to be used as Parallel
                          ; Load
    bsf     PortB,0       ; disable parallel Load
    return

ReadSerialData
    ; shift out DataLo & DataHi serially
    movlb   0
    bcf     PortB,0       ; Parallel Load The Inputs into 74HC165
    bcf     PortB,0       ; disable parallel Load
    bsf     RCSTA,SREN    ; enable single byte reception and wait for data
    movlb   1
    wait1   btffs    PIR,RBFL
            goto     wait1   ; check until 8-bits are received
    movlb   0
    movpf   RCREG,DataLo  ; 1st byte is read
    movlb   0
    bsf     RCSTA,SREN    ; enable another byte of reception and wait for data
    movlb   1
    wait2   btffs    PIR,RBFL
            goto     wait2   ; check until 8-bits are received
    movlb   0
    movpf   RCREG,DataHi  ; 2nd byte is read
    return
    
```


Utility Math Routines

INTRODUCTION

This application note provides some utility math routines for Microchip's second generation of high performance 8-bit microcontroller, the PIC17C42. Three assembly language modules are provided, namely ARITH.ASM, FLOAT.ASM and BCD.ASM. Currently in each file the following subroutines are implemented:

ARITH.ASM

- Single precision 8 x 8 unsigned multiply
- 16 x 16 double precision multiply (signed or unsigned)
- 16 / 16 double precision divide (signed or unsigned)
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- double precision square root
- double precision numerical differentiation
- double precision numerical integration
- Pseudo Random number generation
- Gaussian distributed random number generation

FLOAT.ASM

- Binary floating point addition
- Binary floating point subtraction
- Binary floating point multiplication

BCD.ASM

- 8-bit binary to 2 digit BCD conversion
- 16-bit binary to 5 digit BCD conversion
- 5-bit BCD to 16-bit binary conversion
- 2 digit BCD addition

As more routines are available, they will be added to the library. The latest routines may be obtained either through Microchip's bulletin board or by contacting your nearest Microchip sales office for a copy on a MS-DOS 5 1/4" floppy.

These routines have been optimized wherever possible with a compromise between speed, RAM utilization, and code size. Some routines (multiplication and division) are provided in two forms, one optimized for speed and the other optimized for code size.

All the routines have been implemented as callable subroutines and the usage of each routine is explained below. At the end of the application note, the listing files of the above programs are given.

SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 1.

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of 1-bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code). These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 2.

TABLE 1

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
mpy8x8_F	speed efficient	36	36	0	used
mpy8x8_S	code efficient	13	69	1	used



Math Routines

TABLE 2

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
D_mpyF	Speed Efficient, Signed Arithmetic	204	183	1	used
D_mpyF	Speed Efficient, Unsigned Arithmetic	179	176	0	used
D_mpyS	Code Efficient, Signed Arithmetic	52	254	4	used
D_mpyS	Code Efficient, Unsigned Arithmetic	21	242	3	used

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

The data memory organization is explained in the comment section of the code. Faster execution and code space saving can be achieved by setting "MODE_FAST equ TRUE". However, setting MODE_FAST variable to TRUE restricts that operands and the 32-bit result be in data RAM locations 0x18 and 0x1F (in this mode, MOVFP and MOVPF instructions may be used to transfer data to/from any RAM location to addresses less than 0x1F). If MODE_FAST is set to FALSE, there will be no restriction on the location of the data RAM values used in this subroutine. However, the code will be slightly slower and occupies more program memory.

TABLE 3

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
D_divF	Speed Efficient, Unsigned Arithmetic	325	250	0	used
D_divF	Speed Efficient, Signed Arithmetic	354	260	1	used
D_divS	Code Efficient, Unsigned Arithmetic	31	300	1	used
D_divS	Code Efficient, Signed Arithmetic	39	312	2	used

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

TABLE 4

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Dadd	4	4	0	used
Dsub	4	4	0	used

DOUBLE PRECISION DIVISION

This routine performs a 2's complement division of two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code).

These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 3.

DOUBLE PRECISION ADDITION AND SUBTRACTION

Two routines are provided. One performs a 2's complement addition and the other one performs a 2's complement subtraction of two 16-bit binary numbers. These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 4.

NEGATE A DOUBLE PRECISION NUMBER

These routines negate a double precision number (16-bit and 32-bit). Two routines and two macros are provided to negate a 16-bit number. The subroutines use indirect addressing mode and the macros use direct addressing scheme. A macro is provided to negate a 32-bit number.

FLOATING POINT ROUTINES

Three binary floating point routines are implemented: addition, subtraction and multiplication. To use these routines, the floating point numbers should be represented as follows:

- A 16-bit signed Mantissa (in 2's complement form)
- An 80-bit signed binary exponent. For example, a number "NUM" is represented as follows:

$$\text{NUM} = (\pm 16\text{-bit Mantissa}) * (2^{\pm 8\text{-bit Exponent}})$$

Also, a general purpose Normalization routine is provided and it is recommended that the user call the normalization routine as often as possible to avoid loss of precision. The normalization routine maximizes the number of bits in the mantissa (until there are at least 14-bits).

In case of multiplication, if a 32-bit product is desired (with an 8 bit exponent) the "Mode16" should be set to TRUE. It is recommended that "Mode16" be set to FALSE (this would always produce a 16-bit mantissa, with an 8-bit exponent which is the general floating point format). Only the looped code versions are implemented. In the performance specs, the execution time is not specified, as the time very much depends on the values of the numbers and whether they are normalized or not.

DOUBLE PRECISION SQUARE ROOT

Often in many applications, one needs to find the square root of a number. Of the many numerical methods available to compute the square root of a number, the Newton-Raphson method is one of the most attractive because of its fast convergence rate. In this method, the square root of number, N, is obtained as an approximate solution of

$$f(Y) = Y^2 - N = 0$$

The function $f(Y)$ can be expanded about Y_0 using the first order Taylor polynomial expansion as:

Equation 1:

$$f(Y) = f(Y_0) + (Y - Y_0)f'(Y_0) + \frac{(Y - Y_0)^2 f''(Y_0)}{2!} + \dots$$

If X is a root of $f(Y)$, then $f(X) = 0$: Therefore,

$$f(X) = f(Y_0) + (X - Y_0)f'(Y_0) + \frac{(X - Y_0)^2 f''(Y_0)}{2!} + \dots = 0$$

If Y_0 is an approximate root of $f(Y)$, then the higher order terms in the above equation are negligible.

Therefore, $f(Y_0) + (X - Y_0)f'(Y_0) = 0$

$$\text{i.e., } X = Y_0 + \frac{f(Y_0)}{f'(Y_0)}$$

TABLE 5

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Negate	7	7	0	unused
NegateAlt	7	7	0	used
NegMac	5	5	0	used
AltNegMac	5	5	0	unused
NegMac32 (32 bit)	11	11	0	used

TABLE 6

Name	Comments	Program Memory	Scratch RAM	W Register
F_add	Addition	54	5	used
F_sub	Subtraction	58	5	used
F_mpy	Multiplication	114	6	used

Math Routines

Thus X is a better approximation for Y0. From the previous equation, the sequence {Xn} can be generated:

$$\text{Equation 2: } X_n = X_{n-1} - \frac{f(X_{n-1})}{f'(X_{n-1})}, \quad n > 1$$

For our case, equation 2, reduces to:

$$\text{Equation 3: } \frac{X_{n-1} + \frac{N}{X_{n-1}}}{2}$$

The routine "Sqrt" in ARITH.ASM implements the above equation. Equation 3 requires that at first an initial approximation for the root is known. The better the initial approximation, the faster the convergence rate would be. In the "Sqrt" routine, the initial approximation root is set as N/2. This routine calls the double precision division routine (D_divS).

In the code size, the Division routine (Ddiv_S) size is not included.

BCD ROUTINES

Three routines are provided for general purpose BCD arithmetic:

- BCD to binary conversion
- Binary to BCD conversion
- BCD addition

The BCD to binary conversion routine converts a 5 digit BCD code to a 16 bit binary number. The BCD addition routine adds two BCD digits directly without converting them at first to binary. Note the usage of the "DAW" instruction. The other two routines convert a binary number to a BCD code. The performance specs for the BCD routines is given in the Table 8 below:

NUMERICAL DIFFERENTIATION

This routine performs numerical differentiation of a sequence of data if the input sequence is assumed to be piecewise linear with no discontinuances (this is the

case in most of the real world signals). Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. This routine uses the so called 3 Point formula to compute the differential of a sequence of numbers.

Given an equation f(t), its derivative is given by

$$f'(t) = \frac{df(t)}{dt}$$

The above equation can be approximated using the 3-Point formula as given below:

3-Point Formula:

$$f'(t) = \frac{df(t)}{dt} = \frac{1}{2h} [f(t_0 - 2h) - 4f(t_0 - h) + 3f(t_0)]$$

where t_0 is the point at which the numerical derivative is desired and "h" is the step size. The smaller the value of the step size (h), the better the approximation. In case of say, PID motor control, the step size is proportional to the time intervals at which the new sample value of the position (or speed) is obtained. Using the above equation to compute the differential, three samples are necessary (present value and the last two past values). The subroutine "Diff" is implemented so that 1/2h factor is stored already in a RAM location (location DiffK) as 1/2h and not as "h" because it is more efficient to multiply than divide.

After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, differentiation may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Diff" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

TABLE 7

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Sqrt	22	3300 (approx.)	6	used

TABLE 8

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
BCDtoB	BCD to Binary	30	112	0	used
B2_BCD_Looped	Binary to BCD (16 bit) looped code	32	750	1	used
B2_BCD_Straight	Binary to BCD (16 bit) straight line code	44	572	1	used
BinBCD	Binary to BCD (8 bit)	10	62	0	unused
BCDAdd	BCD addition	5	5	0	used

NUMERICAL INTEGRATION

This routine performs numerical integration using Simpson's Three-Eighths Rule. This is a third order approximation for the function, whose integral is to be computed at a given point. Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. Given a function $f(t)$, it's integral over a range t_0 to t_3 is represented as:

t_3

$\int f(t)dt$. This function is approximated as follows:

t_0

Simpson's Three-Eighths Rule:

t_3

$$\int f(t)dt = \frac{3h}{8} [f(t_0) + 3f(t_1) + 3f(t_2) + f(t_3)]$$

(t_0)

The constant $3h/8$ can be computed before hand and store in a RAM location (in location IntgKLo and IntgKHi as a 16 bit number). After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, integration may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Integrate" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

PSEUDO RANDOM NUMBER GENERATOR

This routine (subroutine "Random 16" provided in ARITH.ASM) generates a pseudo random number sequence. The random points are generated using a 16 bit register and left shifting the contents with the LSB set as shown by the following schematic.

As a test, the random points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC series). The autocorrelation of the captured data is computed using a stand alone program and is shown in Figure 2. From this figure, it can be seen that the data has a strong autocorrelation only at the origin and sharply approaches to zero within a few points. This demonstrates the randomness of the data captured.

FIGURE 1

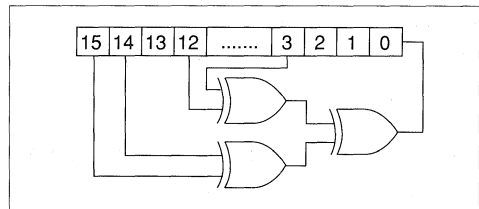


TABLE 9

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Diff	Numerical Differentiation	34	365	10	used

TABLE 10

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Integrate	Numerical Integration	39	370	12	used

Math Routines

FIGURE 2 - AUTOCORRELATION OF THE DATA POINTS GENERATED BY THE RANDOM NUMBER GENERATOR

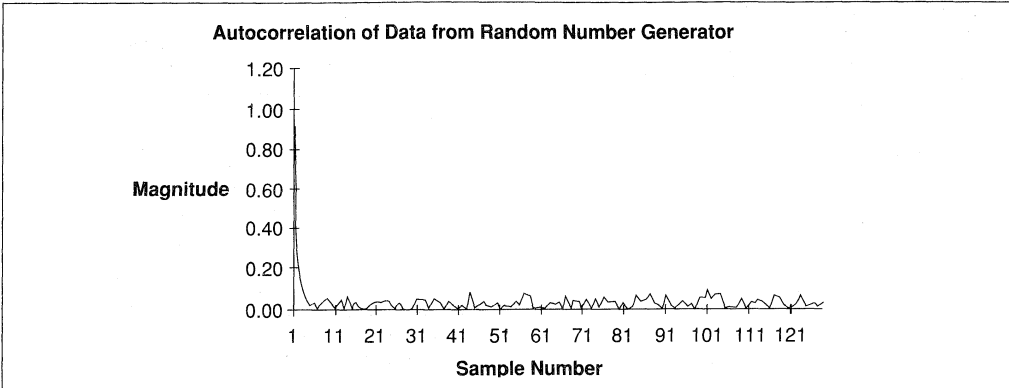


TABLE 11

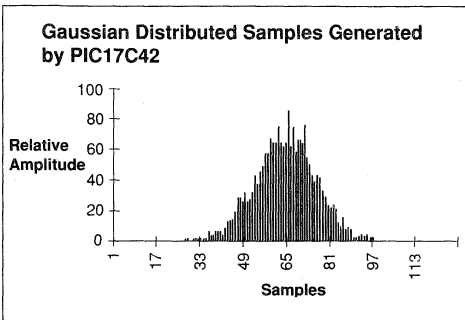
Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Random16	Pseudo Random Number Generator	12	12	0	used

TABLE 12

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Gauss	Gaussian Random Number Generator	21	452	4	used

PN (pseudo noise) sequences are widely used in digital communication systems for synchronization. These code words can also be used for data scrambling because of their good correlation properties. An interesting application of these sequences is system integrity. For example, these sequences can be regularly transmitted to a processor whose watch dog timer will time out if, say, two consecutive PN sequences do not match.

FIGURE 3 - HISTOGRAM OF THE DATA GENERATED BY THE GAUSSIAN GENERATOR



GAUSSIAN DISTRIBUTED RANDOM NUMBER GENERATOR

This routine (subroutine "Gauss" provided in ARITH.ASM) generates a sequence of random numbers with a characteristic of a normal distribution (Gaussian distributed points). This routine calls the pseudo random number generator ("random16") to obtain a near uniformly distributed random points and from these points, the Gaussian distributed points are generated. The method of generating Gaussian points is based on the "Central Limit Theorem", which states that an ensemble of average weighted sum of a sequence of uncorrelated samples tends to have a Gaussian distribution.

As a test, the Gaussian points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC series). A plot of the points captured is shown in Figure 3, which shows that the random points generated have the characteristics of a Gaussian distribution.

*Author: Amar Palacherla
Logic Products Division*

APPENDIX A: GENERAL PURPOSE MATH ROUTINES LISTING FILE OF ARITH.ASM

MPASM B0.54
General Purpose Math Routines For PIC17C42 : Ver 1.0

PAGE 1

```

0001          #define PAGE    EJECT

                TITLE    "General Purpose Math Routines For PIC17C42 : Ver 1.0"

                LIST     P=17C42, C=120, T=ON, L=0, R=DEC
;
; include "17c42.h"
;
;
;*****
; Define RAM Locations necessary For the "ARITH.ASM"
; RAM locations should be defined before calling the library math
; routines
;*****
;
0001          MODE_FAST      equ    TRUE
0000          SIGNED         equ    FALSE
;
;*****
;
if MODE_FAST

                CBLOCK    0x18
0018 0004          ACCaLo, ACCaHi, ACCbLo, ACCbHi    ; Ram Locations for Arithmetic
001C 0004          ACCcLo, ACCcHi, ACCdLo, ACCdHi    ; Routines
                ENDC

                else

                CBLOCK    0x20
                ACCaLo, ACCaHi, ACCbLo, ACCbHi
                ACCcLo, ACCcHi, ACCdLo, ACCdHi
                ENDC

                endif
;

                CBLOCK
0020 0004          tempLo, tempHi, count, sign
                ENDC

                CBLOCK
0024 0002          NumLo, NumHi
0026 0001          iterCnt
                ENDC
;

                CBLOCK                                ; RAM locations for "Diff" routine
0027 0003          XnLo, XnHi, Xn_1_Lo
002A 0003          Xn_1_Hi, Xn_2_Lo, Xn_2_Hi
002D 0002          DiffKLo, DiffKHl                ; DiffK = h = Step Size
002F 0002          DiffLo, DiffHi
                ENDC
;

                CBLOCK                                ; RAM Locations for "Integrate"
0031 0004          X0Lo, X0Hi, X1Lo, X1Hi          ; Routine
0035 0004          X2Lo, X2Hi, X3Lo, X3Hi
0039 0002          IntgKLo, IntgKHl                ; INTEGRATE CONST = 3*h/8
003B 0002          IntgLo, IntgHi

```

Math Routines

```

                                ENDC
                                ;
                                ;*****
                                ;
0018      mulcnd equ   ACCaLO
0019      mulplr equ   ACCaHI
001A      L_byte equ   ACCbLO
001B      H_byte equ   ACCbHI
                                ;
000A      _LUPCNT equ   10           ; Set Desired Number of iterations
001E      SqrtLo equ   ACCdLO       ; for Square Root Routine (NEWTON Iterations)
001F      SqrtHi equ   ACCdHI
                                ;
                                ; Define RAM locations for the Random Number Generators
                                ;
0018      RandLo equ   ACCaLO
0019      RandHi equ   ACCaHI       ; 16 bit Pseudo Random Number
001B      GaussHi equ  ACCbHI
001A      GaussLo equ  ACCbLO       ; 16 bit Gaussian distributed number
0020      GaussTmp equ  tempLo
                                ;

                                ORG   0x0000
                                ;*****
                                ;                               Math Routines Test Program
                                ;*****
                                ;
                                ; Load constant values to ACCa & ACCb for testing
                                ;
main
0000 E02D      call   loadAB          ; result of adding ACCb+ACCa->ACCb
0001 E036      call   D_add           ; Here Accb = 81FE
                                ;
0002 E02D      call   loadAB          ; result of subtracting ACCb - ACCa->ACCb
0003 E03B      call   D_sub          ; Here Accb = 7E00
                                ;
0004 E02D      call   loadAB          ; result of multiplying ACCb*ACCa-
>(ACCd, ACCc)
0005 E050      call   D_mpyS         ; Here (ACCd, ACCc) = 00FF 7E01
                                ;
0006 E02D      call   loadAB          ; result of multiplying ACCb*ACCa-
>(ACCd, ACCc)
0007 E065      call   D_mpyF         ; Here (ACCd, ACCc) = 00FF 7E01
                                ;
0008 E02D      call   loadAB          ; result of multiplying ACCb/ACCa-
>(ACCd, ACCc)
0009 E119      call   D_divS         ; Here (ACCd, ACCc) = 0040 003f
                                ;
000A E02D      call   loadAB          ; result of multiplying ACCb/ACCa-
>(ACCd, ACCc)
000B E138      call   D_divF         ; Here (ACCd, ACCc) = 0040 003f
                                ;
000C B0F3      movlw   0xf3
000D 0125      movwf  NumHi
000E B0F6      movlw   0xf6         ; Set input test number = 62454
000F 0124      movwf  NumLo        ; = F3F6h
0010 E27D      call   Sqrt          ; result = 00F9h = 249 (in SqrtLo)
                                ;                               ; exact sqrt(62454) = 249.9
                                ;
0011 B0FF      movlw   0xff
0012 0119      movwf  mulplr        ; multiplier (in mulplr) = 0FF
0013 B0FF      movlw   0xff        ; multiplicand(W.Reg) = 0FF
0014 0118      movwf  mulcnd
0015 E293      call   mpy8x8_F      ; The result 0FF*0FF = FE01 is in locations
                                ;                               ; H_byte & L_byte
0016 B0FF      movlw   0xff
0017 0119      movwf  mulplr        ; multiplier (in mulplr) = 0FF
0018 B0FF      movlw   0xff        ; multiplicand(W.Reg) = 0FF
0019 0118      movwf  mulcnd

```


Math Routines

```

001A E2B8          call    mpy8x8_S          ; The result 0FF*0FF = FE01 is in
;                                     ; H_byte & L_byte
; Test The Random Number Generators
; Capture data into trace buffer by TABLE WRITES to a
; dummy Program Memory location
;
001B B0FF          movlw   0xff
001C 010D          movwf  tblptrl
001D B05F          movlw   0x5f
001E 010E          movwf  tblptrh
;
001F B030          movlw   0x30
0020 0119          movwf  RandHi
0021 B045          movlw   0x45
0022 0118          movwf  RandLo
;
0023 C028          goto   GaussPoint
;
RandPoint
0024 E311          call   Random16
0025 A418          tlwt   _LOW,RandLo          ; only for data capture
0026 AE19          tablwt _HIGH,0,RandHi       ; using PICMASTER
0027 C024          goto   RandPoint
;
GaussPoint
0028 E31E          call   Gauss
0029 A41A          tlwt   _LOW,GaussLo        ; only for data capture
002A AE1B          tablwt _HIGH,0,GaussHi     ; using PICMASTER
002B C028          goto   GaussPoint
;
self goto self          ; End Of Test Routines
;
loadAB
002D B001          movlw   0x01
002E 0119          movwf  ACCaHI
002F B0FF          movlw   0xff          ; loads ACCa = 01FF
0030 0118          movwf  ACCaLO
;
0031 B07F          movlw   0x7f
0032 011B          movwf  ACCbHI
0033 B0FF          movlw   0xFF          ; loads ACCb = 7FFF
0034 011A          movwf  ACCbLO
0035 0002          return
;
;*****
; Double Precision Arithmetic Routines
;
; Routines : Addition, Subtraction, Multiplication ,Division
; Square Root
;
; NOTE : MODE_FAST must first be set to either
; TRUE or FALSE
;
; MODE_FAST determines the RAM address locations of ACCa thru ACCd
;
; If MODE_FAST is set TRUE, data transfers can be done efficiently
; using "MOVFP" & "MOVPF" instructions instead of indirectly moving
; at first to W Reg and then to the desired RAM locations
;
; The speed increase using this way of locating ACCa to
; ACCd will result in a saving of about 20 Cycles/filter stage
; In this case ( a 2 stage filter), it is faster by 40 Cycles
;
; If due to other constraints, ACCa thru ACCd cannot be set at
; address 0x18 to 0x1f, then the user is required to set
; MODE_FAST to FALSE
;
;*****
; Double Precision Addition

```



Math Routines

```

;
; Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_add
; (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
;
; Performance :
; Program Memory : 4 (excluding call & return)
; Clock Cycles : 4 (excluding call & return)
; W Register : Used
; Scratch RAM : 0
;
;*****;
;
D_add
0036 6018      movfp   ACCaLO,wreg
0037 0F1A      addwfp  ACCbLO      ;addwf lsb
0038 6019      movfp   ACCaHI,wreg
0039 111B      addwfc  ACCbHI      ;addwf msb with carry
003A 0002      return

;
;*****;
; Double Precision Subtraction
;
; Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_sub
; (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
;
; Performance :
; Program Memory : 4 (excluding call & return )
; Clock Cycles : 4 (excluding call & return )
; W Register : Used
; scratch RAM : 0
;
;*****;
;
D_sub
003B 6018      movfp   ACCaLO,wreg
003C 051A      subwfp  ACCbLO
003D 6019      movfp   ACCaHI,wreg
003E 031B      subwfb  ACCbHI
003F 0002      return

;
;*****;
; Function to negate a 16 bit integer
; The two 8 bit integers are assumed to be in 2 consecutive
; locations. Before calling this routine, FSR0 should be loaded with
; the address of the lower byte.
; Assume that ALUSTA register is set for no autoincrement of
; FSR0.
;*****;
;
negateAlt
0040 6000      movfp   indf0,wreg
0041 8D04      bcf     _fs1
0042 2D00      negw   indf0
0043 8504      bsf     _fs1
0044 6000      movfp   indf0,wreg
0045 2900      clr   indf0
0046 0300      subwfb indf0
0047 0002      return

;
negate
0048 1300      comf   indf0
0049 8D04      bcf     _fs1
004A 1500      incf   indf0
004B 8504      bsf     _fs1

```

Math Routines

```

004C 9A04          btfsc      _z
004D 0700          decf      indf0
004E 1300          comf     indf0
004F 0002          return

;
;*****
;                               Double Precision Multiplication
;
;                               ( Optimized for Code : Looped Code )
;
; Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_mpyS
; (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
;
; Performance :
;   Program Memory : 21 (UNSIGNED)
;                   52 (SIGNED)
;   Clock Cycles  : 242 (UNSIGNED :excluding CALL & RETURN)
;                   254 (SIGNED :excluding CALL & RETURN)
;   Scratch RAM   : 1 (used only if SIGNED arithmetic)
;
; Note : The above timing is the worst case timing, when the
;        register ACCb = FFFF. The speed may be improved if
;        the register ACCb contains a number ( out of the two
;        numbers ) with less number of 1s.
;
; Double Precision Multiply ( 16x16 -> 32 )
; ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
; in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
;*****
D_mpyS                ;results in ACCd(16 msb's) and ACCc(16
;
;
;   if SIGNED
;       CALL S_SIGN
;   endif
;
0050 2922          clrfs    count
0051 8422          bsfs    count,4          ; set count = 16
;
;   if MODE_FAST
0052 5A20          movsfp  ACCbLO,tempLo
0053 5B21          movsfp  ACCbHI,tempHi
;   else
;       movsfp  ACCbLO,wreg
;       movsfp  tempLo
;       movsfp  ACCbHI,wreg
;       movsfp  tempHi
;   endif
0054 291F          clrfs    ACCdHI
0055 291E          clrfs    ACCdLO
;
; shift right and addwf 16 times
;
mpyLoop
0056 1921          rrcfs    tempHi
0057 1920          rrcfs    tempLo
0058 9004          btfss   _carry
0059 C05E          goto    NoAdd          ; LSB is 0, so no need to addwf
005A 6018          movsfp  ACCaLO,wreg
005B 0F1E          addwfs  ACCdLO          ;addwf lsb
005C 6019          movsfp  ACCaHI,wreg
005D 111F          addwfs  ACCdHI          ;addwf msb
;   NoAdd
005E 191F          rrcfs    ACCdHI
005F 191E          rrcfs    ACCdLO

```

Math Routines

```

0060 191D          rrcf    ACCcHI
0061 191C          rrcf    ACCcLO
0062 1722          decfsz  count
0063 C056          goto    mpyLoop
;
; if SIGNED
; btffs          sign,MSB
; return
; comf    ACCcLO
; incf    ACCcLO
; btffc    _z
; decf    ACCcHI
; comf    ACCcHI
; btffc    _z
; decf    ACCdLO
; comf    ACCdLO
; btffc    _z
; decf    ACCdHI
; comf    ACCdHI
; return
; else
; return
; endif
;
; Assemble this section only if Signed Arithmetic Needed
;
; if SIGNED
;
; S_SIGN
; movfp    ACCaHI,wreg
; xorwf    ACCbHI,w
; movwf    sign          ; MSB of sign determines whether signed
; btffs    ACCbHI,MSB   ; if MSB set go & negate ACCb
; goto    chek_A
; comf    ACCbLO
; incf    ACCbLO
; btffc    _z          ; negate ACCb
; decf    ACCbHI
; comf    ACCbHI
;
; chek_A
; btffs    ACCaHI,MSB   ; if MSB set go & negate ACCa
; return
; comf    ACCaLO
; incf    ACCaLO
; btffc    _z          ; negate ACCa
; decf    ACCaHI
; comf    ACCaHI
; return
;
; endif
;
; *****
; Double Precision Multiplication
;
; ( Optimized for Speed : straight Line Code )
;
; Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
; (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
; (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
; (c) CALL D_mpy
; (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
;
; Performance :
; Program Memory : 179 (UNSIGNED)
;                 204 (SIGNED)
; Clock Cycles   : 176 (UNSIGNED :excluding CALL & RETURN)
;                 183 (SIGNED :excluding CALL & RETURN)
;

```

Math Routines

```

; Note : The above timing is the worst case timing, when the
; register ACCb = FFFF. The speed may be improved if
; the register ACCb contains a number ( out of the two
; numbers ) with less number of 1s.
;
; The performance specs are for Unsigned arithmetic ( i.e,
; with "SIGNED equ FALSE ").
;
; Upon return from subroutine, the input registers
;
;
;*****
; Multiplication Macro
;*****
mulMac MACRO
    variable i

    i = 0

    if SIGNED
        .while i < 15
    else
        .while i < 16
    endif
    .if i < 8
        btfs    ACCbLO,i        ; test low byte
    .else
        btfs    ACCbHI,i-8      ; test high byte
    .fi
    goto    NoAdd#v(i)        ; LSB is 0, so no need to addwf
    movfp   ACCaLO,wreg
    addwf   ACCdLO            ;addwf lsb
    movfp   ACCaHI,wreg
    addwfc  ACCdHI            ;addwf msb
NoAdd#v(i)
    rrcf    ACCdHI
    rrcf    ACCdLO
    rrcf    ACCcHI
    rrcf    ACCcLO
    bcf     _carry
    i = i+1
        .endw
    if SIGNED
        rrcf    ACCdHI
        rrcf    ACCdLO
        rrcf    ACCcHI
        rrcf    ACCcLO
        bcf     _carry
    endif
;
; ENDM
;
;*****
; Double Precision Negate Macros
;*****
AltNegMac MACRO fileRegLo,fileRegHi
    movfp   fileRegLo,wreg
    negw    fileRegLo
    movfp   fileRegHi,wreg
    clrf   fileRegHi
    subwfb  fileRegHi
    ENDM

;
negMac MACRO fileRegLo, fileRegHi
    comf   fileRegLo        ; negate FileReg ( -FileReg -> FileReg )
    incf   fileRegLo
    btfsc  _z

```

Math Routines

```

    decf    fileRegHi
    comf   fileRegHi
    ENDM
;
NegMac32      MACRO   x3,x2,x1,x0
    movfp   x3,wreg
    negw    x3
    movfp   x2,wreg
    clrf   x2
    subwfb x2
    movfp   x1,wreg
    clrf   x1
    subwfb x1
    movfp   x0,wreg
    clrf   x0
    subwfb x0
    ENDM
;
;*****;
;           Double Precision Multiply ( 16x16 -> 32 )
;           ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
;           in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
;
D_mpyF                                ;results in ACCd(16 msb's) and ACCc(16
;
;   if   SIGNED
;
;       movfp   ACCaHI,wreg
;       xorwf   ACCbHI,w
;       movwf   sign
;       btfss   ACCbHI,MSB           ; if MSB set go & negate ACCb
;       goto    chek_A_MSB_MPY
;
;       negMac  ACCbLO,ACCbHI
;
;       chek_A_MSB_MPY
;       btfss   ACCaHI,MSB           ; if MSB set go & negate ACCa
;       goto    continue_MPY
;       negMac  ACCaLO,ACCaHI
;
;   endif
;
;   continue_MPY
0065 291F      clrf   ACCdHI
0066 291E      clrf   ACCdLO
0067 8804      bcf    _carry
;
; use the mulMac macro 16 times
;
mulMac
0000          variable i
0000          i = 0

;   if SIGNED
;       .while i < 15
;
;   else
;       .while i < 16
;   endif
;   .if i < 8
;       btfss   ACCbLO,i           ; test low byte
;   .else
;       btfss   ACCbHI,i-8       ; test high byte
;   .fi
;       goto    NoAdd#v(i)       ; LSB is 0, so no need to addw
;       movfp   ACCaLO,wreg

```

Math Routines

```

                                addwf   ACCdLO           ;addwf lsb
                                movfp   ACCaHI,wreg
                                addwfc  ACCdHI           ;addwf msb
NoAdd#v(i)
                                rrcf    ACCdHI
                                rrcf    ACCdLO
                                rrcf    ACCcHI
                                rrcf    ACCcLO
                                bcf     _carry
                                i = i+1
                                .endw

                                .if i < 8
0068 901A                       btfss   ACCbLO,i       ; test low byte
                                .else
                                btfss   ACCbHI,i-8       ; test high byte
                                .fi

0069 C06E                       goto    NoAdd0         ; LSB is 0, so no need to addwf
006A 6018                       movfp   ACCaLO,wreg
006B 0F1E                       addwf   ACCdLO         ;addwf lsb
006C 6019                       movfp   ACCaHI,wreg
006D 111F                       addwfc  ACCdHI         ;addwf msb
                                NoAdd0

006E 191F                       rrcf    ACCdHI
006F 191E                       rrcf    ACCdLO
0070 191D                       rrcf    ACCcHI
0071 191C                       rrcf    ACCcLO
0072 8804                       bcf     _carry
0001                               i = i+1

                                .if i < 8
0073 911A                       btfss   ACCbLO,i       ; test low byte
                                .else
                                btfss   ACCbHI,i-8       ; test high byte
                                .fi

0074 C079                       goto    NoAdd1         ; LSB is 0, so no need to addwf
0075 6018                       movfp   ACCaLO,wreg
0076 0F1E                       addwf   ACCdLO         ;addwf lsb
0077 6019                       movfp   ACCaHI,wreg
0078 111F                       addwfc  ACCdHI         ;addwf msb
                                NoAdd1

0079 191F                       rrcf    ACCdHI
007A 191E                       rrcf    ACCdLO
007B 191D                       rrcf    ACCcHI
007C 191C                       rrcf    ACCcLO
007D 8804                       bcf     _carry
0002                               i = i+1

                                .if i < 8
007E 921A                       btfss   ACCbLO,i       ; test low byte
                                .else
                                btfss   ACCbHI,i-8       ; test high byte
                                .fi

007F C084                       goto    NoAdd2         ; LSB is 0, so no need to addwf
0080 6018                       movfp   ACCaLO,wreg
0081 0F1E                       addwf   ACCdLO         ;addwf lsb
0082 6019                       movfp   ACCaHI,wreg
0083 111F                       addwfc  ACCdHI         ;addwf msb
                                NoAdd2

0084 191F                       rrcf    ACCdHI
0085 191E                       rrcf    ACCdLO
0086 191D                       rrcf    ACCcHI
0087 191C                       rrcf    ACCcLO

```

Math Routines

```

0088 8804          bcf      _carry
0003              i = i+1

                                .if i < 8
0089 931A          btfss   ACCbLO,i      ; test low byte
                                .else
                                btfss   ACCbHI,i-8    ; test high byte
                                .fi

008A C08F          goto    NoAdd3          ; LSB is 0, so no need to addwf
008B 6018          movf    ACCaLO,wreg
008C 0F1E          addwf   ACCdLO          ;addwf lsb
008D 6019          movf    ACCaHI,wreg
008E 111F          addwfc  ACCdHI          ;addwf msb

                                NoAdd3
008F 191F          rrcf    ACCdHI
0090 191E          rrcf    ACCdLO
0091 191D          rrcf    ACCcHI
0092 191C          rrcf    ACCcLO
0093 8804          bcf      _carry
0004              i = i+1

                                .if i < 8
0094 941A          btfss   ACCbLO,i      ; test low byte
                                .else
                                btfss   ACCbHI,i-8    ; test high byte
                                .fi

0095 C09A          goto    NoAdd4          ; LSB is 0, so no need to addwf
0096 6018          movf    ACCaLO,wreg
0097 0F1E          addwf   ACCdLO          ;addwf lsb
0098 6019          movf    ACCaHI,wreg
0099 111F          addwfc  ACCdHI          ;addwf msb

                                NoAdd4
009A 191F          rrcf    ACCdHI
009B 191E          rrcf    ACCdLO
009C 191D          rrcf    ACCcHI
009D 191C          rrcf    ACCcLO
009E 8804          bcf      _carry
0005              i = i+1

                                .if i < 8
009F 951A          btfss   ACCbLO,i      ; test low byte
                                .else
                                btfss   ACCbHI,i-8    ; test high byte
                                .fi

00A0 C0A5          goto    NoAdd5          ; LSB is 0, so no need to addwf
00A1 6018          movf    ACCaLO,wreg
00A2 0F1E          addwf   ACCdLO          ;addwf lsb
00A3 6019          movf    ACCaHI,wreg
00A4 111F          addwfc  ACCdHI          ;addwf msb

                                NoAdd5
00A5 191F          rrcf    ACCdHI
00A6 191E          rrcf    ACCdLO
00A7 191D          rrcf    ACCcHI
00A8 191C          rrcf    ACCcLO
00A9 8804          bcf      _carry
0006              i = i+1

                                .if i < 8
00AA 961A          btfss   ACCbLO,i      ; test low byte
                                .else
                                btfss   ACCbHI,i-8    ; test high byte
                                .fi

```


Math Routines

```

00AB C0B0          goto      NoAdd6          ; LSB is 0, so no need to addwf
00AC 6018          movfp    ACCaLO,wreg
00AD 0F1E          addwf   ACCdLO          ;addwf lsb
00AE 6019          movfp    ACCaHI,wreg
00AF 111F          addwfc  ACCdHI          ;addwf msb
                NoAdd6

00B0 191F          rrcf   ACCdHI
00B1 191E          rrcf   ACCdLO
00B2 191D          rrcf   ACCcHI
00B3 191C          rrcf   ACCcLO
00B4 8804          bcf    _carry
0007              i = i+1

                .if i < 8
00B5 971A          btfss  ACCbLO,i        ; test low byte
                .else
                btfss  ACCbHI,i-8      ; test high byte
                .fi

00B6 C0BB          goto      NoAdd7          ; LSB is 0, so no need to addwf
00B7 6018          movfp    ACCaLO,wreg
00B8 0F1E          addwf   ACCdLO          ;addwf lsb
00B9 6019          movfp    ACCaHI,wreg
00BA 111F          addwfc  ACCdHI          ;addwf msb
                NoAdd7

00BB 191F          rrcf   ACCdHI
00BC 191E          rrcf   ACCdLO
00BD 191D          rrcf   ACCcHI
00BE 191C          rrcf   ACCcLO
00BF 8804          bcf    _carry
0008              i = i+1

                .if i < 8
                btfss  ACCbLO,i        ; test low byte
                .else
                btfss  ACCbHI,i-8      ; test high byte
                .fi

00C0 901B          .fi

00C1 C0C6          goto      NoAdd8          ; LSB is 0, so no need to addwf
00C2 6018          movfp    ACCaLO,wreg
00C3 0F1E          addwf   ACCdLO          ;addwf lsb
00C4 6019          movfp    ACCaHI,wreg
00C5 111F          addwfc  ACCdHI          ;addwf msb
                NoAdd8

00C6 191F          rrcf   ACCdHI
00C7 191E          rrcf   ACCdLO
00C8 191D          rrcf   ACCcHI
00C9 191C          rrcf   ACCcLO
00CA 8804          bcf    _carry
0009              i = i+1

                .if i < 8
                btfss  ACCbLO,i        ; test low byte
                .else
                btfss  ACCbHI,i-8      ; test high byte
                .fi

00CB 911B          .fi

00CC C0D1          goto      NoAdd9          ; LSB is 0, so no need to addwf
00CD 6018          movfp    ACCaLO,wreg
00CE 0F1E          addwf   ACCdLO          ;addwf lsb
00CF 6019          movfp    ACCaHI,wreg
00D0 111F          addwfc  ACCdHI          ;addwf msb
                NoAdd9

00D1 191F          rrcf   ACCdHI
00D2 191E          rrcf   ACCdLO

```

Math Routines

```

00D3 191D                rrcf    ACCcHI
00D4 191C                rrcf    ACCcLO
00D5 8804                bcf     _carry
00DA                    i = i+1

                        .if i < 8
                        btfss   ACCbLO,i      ; test low byte

                        .else

00D6 921B                btfss   ACCbHI,i-8      ; test high byte
                        .fi

00D7 C0DC                goto    NoAdd10         ; LSB is 0, so no need to addwf
00D8 6018                movfp  ACCaLO,wreg
00D9 0F1E                addwf  ACCgLO           ;addwf lsb
00DA 6019                movfp  ACCaHI,wreg
00DB 111F                addwfc ACCdHI           ;addwf msb
                        NoAdd10

00DC 191F                rrcf    ACCdHI
00DD 191E                rrcf    ACCdLO
00DE 191D                rrcf    ACCcHI
00DF 191C                rrcf    ACCcLO
00E0 8804                bcf     _carry
000B                    i = i+1

                        .if i < 8
                        btfss   ACCbLO,i      ; test low byte

                        .else

00E1 931B                btfss   ACCbHI,i-8      ; test high byte
                        .fi

00E2 C0E7                goto    NoAdd11         ; LSB is 0, so no need to addwf
00E3 6018                movfp  ACCaLO,wreg
00E4 0F1E                addwf  ACCdLO           ;addwf lsb
00E5 6019                movfp  ACCaHI,wreg
00E6 111F                addwfc ACCdHI           ;addwf msb
                        NoAdd11

00E7 191F                rrcf    ACCdHI
00E8 191E                rrcf    ACCdLO
00E9 191D                rrcf    ACCcHI
00EA 191C                rrcf    ACCcLO
00EB 8804                bcf     carry
000C                    i = i+1

                        .if i < 8
                        btfss   ACCbLO,i      ; test low byte

                        .else

00EC 941B                btfss   ACCbHI,i-8      ; test high byte
                        .fi

00ED C0F2                goto    NoAdd12         ; LSB is 0, so no need to addwf
00EE 6018                movfp  ACCaLO,wreg
00EF 0F1E                addwf  ACCdLO           ;addwf lsb
00F0 6019                movfp  ACCaHI,wreg
00F1 111F                addwfc ACCdHI           ;addwf msb
                        NoAdd12

00F2 191F                rrcf    ACCdHI
00F3 191E                rrcf    ACCdLO
00F4 191D                rrcf    ACCcHI
00F5 191C                rrcf    ACCcLO
00F6 8804                bcf     _carry
000D                    i = i+1

                        .if i < 8
                        btfss   ACCbLO,i      ; test low byte

                        .else

```

Math Routines

```

00F7 951B                btfss    ACCbHI,i-8    ; test high byte
                        .fi
00F8 C0FD                goto     NoAdd13       ; LSB is 0, so no need to addwf
00F9 6018                movfp   ACCaLO,wreg
00FA 0F1E                addwf   ACCdLO         ;addwf lsb
00FB 6019                movfp   ACCaHI,wreg
00FC 111F                addwfc  ACCdHI         ;addwf msb
                        NoAdd13
00FD 191F                rrcf    ACCdHI
00FE 191E                rrcf    ACCdLO
00FF 191D                rrcf    ACCcHI
0100 191C                rrcf    ACCcLO
0101 8804                bcf     _carry
000E                    i = i+1

                        .if i < 8
                        btfss    ACCbLO,i    ; test low byte
                        .else
0102 961B                btfss    ACCbHI,i-8    ; test high byte
                        .fi
0103 C108                goto     NoAdd14       ; LSB is 0, so no need to addwf
0104 6018                movfp   ACCaLO,wreg
0105 0F1E                addwf   ACCdLO         ;addwf lsb
0106 6019                movfp   ACCaHI,wreg
0107 111F                addwfc  ACCdHI         ;addwf msb
                        NoAdd14
0108 191F                rrcf    ACCdHI
0109 191E                rrcf    ACCdLO
010A 191D                rrcf    ACCcHI
010B 191C                rrcf    ACCcLO
010C 8804                bcf     _carry
000F                    i = i+1

                        .if i < 8
                        btfss    ACCbLO,i    ; test low byte
                        .else
010D 971B                btfss    ACCbHI,i-8    ; test high byte
                        .fi
010E C113                goto     NoAdd15       ; LSB is 0, so no need to addwf
010F 6018                movfp   ACCaLO,wreg
0110 0F1E                addwf   ACCdLO         ;addwf lsb
0111 6019                movfp   ACCaHI,wreg
0112 111F                addwfc  ACCdHI         ;addwf msb
                        NoAdd15
0113 191F                rrcf    ACCdHI
0114 191E                rrcf    ACCdLO
0115 191D                rrcf    ACCcHI
0116 191C                rrcf    ACCcLO
0117 8804                bcf     _carry
0010                    i = i+1

                        if SIGNED
                        rrcf    ACCdHI
                        rrcf    ACCdLO
                        rrcf    ACCcHI
                        rrcf    ACCcLO
                        bcf     _carry
                        endif

```

Math Routines

```

;
;
; if SIGNED
;   btfs    sign,MSB      ; negate (ACCC,ACCD)
;   return
;   NegMac32 ACCcHI,ACCcLO,ACCdHI, ACCdLO
;   return
; else
;   return
; endif

;
;*****
;                               Double Precision Division
;
;                               ( Optimized for Code : Looped Code )
;
;*****
;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
;                               Remainder in ACCc (16 bits)
;   (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
;   (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
;   (c) CALL D_div
;   (d) The 16 bit result is in location ACCbHI & ACCbLO
;   (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;
; Performance :
;   Program Memory :   31 (UNSIGNED)
;                   39 (SIGNED)
;   Clock Cycles   :  300 (UNSIGNED : excluding CALL & RETURN)
;                   312 (SIGNED : excluding CALL & RETURN)
;
; NOTE :
;   The performance specs are for Unsigned arithmetic ( i.e.,
;   with "SIGNED equ FALSE ").
;
;*****
;   Double Precision Divide ( 16/16 -> 16 )
;
;   ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
; with Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
(ACCcHI,ACCcLO) .
;
;   B/A = (Q) + (R)/A
; or   B = A*Q + R
;
;   where  B : Numerator
;          A : Denominator
;          Q : Quotient (Integer Result)
;          R : Remainder
;
; Note : Check for ZERO Denominator or Numerator is not performed
;        A ZERO Denominator will produce incorrect results
;
; SIGNED Arithmetic :
;   In case of signed arithmetic, if either
; numerator or denominator is negative, then both Q & R are
; represented as negative numbers
;   -(B/A) = -(Q) + (-R)/A
; or   -B = (-Q)*A + (-R)
;*****
;
; D_divS
;
0119 8404      bsf     _fs0
011A 8504      bsf     _fs1      ; set no auto-incrmt for fsr0

if    SIGNED

```

Math Routines

```

                CALL    S_SIGN
            endif
;
011B 2922        clrf    count
011C 8422        bsf    count,4        ; set count = 16
011D 291D        clrf    ACCcHI
011E 291C        clrf    ACCcLO
011F 291E        clrf    ACCdLO
0120 291F        clrf    ACCdHI
;
; Looped code
;
divLoop
0121 8804        bcf    _carry
0122 1B1A        rlc    ACCbLO
0123 1B1B        rlc    ACCbHI
0124 1B1C        rlc    ACCcLO
0125 1B1D        rlc    ACCcHI
0126 6019        movfp  ACCaHI,wreg
0127 041D        subwf  ACCcHI,w        ;check if a>c
0128 9204        btfss  _z
0129 C12C        goto   notz
012A 6018        movfp  ACCaLO,wreg
012B 041C        subwf  ACCcLO,w        ; if msb equal then check lsb
notz
012C 9004        btfss  _carry        ; carry set if c>a
012D C133        goto   nosub        ; if c < a
subca
012E 6018        movfp  ACCaLO,wreg        ; c-a into c
012F 051C        subwf  ACCcLO
0130 6019        movfp  ACCaHI,wreg
0131 031D        subwfb ACCcHI
0132 8004        bsf    _carry        ;shift a 1 into d (result)
nosub
0133 1B1E        rlc    ACCdLO
0134 1B1F        rlc    ACCdHI
0135 1722        decfsz count
0136 C121        goto   divLoop
;
; if SIGNED
; btfss    sign,MSB
; return
; movlw   ACCcLO
; movwf   fsr0
; call   negate
; movlw   ACCdLO
; movwf   fsr0
; call   negate
; return
else
0137 0002        return
endif
;
;*****
; Double Precision Division
;
; ( Optimized for Speed : straight Line Code )
;
;*****
; Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
; Remainder in ACCc (16 bits)
;
; (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
; (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
; (c) CALL D_div
; (d) The 16_bit result is in location ACCbHI & ACCbLO
; (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;
;

```

Math Routines

```

;          B/A = (Q) + (R)/A
;      or   B = A*Q + R
;
;          where  B :   Numerator
;                A :   Denominator
;                Q :   Quotient (Integer Result)
;                R :   Remainder
;
;      Note : Check for ZERO Denominator or Numerator is not performed
;            A ZERO Denominator will produce incorrect results
;
;      SIGNED Arithmetic :
;            In case of signed arithmetic, if either
;      numerator or denominator is negative, then both Q & R are
;      represented as negative numbers
;            -(B/A) = -(Q) + (-R)/A
;      or      -B = (-Q)*A + (-R)
;
;      Performance :
;            Program Memory   : 325 (UNSIGNED)
;                               354 (SIGNED)
;            Clock Cycles    : 250 (UNSIGNED : excluding CALL & RETURN)
;                               260 (SIGNED  : excluding CALL & RETURN)
;
;*****;
;      division macro
;
divMac MACRO
variable i

    i = 0
    .while i < 16
;
        bcf     _carry
        rlcfc  ACCbLO
        rlcfc  ACCbHI
        rlcfc  ACCcLO
        rlcfc  ACCcHI
        movfp  ACCaHI,wreg
        subwf  ACCcHI,w      ;check if a>c
        btfss  _z
        goto  notz#v(i)
        movfp  ACCaLO,wreg
        subwf  ACCcLO,w      ;if msb equal then check lsb
notz#v(i)  btfss  _carry      ;carry set if c>a
        goto  nosub#v(i)    ; if c < a
subca#v(i) movfp  ACCaLO,wreg ;c-a into c
        subwf  ACCcLO
        movfp  ACCaHI,wreg
        subwfb ACCcHI
        bsf    _carry      ;shift a 1 into d (result)
nosub#v(i) rlcfc  ACCdLO
        rlcfc  ACCdHI
        i=i+1
    .endw
;
    ENDM
;
;*****;
;      Double Precision Divide ( 16/16 -> 16 )
;
;      ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
;      with Quotient in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
(ACCcHI,ACCcLO).
;
;      NOTE : Before calling this routine, the user should make sure that
;            the Numerator(ACCb) is greater than Denominator(ACCa). If
;            the case is not true, the user should scale either Numerator
;            or Denominator or both such that Numerator is greater than

```

```

;           the Denominator.
;
;
;*****
;
D_divF
;
;   if   SIGNED
;       movfp   ACCaHI,wreg
;       xorwf   ACCbHI,w
;       movwf   sign
;       btfs   ACCbHI,MSB           ; if MSB set go & negate ACCb
;       goto   chek_A_MSB_DIV
;
;       negMac  ACCbLO,ACCbHI
;
; chek_A_MSB_DIV
;       btfs   ACCaHI,MSB           ; if MSB set go & negate ACCa
;       goto   continue_DIV
;       negMac  ACCaLO,ACCaHI
;
;   endif
;
; continue_DIV
0138 291D      clrf   ACCcHI
0139 291C      clrf   ACCcLO
013A 291E      clrf   ACCdLO
013B 291F      clrf   ACCdHI

;
; straight line code : using the macro divMac
;
;   divMac
0000          variable i
0000          i = 0
;             .while i < 16
;
;             bcf   _carry
;             rlc   ACCbLO
;             rlc   ACCbHI
;             rlc   ACCcLO
;             rlc   ACCcHI
;             movfp ACCaHI,wreg
;             subwf ACCcHI,w           ;check if a>c
;             btfs  _z
;             goto  notz#v(i)
;             movfp ACCaLO,wreg
;             subwf ACCcLO,w           ;if msb equal then check lsb
notz#v(i)     btfs  _carry             ;carry set if c>a
;             goto  nosub#v(i)        ; if c < a
;             movfp ACCaLO,wreg       ;c-a into c
subca#v(i)   subwf  ACCcLO
;             movfp ACCaHI,wreg
;             subwfb ACCcHI
;             bsf   _carry           ;shift a 1 into d (result)
nosub#v(i)   rlc   ACCdLO
;             rlc   ACCdHI
;             i=i+1
;             .endw
;
013C 8804      bcf   _carry
013D 1B1A      rlc   ACCbLO
013E 1B1B      rlc   ACCbHI
013F 1B1C      rlc   ACCcLO
0140 1B1D      rlc   ACCcHI
0141 6019      movfp ACCaHI,wreg
0142 041D      subwf  ACCcHI,w           ;check if a>c

```

Math Routines

```

0143 9204                btfss    _z
0144 C147                goto     notz0
0145 6018                movfp   ACCaLO,wreg
0146 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
0147 9004                notz0   btfss    _carry   ;carry set if c>a
0148 C14E                goto     nosub0      ; if c < a
0149 6018                subca0  movfp   ACCaLO,wreg ;c-a into c
014A 051C                subwf   ACCcLO
014B 6019                movfp   ACCaHI,wreg
014C 031D                subwfb  ACCcHI
014D 8004                bsf     _carry       ;shift a 1 into d (result)
014E 1B1E                nosub0  rlcfc   ACCdLO
014F 1B1F                rlcfc   ACCdHI
0001                    i = i+1

;

0150 8804                bcf     _carry
0151 1B1A                rlcfc   ACCbLO
0152 1B1B                rlcfc   ACCbHI
0153 1B1C                rlcfc   ACCcLO
0154 1B1D                rlcfc   ACCcHI
0155 6019                movfp   ACCaHI,wreg
0156 041D                subwf   ACCcHI,w     ;check if a>c
0157 9204                btfss    _z
0158 C15B                goto     notz1
0159 6018                movfp   ACCaLO,wreg
015A 041C                subwf   ACCcLO,w     ;if msb equal then check lsb
015B 9004                notz1   btfss    _carry   ;carry set if c>a
015C C162                goto     nosub1     ; if c < a
015D 6018                subca1  movfp   ACCaLO,wreg ;c-a into c
015E 051C                subwf   ACCcLO
015F 6019                movfp   ACCaHI,wreg
0160 031D                subwfb  ACCcHI
0161 8004                bsf     _carry       ;shift a 1 into d (result)
0162 1B1E                nosub1  rlcfc   ACCdLO
0163 1B1F                rlcfc   ACCdHI
0002                    i = i+1

;

0164 8804                bcf     _carry
0165 1B1A                rlcfc   ACCbLO
0166 1B1B                rlcfc   ACCbHI
0167 1B1C                rlcfc   ACCcLO
0168 1B1D                rlcfc   ACCcHI
0169 6019                movfp   ACCaHI,wreg
016A 041D                subwf   ACCcHI,w     ;check if a>c
016B 9204                btfss    _z
016C C16F                goto     notz2
016D 6018                movfp   ACCaLO,wreg
016E 041C                subwf   ACCcLO,w     ;if msb equal then check lsb
016F 9004                notz2   btfss    _carry   ;carry set if c>a
0170 C176                goto     nosub2     ; if c < a
0171 6018                subca2  movfp   ACCaLO,wreg ;c-a into c
0172 051C                subwf   ACCcLO
0173 6019                movfp   ACCaHI,wreg
0174 031D                subwfb  ACCcHI
0175 8004                bsf     _carry       ;shift a 1 into d (result)
0176 1B1E                nosub2  rlcfc   ACCdLO
0177 1B1F                rlcfc   ACCdHI
0003                    i = i+1

;

0178 8804                bcf     _carry
0179 1B1A                rlcfc   ACCbLO
017A 1B1B                rlcfc   ACCbHI
017B 1B1C                rlcfc   ACCcLO
017C 1B1D                rlcfc   ACCcHI
017D 6019                movfp   ACCaHI,wreg
017E 041D                subwf   ACCcHI,w     ;check if a>c

```


Math Routines

```

017F 9204                btfss    _z
0180 C183                goto     notz3
0181 6018                movfp   ACCaLO,wreg
0182 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
0183 9004                notz3    btfss    _carry  ;carry set if c>a
0184 C18A                goto     nosub3      ; if c < a
0185 6018                subca3  movfp   ACCaLO,wreg ;c-a into c
0186 051C                subwf   ACCcLO
0187 6019                movfp   ACCaHI,wreg
0188 031D                subwfb  ACCcHI
0189 8004                bsf     _carry
018A 1B1E                nosub3  rlcfc   ACCdLO      ;shift a 1 into d (result)
018B 1B1F                rlcfc   ACCdHI
0004                i = i+1

```

```

;
018C 8804                bcf     _carry
018D 1B1A                rlcfc   ACCbLO
018E 1B1B                rlcfc   ACCbHI
018F 1B1C                rlcfc   ACCcLO
0190 1B1D                rlcfc   ACCcHI
0191 6019                movfp   ACCaHI,wreg
0192 041D                subwf   ACCcHI,w      ;check if a>c
0193 9204                btfss    _z
0194 C197                goto     notz4
0195 6018                movfp   ACCaLO,wreg
0196 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
0197 9004                notz4    btfss    _carry  ;carry set if c>a
0198 C19E                goto     nosub4      ; if c < a
0199 6018                subca4  movfp   ACCaLO,wreg ;c-a into c
019A 051C                subwf   ACCcLO
019B 6019                movfp   ACCaHI,wreg
019C 031D                subwfb  ACCcHI
019D 8004                bsf     _carry
019E 1B1E                nosub4  rlcfc   ACCdLO      ;shift a 1 into d (result)
019F 1B1F                rlcfc   ACCdHI
0005                i = i+1

```

```

;
01A0 8804                bcf     _carry
01A1 1B1A                rlcfc   ACCbLO
01A2 1B1B                rlcfc   ACCbHI
01A3 1B1C                rlcfc   ACCcLO
01A4 1B1D                rlcfc   ACCcHI
01A5 6019                movfp   ACCaHI,wreg
01A6 041D                subwf   ACCcHI,w      ;check if a>c
01A7 9204                btfss    _z
01A8 C1AB                goto     notz5
01A9 6018                movfp   ACCaLO,wreg
01AA 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
01AB 9004                notz5    btfss    _carry  ;carry set if c>a
01AC C1B2                goto     nosub5      ; if c < a
01AD 6018                subca5  movfp   ACCaLO,wreg ;c-a into c
01AE 051C                subwf   ACCcLO
01AF 6019                movfp   ACCaHI,wreg
01B0 031D                subwfb  ACCcHI
01B1 8004                bsf     _carry
01B2 1B1E                nosub5  rlcfc   ACCdLO      ;shift a 1 into d (result)
01B3 1B1F                rlcfc   ACCdHI
0006                i = i+1

```

```

;
01B4 8804                bcf     _carry
01B5 1B1A                rlcfc   ACCbLO
01B6 1B1B                rlcfc   ACCbHI
01B7 1B1C                rlcfc   ACCcLO
01B8 1B1D                rlcfc   ACCcHI
01B9 6019                movfp   ACCaHI,wreg
01BA 041D                subwf   ACCcHI,w      ;check if a>c

```



Math Routines

```

01BB 9204                btfss    _z
01BC C1BF                goto     notz6
01BD 6018                movfp   ACCaLO,wreg
01BE 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
01BF 9004                notz6    btfss    _carry    ;carry set if c>a
01C0 C1C6                goto     nosub6      ; if c < a
01C1 6018                subca6  movfp   ACCaLO,wreg ;c-a into c
01C2 051C                subwf   ACCcLO
01C3 6019                movfp   ACCaHI,wreg
01C4 031D                subwfb  ACCcHI
01C5 8004                bsf     _carry        ;shift a 1 into d (result)
01C6 1B1E                nosub6  rlcfc   ACCdLO
01C7 1B1F                rlcfc   ACCdHI
0007                    i = i+1

;

01C8 8804                bcf     _carry
01C9 1B1A                rlcfc   ACCbLO
01CA 1B1B                rlcfc   ACCbHI
01CB 1B1C                rlcfc   ACCcLO
01CC 1B1D                rlcfc   ACCcHI
01CD 6019                movfp   ACCaHI,wreg
01CE 041D                subwf   ACCcHI,w      ;check if a>c
01CF 9204                btfss    _z
01D0 C1D3                goto     notz7
01D1 6018                movfp   ACCaLO,wreg
01D2 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
01D3 9004                notz7    btfss    _carry    ;carry set if c>a
01D4 C1DA                goto     nosub7      ; if c < a
01D5 6018                subca7  movfp   ACCaLO,wreg ;c-a into c
01D6 051C                subwf   ACCcLO
01D7 6019                movfp   ACCaHI,wreg
01D8 031D                subwfb  ACCcHI
01D9 8004                bsf     _carry        ;shift a 1 into d (result)
01DA 1B1E                nosub7  rlcfc   ACCdLO
01DB 1B1F                rlcfc   ACCdHI
0008                    i = i+1

;

01DC 8804                bcf     _carry
01DD 1B1A                rlcfc   ACCbLO
01DE 1B1B                rlcfc   ACCbHI
01DF 1B1C                rlcfc   ACCcLO
01E0 1B1D                rlcfc   ACCcHI
01E1 6019                movfp   ACCaHI,wreg
01E2 041D                subwf   ACCcHI,w      ;check if a>c
01E3 9204                btfss    _z
01E4 C1E7                goto     notz8
01E5 6018                movfp   ACCaLO,wreg
01E6 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
01E7 9004                notz8    btfss    _carry    ;carry set if c>a
01E8 C1EE                goto     nosub8      ; if c < a
01E9 6018                subca8  movfp   ACCaLO,wreg ;c-a into c
01EA 051C                subwf   ACCcLO
01EB 6019                movfp   ACCaHI,wreg
01EC 031D                subwfb  ACCcHI
01ED 8004                bsf     _carry        ;shift a 1 into d (result)
01EE 1B1E                nosub8  rlcfc   ACCdLO
01EF 1B1F                rlcfc   ACCdHI
0009                    i = i+1

;

01F0 8804                bcf     _carry
01F1 1B1A                rlcfc   ACCbLO
01F2 1B1B                rlcfc   ACCbHI
01F3 1B1C                rlcfc   ACCcLO
01F4 1B1D                rlcfc   ACCcHI
01F5 6019                movfp   ACCaHI,wreg
01F6 041D                subwf   ACCcHI,w      ;check if a>c

```

Math Routines

```

01F7 9204                btfss    _z
01F8 C1FB                goto     notz9
01F9 6018                movfp   ACCaLO,wreg
01FA 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
01FB 9004                notz9    btfss    _carry    ;carry set if c>a
01FC C202                goto     nosub9      ; if c < a
01FD 6018                subca9  movfp   ACCaLO,wreg ;c-a into c
01FE 051C                subwf   ACCcLO
01FF 6019                movfp   ACCaHI,wreg
0200 031D                subwfb  ACCcHI
0201 8004                bsf     _carry        ;shift a 1 into d (result)
0202 1B1E                nosub9  rlcfc  ACCdLO
0203 1B1F                rlcfc  ACCdHI
000A                    i = i+1

;

0204 8804                bcf     _carry
0205 1B1A                rlcfc  ACCbLO
0206 1B1B                rlcfc  ACCbHI
0207 1B1C                rlcfc  ACCcLO
0208 1B1D                rlcfc  ACCcHI
0209 6019                movfp   ACCaHI,wreg
020A 041D                subwf   ACCcHI,w      ;check if a>c
020B 9204                btfss    _z
020C C20F                goto     notz10
020D 6018                movfp   ACCaLO,wreg
020E 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
020F 9004                notz10  btfss    _carry    ;carry set if c>a
0210 C216                goto     nosub10     ; if c < a
0211 6018                subca10 movfp   ACCaLO,wreg ;c-a into c
0212 051C                subwf   ACCcLO
0213 6019                movfp   ACCaHI,wreg
0214 031D                subwfb  ACCcHI
0215 8004                bsf     _carry        ;shift a 1 into d (result)
0216 1B1E                nosub10 rlcfc  ACCdLO
0217 1B1F                rlcfc  ACCdHI
000B                    i = i+1

;

0218 8804                bcf     _carry
0219 1B1A                rlcfc  ACCbLO
021A 1B1B                rlcfc  ACCbHI
021B 1B1C                rlcfc  ACCcLO
021C 1B1D                rlcfc  ACCcHI
021D 6019                movfp   ACCaHI,wreg
021E 041D                subwf   ACCcHI,w      ;check if a>c
021F 9204                btfss    _z
0220 C223                goto     notz11
0221 6018                movfp   ACCaLO,wreg
0222 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
0223 9004                notz11  btfss    _carry    ;carry set if c>a
0224 C22A                goto     nosub11     ; if c < a
0225 6018                subcall movfp   ACCaLO,wreg ;c-a into c
0226 051C                subwf   ACCcLO
0227 6019                movfp   ACCaHI,wreg
0228 031D                subwfb  ACCcHI
0229 8004                bsf     _carry        ;shift a 1 into d (result)
022A 1B1E                nosub11 rlcfc  ACCdLO
022B 1B1F                rlcfc  ACCdHI
000C                    i = i+1

;

022C 8804                bcf     _carry
022D 1B1A                rlcfc  ACCbLO
022E 1B1B                rlcfc  ACCbHI
022F 1B1C                rlcfc  ACCcLO
0230 1B1D                rlcfc  ACCcHI
0231 6019                movfp   ACCaHI,wreg
0232 041D                subwf   ACCcHI,w      ;check if a>c

```

Math Routines

```

0233 9204                btfss    _z
0234 C237                goto     notz12
0235 6018                movf   ACCaLO,wreg
0236 041C                subwf  ACCcLO,w      ;if msb equal then check lsb
0237 9004                notz12 btfss  _carry      ;carry set if c>a
0238 C23E                goto     nosub12    ; if c < a
0239 6018                subca12 movf   ACCaLO,wreg ;c-a into c
023A 051C                subwf  ACCcLO
023B 6019                movf   ACCaHI,wreg
023C 031D                subwfb ACCcHI
023D 8004                bsf   _carry      ;shift a 1 into d (result)
023E 1B1E                nosub12 rlc   ACCdLO
023F 1B1F                rlc   ACCdHI
000D                    i = i+1

;

0240 8804                bcf   _carry
0241 1B1A                rlc   ACCbLO
0242 1B1B                rlc   ACCbHI
0243 1B1C                rlc   ACCcLO
0244 1B1D                rlc   ACCcHI
0245 6019                movf   ACCaHI,wreg
0246 041D                subwf  ACCcHI,w      ;check if a>c
0247 9204                btfss  _z
0248 C24B                goto     notz13
0249 6018                movf   ACCaLO,wreg
024A 041C                subwf  ACCcLO,w      ;if msb equal then check lsb
024B 9004                notz13 btfss  _carry      ;carry set if c>a
024C C252                goto     nosub13    ; if c < a
024D 6018                subca13 movf   ACCaLO,wreg ;c-a into c
024E 051C                subwf  ACCcLO
024F 6019                movf   ACCaHI,wreg
0250 031D                subwfb ACCcHI
0251 8004                bsf   _carry      ;shift a 1 into d (result)
0252 1B1E                nosub13 rlc   ACCdLO
0253 1B1F                rlc   ACCdHI
000E                    i = i+1

;

0254 8804                bcf   _carry
0255 1B1A                rlc   ACCbLO
0256 1B1B                rlc   ACCbHI
0257 1B1C                rlc   ACCcLO
0258 1B1D                rlc   ACCcHI
0259 6019                movf   ACCaHI,wreg
025A 041D                subwf  ACCcHI,w      ;check if a>c
025B 9204                btfss  _z
025C C25F                goto     notz14
025D 6018                movf   ACCaLO,wreg
025E 041C                subwf  ACCcLO,w      ;if msb equal then check lsb
025F 9004                notz14 btfss  _carry      ;carry set if c>a
0260 C266                goto     nosub14    ; if c < a
0261 6018                subca14 movf   ACCaLO,wreg ;c-a into c
0262 051C                subwf  ACCcLO
0263 6019                movf   ACCaHI,wreg
0264 031D                subwfb ACCcHI
0265 8004                bsf   _carry      ;shift a 1 into d (result)
0266 1B1E                nosub14 rlc   ACCdLO
0267 1B1F                rlc   ACCdHI
000F                    i = i+1

;

0268 8804                bcf   _carry
0269 1B1A                rlc   ACCbLO
026A 1B1B                rlc   ACCbHI
026B 1B1C                rlc   ACCcLO
026C 1B1D                rlc   ACCcHI
026D 6019                movf   ACCaHI,wreg
026E 041D                subwf  ACCcHI,w      ;check if a>c

```

Math Routines

```

026F 9204                btfss    _z
0270 C273                goto     notz15
0271 6018                movfp   ACCaLO,wreg
0272 041C                subwf   ACCcLO,w      ;if msb equal then check lsb
0273 9004                notz15   btfss    _carry    ;carry set if c>a
0274 C27A                goto     nosub15     ; if c < a
0275 6018                subcal5  movfp   ACCaLO,wreg ;c-a into c
0276 051C                subwf   ACCcLO
0277 6019                movfp   ACCaHI,wreg
0278 031D                subwfb  ACCcHI
0279 8004                bsf     _carry      ;shift a 1 into d (result)
027A 1B1E                nosub15  rlc     ACCdLO
027B 1B1F                rlc     ACCdHI
0010                    i = i+1
;
;
;
027C 027C                if SIGNED
                    btfss    sign,MSB      ; negate (ACCc,ACCd)
                    return
                    negMac  ACCcLO,ACCcHI
                    negMac  ACCdLO,ACCdHI
                    return
                    else
027C 0002                return
                    endif
;
;*****
;
;
;           Square Root By Newton Raphson Method
;
; This routine computes the square root of a 16 bit number(with
; low byte in NumLo & high byte in NumHi ). After loading NumLo &
; NumHi with the desired number whose square root is to be computed,
; branch to location Sqrt ( by "GOTO Sqrt" ). " CALL Sqrt" cannot
; be issued because the Sqrt function makes calls to Math routines
; and the stack is completely used up.
; The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
; The total number of iterations is set to ten. If more iterations
; are desired, change "LupCnt equ .10" to the desired value. Also,
; the initial guess value of the square root is given set as
; input/2 ( in subroutine "init" ). The user may modify this scheme
; if a better initial approximation value is known. A good initial
; guess will help the algorithm converge at a faster rate and thus
; less number of iterations required.
; Two utility math routines are used by this program : D_divS
; and D_add. These two routines are listed as seperate routines
; under double precision Division and double precision addition
; respectively.
;
; Note : If square root of an 8 bit number is desired, it is probably
; better to have a table look scheme rather than using numerical
; methods.
; This method is computationally quite intensive and
; slow, but very accurate and the convergence rate is high
;
; Performance :
; Program Memory : 22 (excluding D_divS subroutine)
; Clock Cycles : 3000 (approximately,with 10 iterations)
;
; The #of cycles depends on Number of Iterations Selected.
; In a lot of cases 5 or less iterations may be sufficient
;
;*****
;
;           Newton-Raphson Method
;*****

```

Math Routines

```

Sqrt
027D E28B      call    SqrtInit          ; compute initial sqrt = Num/2
nextIter
    if MODE_FAST
027E 7A24      movfp  NumLo,ACCbLO
027F 7B25      movfp  NumHi,ACCbHI
    else
        movfp  NumLo,wreg
        movwf  ACCbLO
        movfp  NumHi,wreg
        movwf  ACCbHI
    endif
;
0280 E119      call    D_divs              ; double precision division
                                ; double precision addition
                                ; ACCd + ACCa -> ACCd
0281 601E      movfp  ACCdLO,wreg
0282 0F18      addwf  ACCaLO              ;addwf lsb
0283 601F      movfp  ACCdHI,wreg
0284 1119      addwfc ACCaHI              ;addwf msb
                                ; now divide by 2
0285 8804      bcf    _carry
0286 1919      rrcf  ACCaHI
0287 1918      rrcf  ACCaLO
;
0288 1726      decfsz iterCnt
0289 C27E      goto  nextIter
028A 0002      return                    ; End Sqrt
;
SqrtInit
028B B00A      movlw  _LUPCNT
028C 0126      movwf  iterCnt          ; set number of iterations
    if MODE_FAST
028D 7925      movfp  NumHi,ACCaHI
028E 7824      movfp  NumLo,ACCaLO
    else
        movfp  NumHi,wreg
        movwf  ACCaHI
        movfp  NumLo,wreg          ; set initial guess root = NUM/2
        movwf  ACCaLO
    endif
028F 8804      bcf    _carry
0290 1919      rrcf  ACCaHI
0291 1918      rrcf  ACCaLO          ; set initial sqrt = Num/2
0292 0002      return
;
;*****
;
;          8x8 Software Multiplier
;          ( Fast Version : Straight Line Code )
;
;          The 16 bit result is stored in 2 bytes
;
;          Before calling the subroutine " mpy ", the multiplier should
;          be loaded in location " mulplr ", and the multiplicand in
;          " mulcnd ". The 16 bit result is stored in locations
;          H_byte & L_byte.
;
;          Performance :
;
;          Program Memory : 36 words
;          # of cycles     : 36      (excluding call & return)
;          Scratch RAM    : 0 locations
;          W Register     : Used
;
;          This routine is optimized for speed efficiency ( straight line code )
;          For code efficiency, refer to "mult8x8S.asm" ( looped code )
;*****
;          Define a macro for adding & right shifting
;
multiply MACRO
variable i ;

```

```

        i = 0
        .while i < 8
            btfsc    mulplr,i
            addwf   H_byte
            rrcf   H_byte
            rrcf   L_byte
            i = i+1 ;
        .endw
    ENDM                                     ; End of macro
;
;
mpy8x8_F
0293 291B    clrf   H_byte
0294 291A    clrf   L_byte
0295 6018    movfp  mulcnd,wreg           ; move the multiplicand to W reg.
0296 8804    bcf   _carry                ; Clear the carry bit in the status Reg.
;
    multiply
0000        variable i                    ;
0000        i = 0
        .while i < 8
            btfsc    mulplr,i
            addwf   H_byte
            rrcf   H_byte
            rrcf   L_byte
            i = i+1 ;
        .endw

0297 9819        btfsc    mulplr,i
0298 0F1B        addwf   H_byte
0299 191B        rrcf   H_byte
029A 191A        rrcf   L_byte
0001            i = i+1 ;

029B 9919        btfsc    mulplr,i
029C 0F1B        addwf   H_byte
029D 191B        rrcf   H_byte
029E 191A        rrcf   L_byte
0002            i = i+1 ;

029F 9A19        btfsc    mulplr,i
02A0 0F1B        addwf   H_byte
02A1 191B        rrcf   H_byte
02A2 191A        rrcf   L_byte
0003            i = i+1 ;

02A3 9B19        btfsc    mulplr,i
02A4 0F1B        addwf   H_byte
02A5 191B        rrcf   H_byte
02A6 191A        rrcf   L_byte
0004            i = i+1 ;

02A7 9C19        btfsc    mulplr,i
02A8 0F1B        addwf   H_byte
02A9 191B        rrcf   H_byte
02AA 191A        rrcf   L_byte
0005            i = i+1 ;

02AB 9D19        btfsc    mulplr,i
02AC 0F1B        addwf   H_byte
02AD 191B        rrcf   H_byte
02AE 191A        rrcf   L_byte
0006            i = i+1 ;

02AF 9E19        btfsc    mulplr,i
02B0 0F1B        addwf   H_byte
02B1 191B        rrcf   H_byte
02B2 191A        rrcf   L_byte
0007            i = i+1 ;

```

Math Routines

```

02B3 9F19          btfsc    mulplr,i
02B4 0F1B          addwf   H_byte
02B5 191B          rrcf   H_byte
02B6 191A          rrcf   L_byte
0008              i = i+1      ;

;
02B7 02B7 0002    ;          return
;
;*****
;          8x8 Software Multiplier
;          ( Code Efficient : Looped Code )
;
; The 16 bit result is stored in 2 bytes
;
; Before calling the subroutine " mpy ", the multiplier should
; be loaded in location " mulplr ", and the multiplicand in
; " mulcnd ". The 16 bit result is stored in locations
; H_byte & L_byte.
;
; Performance :
;          Program Memory : 13 words (excluding call & return)
;          # of cycles    : 69      (excluding call & return)
;          Scratch RAM    : 1 byte
;          W Register     : Used
;
; This routine is optimized for code efficiency ( looped code )
; For time efficiency code refer to "mult8x8F.asm" ( straight line code )
;*****
;
mpy8x8_s
02B8 291B          clrfs   H_byte
02B9 291A          clrfs   L_byte
02BA 2922          clrfs   count
02BB 8322          bsf    count,3      ; set count = 8
02BC 6018          movfpl mulcnd,wreg
02BD 8804          bcf    _carry      ; Clear the carry bit in the status Reg.

loop
02BE 9819          btfsc   mulplr,0
02BF 0F1B          addwf   H_byte
02C0 191B          rrcf   H_byte
02C1 191A          rrcf   L_byte
02C2 2119          rrcf   mulplr
02C3 1722          decfsz count
02C4 C2BE          goto   loop

;
02C5 0002          return
;
;*****
;          Numerical Differentiation
;
; The so called "Three-Point Formula" is implemented to
; differentiate a sequence of points (uniformly sampled).
; The eqn implemented is :
;          f'(Xn) = [ f(Xn - 2h) - 4*f(Xn - h) + 3*f(Xn)]*0.5/h
; where Xn is the present sample and 'h' is the step size.
;
; The above formula may be rewritten as :
;
;          f'(Xn) = [ 0.5*f(Xn -2) - 2*f(Xn - 1) + 0.5*3*f(Xn)]*1/DiffK
; where DiffK = h = Step Size
;
; This differentiation routine can be used very effectively
; in the computation of the differential component part in
; a PID Loop calculation in Motor Control Applications
;
; Double precision arithmetic is used throught

```



```

; The present sample value is assumed to be in locations
; (XnHi, XnLo). The past two values are assumed to be in locations
; (Xn_1_Hi, Xn_1_Lo) & (Xn_2_Hi, Xn_2_Lo).
; The output value is located in DiffHi & DiffLo. No overflow
; checking mechanism is implemented. If the values are limited
; to 12 bits, then the user need not worry about overflows
;
; It is user's responsibility to update the past values with the
; present values before calling this routine.
; After computation, the present value Xn is not moved to Xn_1
; because the user may want these values to be intact for other
; computations ( say numerical integration)
; Also it is user's responsibility to set past 2 values
; (Xn_1 & Xn_2) values to be zero on initialization.
;
;*****
;
Diff
02C6 602B      movfp    Xn_2_Lo,wreg
02C7 0E27      addwf   XnLo,w
02C8 011A      movwf   ACCbLO
02C9 602C      movfp    Xn_2_Hi,wreg
02CA 1028      addwfc  XnHi,w
02CB 011B      movwf   ACCbHI      ; Y = f(Xn-2) + f(Xn)
;
02CC 6027      movfp    XnLo,wreg
02CD 0F1A      addwf   ACCbLO
02CE 6028      movfp    XnHi,wreg
02CF 111B      addwfc  ACCbHI
02D0 6027      movfp    XnLo,wreg
02D1 0F1A      addwf   ACCbLO
02D2 6028      movfp    XnHi,wreg
02D3 111B      addwfc  ACCbHI      ; Y = f(Xn-2) + 3*f(Xn)
;
02D4 8804      bcf     _carry
02D5 191B      rrcf   ACCbHI
02D6 191A      rrcf   ACCbLO      ; Y = 0.5*[ f(Xn-2) + 3*f(Xn) ]
;
02D7 6029      movfp    Xn_1_Lo,wreg
02D8 051A      subwf   ACCbLO
02D9 602A      movfp    Xn_1_Hi,wreg
02DA 031B      subwfb  ACCbHI
02DB 6029      movfp    Xn_1_Lo,wreg
02DC 051A      subwf   ACCbLO
02DD 602A      movfp    Xn_1_Hi,wreg
02DE 031B      subwfb  ACCbHI      ; Y = 0.5*[f(Xn-2) + 3*f(Xn)] - 2*f(Xn-1)
;
02DF 602D      movfp    DiffKLo,wreg
02E0 0118      movwf   ACCaLO
02E1 602E      movfp    DiffKHi,wreg
02E2 0119      movwf   ACCaHI
;
02E3 E119      call    D_divs
02E4 601A      movfp    ACCbLO,wreg
02E5 012F      movwf   DiffLo
02E6 601B      movfp    ACCbHI,wreg
02E7 0130      movwf   DiffHi      ; result = Y/h
;
02E8 0002      return
;
;*****
;
; Numerical Integration
;
;
; Simpson's Three-Eighths Rule is implemented
;
; Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3) ]*3*h/8

```

Math Routines

```

;
; where 'h' is the step size and the integral is over the
; range X0 to X3
; The above equation can be rewritten as
;
;
;      Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*IntgK
;
; where IntgK = 3*h/8 (in locations (IntgKHi, IntgKHi)
;
; This Integration routine can be used very effectively
; in the computation of the integral component part in
; a PID Loop calculation in Motor Control Applications
;
; Double precision arithmetic is used throught
; The three input values over which the integral is to be computed
; are assumed to be in locations (X0Lo,X0Hi), (X1Lo,X1Hi) , (X2Lo,X2Hi)
; and (X3Lo,X3Hi)
; The output value is located in IntgHi & IntgLo. No overflow
; checking mechanism is implemented. If the values are limited
; to 12 bits, then the user need not worry about overflows
;
; It is user's responsibility to update the past values with the
; present values before calling this routine.
; After computation, the present value Xn is not moved to Xn_1
; because the user may want these values to be intact for other
; computations ( say numerical integration)
; Also it is user's responsibility to set past 2 values
; (Xn_1 & Xn_2) values to be zero on initialization.
;
;
;*****
;
Integrate
02E9 6031      movfp    X0Lo,wreg
02EA 0E37      addwf    X3Lo,w
02EB 011A      movwf    ACCbLO
02EC 6032      movfp    X0Hi,wreg
02ED 1038      addwfc   X3Hi,w
02EE 011B      movwf    ACCbHI      ; Intg = f(X0) + f(X3)
;
02EF 6033      movfp    X1Lo,wreg
02F0 0F1A      addwf    ACCbLO
02F1 6034      movfp    X1Hi,wreg
02F2 111B      addwfc   ACCbHI      ; Intg = f(X0) + f(X3) +X1
02F3 6033      movfp    X1Lo,wreg
02F4 0F1A      addwf    ACCbLO
02F5 6034      movfp    X1Hi,wreg
02F6 111B      addwfc   ACCbHI      ; Intg = f(X0) + f(X3) +2*X1
02F7 6033      movfp    X1Lo,wreg
02F8 0F1A      addwf    ACCbLO
02F9 6034      movfp    X1Hi,wreg
02FA 111B      addwfc   ACCbHI      ; Intg = f(X0) + f(X3) +3*X1
;
02FB 6035      movfp    X2Lo,wreg
02FC 0F1A      addwf    ACCbLO
02FD 6036      movfp    X2Hi,wreg
02FE 111B      addwfc   ACCbHI      ; Intg = f(X0) + f(X3) +3*X1 + X2
02FF 6035      movfp    X2Lo,wreg
0300 0F1A      addwf    ACCbLO
0301 6036      movfp    X2Hi,wreg
0302 111B      addwfc   ACCbHI      ; Intg = f(X0) + f(X3) +3*X1 + 2*X2
0303 6035      movfp    X2Lo,wreg
0304 0F1A      addwf    ACCbLO
0305 6036      movfp    X2Hi,wreg
0306 111B      addwfc   ACCbHI      ; Intg = f(X0) + f(X3) +3*X1 + 3*X2
;
0307 6039      movfp    IntgKLo,wreg
0308 0118      movwf    ACCaLO
0309 603A      movfp    IntgKHi,wreg

```

Math Routines

```

030A 0119          movwf    ACCaHI      ; ACCa = IntgK (prepare for multiplica-
tion)

;
030B E050          call     D_mpyS      ; make sure to set for either SIGNED or
UNSIGNEd
030C 601E          movfp    ACCdLO,wreg
030D 013B          movwf    IntgLo      ; 32 bit result in ACCd & ACCc
030E 601F          movfp    ACCdHI,wreg
030F 013C          movwf    IntgHi      ; upper 16 bits - result

;
0310 0002          return

;
;*****
;
;                      Random Number Generator
;
; This routine generates a 16 Bit Pseudo Sequence Random Generator
; It is based on Linear shift register feedback. The sequence
; is generated by (Q15 xorwf Q14 xorwf Q12 xorwf Q3 )
;
; The 16 bit random number is in location RandHi(high byte)
; & RandLo (low byte)
;
; Before calling this routine, make sure the initial values
; of RandHi & RandLo are NOT ZERO
; A good choice of initial random number is 0x3045
;*****
Random16
0311 1A19          rlcwf    RandHi,w
0312 0C19          xorwf    RandHi,w
0313 1B00          rlcwf    wreg          ; carry bit = xorwf(Q15,14)

;
0314 1D19          swapf    RandHi
0315 1C18          swapf    RandLo,w
0316 2300          rlncf    wreg
0317 0C19          xorwf    RandHi,w      ; LSB = xorwf(Q12,Q3)
0318 1D19          swapf    RandHi
0319 B501          andlw   0x01
031A 1B18          rlcwf    RandLo
031B 0D18          xorwf    RandLo
031C 1B19          rlcwf    RandHi
031D 0002          return

;
;*****
;
;                      Gaussian Noise Generator
;
; This routine generates a 16 Bit Gaussian distributed random
; points. This routine calls the routine "Random16", which
; generates a psuedo random noise sequence. Gaussian noise
; is computed using the CENTRAL LIMIT THEOREM.
; The Central Limit Theorem states that the average weighted
; sum of uncorelated samples tends to have a Gaussian distribution
; For practical purposes, the sum could be over a sample size
; of 32 Random numbers. Better results could result if a larger
; sample size is desired. For faster results, a sum over 16 samples
; would also be adequate ( say, for applications like Speech synthesis,
; channel simulations, etc).
;
; The 16 bit Gaussian distributed point is in locations
; GaussHi & GaussLo
;
; Before calling this routine, the initial seed of Random
; number should be NON ZERO ( refer to notes on "Random16" routine
;
;*****
Gauss
031E 2922          clrf     count

```

Math Routines

```
031F 8522          bsf     count,5          ; set Sample size = 32
0320 291A          clrfd  GaussLo
0321 291B          clrfd  GaussHi
0322 2920          clrfd  GaussTmp
;
NextGauss
0323 E311          call   Random16         ; get a random value
0324 6018          movfp  RandLo,wreg
0325 0F1A          addwf  GaussLo
0326 6019          movfp  RandHi,wreg
0327 111B          addwfc GaussHi
0328 2900          clrfd  wreg
0329 1120          addwfc GaussTmp
032A 1722          decfsz count
032B C323          goto   NextGauss      ; sum 16 random numbers
;
032C B005          movlw  5
GaussDiv16
032D 1920          rrcfd  GaussTmp
032E 191B          rrcfd  GaussHi
032F 191A          rrcfd  GaussLo      ; wegthed average
0330 1700          decfsz wreg          ; divide by 32
0331 C32D          goto   GaussDiv16
;
0332 0002          return
;
END                ; End Of arith.asm
```

```
Errors   : 0
Warnings : 0
```

BCD ARITHMETIC ROUTINES LISTING FILE OF BCD.ASM

MPASM B0.54

PAGE 1

```

0001          #define PAGE    EJECT

                ;TITLE    "BCD Arithmetic Routines : Ver 1.0"

                ;*****
                ;                BCD Arithmetic Routines
                ;*****

                LIST        P=17C42, C=80, L=0, R=DEC

                include    "17c42.h"

                CBLOCK    0x20
0020 0002                Lbyte, Hbyte
0022 0003                R2, R1, R0                ;must maintain R2, R1, R0 sequence

0025 0001                count
0026 0002                Num1, Num2

                ENDC

                ;
0026                BCD    equ    Num1
0026                Htemp    equ    Num1
0027                Ltemp    equ    Num2
                ;

                ORG        0x0000
                ;*****
                ;                BCD Arithmetic Test Program
                ;*****

                ;
                main
0000 2B21                setf    Hbyte
0001 2B20                setf    Lbyte
                ;                ; 16 bit binary num = 0xfff

0002 E01F                call    B2_BCD_Looped    ; after conversion the Deci
                ;                ; in R0, R1, R2 = 06,55,35

0003 2B21                setf    Hbyte
0004 2B20                setf    Lbyte
0005 E03F                call    B2_BCD_Straight    ; same as above, but straig

                ;
0006 B006                movlw   0x06
0007 0124                movwf   R0
0008 B055                movlw   0x55
0009 0123                movwf   R1
000A B035                movlw   0x35
000B 0122                movwf   R2                ; setf R0R1R2 = 65535

                ;
000C E082                call    BCDtoB    ; after conversion Hbyte =
                ;                ; and Lbyte = 0xff
    
```

Math Routines

```
000D B099          movlw    0x99
000E 0126          movwf   Num1
000F B099          movlw    0x99
0010 0127          movwf   Num2          ; setf Num1 = Num2 = 0x99

;
0011 E093          call    BCDAdd        ; after addition, Num2 = 98
;
;                  ; and Num1 = 01 ( 99+99 = 1
;
;
0012 B063          movlw    0x63          ; setf Wreg = 63 hex
0013 E015          call    BinBCD        ; after conversion, BCD = 9
;
;                  ; 63 hex = 99 decimal.
;
;
0014 C014          self    goto    self
;
;*****
;
;                  Binary To BCD Conversion Routine (8 bit)
;
;                  This routine converts the 8 bit binary number in th
; to a 2 digit BCD number in location BCD( compacted BCD Co
;
;                  The least significant digit is returned in location
; the most significant digit is returned in location MSD.
;
; Performance :
;                  Program Memory : 10
;                  Clock Cycles   : 62 (worst case when W =
;
;                  ( i.e max Decimal nu
;*****
;
; BinBCD
0015 2926          clr     BCD
; again
0016 B1F6          addlw   -10
0017 9004          btfs   _carry
0018 C01B          goto   swapBCD
0019 1526          incf   BCD
001A C016          goto   again
; swapBCD
001B B10A          addlw   10
001C 1D26          swapf  BCD
001D 0926          iorwf  BCD
001E 0002          return
;
;*****
;
;                  Binary To BCD Conversion Routine (16 Bit)
;
;                  (LOOPED Version)
;
;                  This routine converts a 16 Bit binary Number to a 5
; BCD Number.
;
;                  The 16 bit binary number is input in locations Hbyt
;
; Lbyte with the high byte in Hbyte.
;
;                  The 5 digit BCD number is returned in R0, R1 and R2
```

Math Routines

```
; containing the MSD in its right most nibble.
;
; Performance :
; Program Memory : 32
; Clock Cycles : 750
;
;*****
;
B2_BCD_Looped
001F 8404      bsf      _fs0
0020 8504      bsf      _fs1      ; set fsr0 for no auto inc

;
0021 8804      bcf      _carry
0022 2925      clrfs   count
0023 8425      bsf      count,4      ; set count = 16
0024 2924      clrfs   R0
0025 2923      clrfs   R1
0026 2922      clrfs   R2
loop16a
0027 1B20      rlcfs   Lbyte
0028 1B21      rlcfs   Hbyte
0029 1B22      rlcfs   R2
002A 1B23      rlcfs   R1
002B 1B24      rlcfs   R0
;
002C 2725      dcfsnz  count
002D 0002      return
adjDEC
002E B022      movlw   R2      ; load R2 as indirect addr

002F 0101      movwf   fsr0
0030 E036      call   adjBCD
;
0031 1501      incf   fsr0
0032 E036      call   adjBCD
;
0033 1501      incf   fsr0
0034 E036      call   adjBCD
;
0035 C027      goto   loop16a
;
adjBCD
0036 6000      movfp   indf0,wreg
0037 B103      addlw  0x03
0038 9B00      btfs   wreg,3      ; test if result > 7
0039 0100      movwf   indf0
003A 6000      movfp   indf0,wreg
003B B130      addlw  0x30
003C 9F00      btfs   wreg,7      ; test if result > 7
003D 0100      movwf   indf0      ; save as MSD
003E 0002      return
;
;*****
; Binary To BCD Conversion Routine (16 Bit)
;
; (Partial Straight Line Version)
;
; This routine converts a 16 Bit binary Number to a 5
; BCD Number.
;
; The 16 bit binary number is input in locations Hbyt
; Lbyte with the high byte in Hbyte.
; The 5 digit BCD number is returned in R0, R1 and R2
```



Math Routines

```
; containing the MSD in its right most nibble.
;
; Performance :
;           Program Memory : 44
;           Clock Cycles   : 572
;
;*****
;
; B2_BCD_Straight
003F 8404      bsf      _fs0
0040 8504      bsf      _fs1      ; set fsr0 for no auto inc
;
0041 8804      bcf      _carry
0042 2925      clrfs  count
0043 8425      bsf      count,4      ; set count = 16
0044 2924      clrfs  R0
0045 2923      clrfs  R1
0046 2922      clrfs  R2
loop16b
0047 1B20      rlcfs  Lbyte
0048 1B21      rlcfs  Hbyte
0049 1B22      rlcfs  R2
004A 1B23      rlcfs  R1
004B 1B24      rlcfs  R0
;
004C 2725      dcfsnz count
004D 0002      return      ; DONE
004E B022      movlwf R2      ; load R2 as indirect addr
;
004F 0101      movwfs fsr0
; adjustBCD
0050 6000      movfpl indf0,wreg
0051 B103      addlw  0x03
0052 9B00      btfscc wreg,3      ; test if result > 7
0053 0100      movwfs indf0
0054 6000      movfpl indf0,wreg
0055 B130      addlw  0x30
0056 9F00      btfscc wreg,7      ; test if result > 7
0057 0100      movwfs indf0      ; save as MSD
;
0058 1501      incfs  fsr0
; adjustBCD
0059 6000      movfpl indf0,wreg
005A B103      addlw  0x03
005B 9B00      btfscc wreg,3      ; test if result > 7
005C 0100      movwfs indf0
005D 6000      movfpl indf0,wreg
005E B130      addlw  0x30
005F 9F00      btfscc wreg,7      ; test if result > 7
0060 0100      movwfs indf0      ; save as MSD
;
0061 1501      incfs  fsr0
; adjustBCD
0062 6000      movfpl indf0,wreg
0063 B103      addlw  0x03
0064 9B00      btfscc wreg,3      ; test if result > 7
0065 0100      movwfs indf0
0066 6000      movfpl indf0,wreg
0067 B130      addlw  0x30
0068 9F00      btfscc wreg,7      ; test if result > 7
0069 0100      movwfs indf0      ; save as MSD
;
006A C047      goto   loop16b
;
;*****
;
; BCD To Binary Conversion
```


Math Routines

```
;
;   This routine converts a 5 digit BCD number to a 16
;   number.
;   The input 5 digit BCD numbers are asumed to be in 1
;   R0, R1 & R2 with R0 containing the MSD in its right most
;
;   The 16 bit binary number is output in registers Hbyte
;   ( high byte & low byte repectively ).
;
;   The method used for conversion is :
;   input number X = abcde ( the 5 digit BCD nu
;
;   X = (R0,R1,R2) = abcde = 10[10[10[10a+b]+c]+d]+e
;
;   Performance :
;   Program Memory : 30
;   Clock Cycles   : 112
;
;*****
;
;   mpy10b
006B B50F      andlw   0x0f
006C 0F20      addwfc  Lbyte
006D 9804      btfsc  _carry
006E 1521      incf   Hbyte

;   mpy10a
006F 8804      bcf    _carry      ; multiply by 2
0070 1A20      rlcfc  Lbyte,w
0071 0127      movwfc Ltemp
0072 1A21      rlcfc  Hbyte,w      ; (Htemp,Ltemp) = 2*N
0073 0126      movwfc Htemp

;
0074 8804      bcf    _carry      ; multiply by 2
0075 1B20      rlcfc  Lbyte
0076 1B21      rlcfc  Hbyte
0077 8804      bcf    _carry      ; multiply by 2
0078 1B20      rlcfc  Lbyte
0079 1B21      rlcfc  Hbyte
007A 8804      bcf    _carry      ; multiply by 2
007B 1B20      rlcfc  Lbyte
007C 1B21      rlcfc  Hbyte      ; (Hbyte,Lbyte) = 8*N

;
007D 6027      movwfc Ltemp,wreg
007E 0F20      addwfc Lbyte
007F 6026      movwfc Htemp,wreg
0080 1121      addwfc Hbyte
0081 0002      return      ; (Hbyte,Lbyte) = 10*N

;
;   BCDtoB
0082 2921      clrfc  Hbyte
0083 6024      movwfc R0,wreg
0084 B50F      andlw   0x0f
0085 0120      movwfc Lbyte
0086 E06F      call   mpy10a      ; result = 10a+b

;
0087 1C23      swapfc R1,w
0088 E06B      call   mpy10b      ; result = 10[10a+b]

;
0089 6023      movwfc R1,wreg
008A E06B      call   mpy10b      ; result = 10[10[10a+b]+c]

;
008B 1C22      swapfc R2,w
```

Math Routines

```
008C E06B          call    mpy10b          ; result = 10[10[10[10a+b]+
;
008D 6022          movfp   R2,wreg
008E B50F          andlw  0x0f
008F 0F20          addwf  Lbyte
0090 9804          btfs   _carry
0091 1521          incf   Hbyte          ; result = 10[10[10[10a+b]
;
0092 0002          return          ; BCD to binary conversion
;
;*****
;
;           Unsigned BCD Addition
;
;           This routine performs a 2 Digit Unsigned BCD Additi
;
;           It is assumed that the two BCD numbers to be added are in
;
;           locations Num1 & Num2. The result is the sum of Num1+Num2
;
;           and is stored in location Num2 and the overflow carry is
;
;           in location Num1
;
;           Performance :
;           Program Memory :      5
;           Clock Cycles   :      5
;
;*****
;
BCDAdd
0093 6026          movfp   Num1,wreg
0094 0E27          addwf  Num2,w          ; perform binary addition
;
0095 2F27          daw   Num2          ; adjust for BCD addition
0096 2926          clrf  Num1
0097 1B26          rlc   Num1          ; set Num1 = carry bit
0098 0002          return
;
;*****
;
;           END
```

```
Errors   : 0
Warnings : 0
```

BINARY FLOATING POINT ARITHMETIC ROUTINES LISTING FILE OF FLOAT.ASM

MPASM B0.54

PAGE 1

Binary Floating Arithmetic Routines For PIC17C42 : Ver 1.0

```

0001          #define PAGE    EJECT

                TITLE    "Binary Floating Arithmetic Routines For PIC17C42 : Ver 1.0"

                LIST     P=17C42, C=80, L=0, T=ON, R=DEC
;
                include "17c42.h"

;
;*****
;
;           Binary Floating Point Addition, Subtraction
;
;           Multiplication Routines
;
; Mantissa = 16 bits
; Exponent = 8 bits    ( exponent is binary and not decima
;
;           i.e a number ABCD EXP(X) = 0xABCD
;
;
;           Before calling any of the following floating point
; it is required to set Indirect Register 0 ( FSR0 ) for
; No-Autoincrement ( i.e. Set bits FS0 & FS1 in ALUSTA to
;
;*****
;
                CBLOCK 0x20
0020 0003                ACCaLO, ACCaHI, EXPa
0023 0003                ACCbLO, ACCbHI, EXPb
0026 0002                ACCcLO, ACCcHI
0028 0002                ACCdLO, ACCdHI
002A 0002                temp, sign
                ENDC
;
0001          Model6 equ    TRUE          ; Change this to FALSE for 32 bit

;
                ORG     0x0000
;
;*****
;
;           Floating Point Routines Test Program
;*****
;
; main
;
0000 8404                bsf     _fs0          ; set FSR0 for no autoincr
0001 8504                bsf     _fs1
;
0002 E009                call    loadAB          ; result of adding ACCb(EXP
0003 E019                call    F_add          ; Here Accb = 403F, EXPb =

```

Math Routines

```

0004 E009          call    loadAB          ; result of subtracting ACC
0005 E016          call    F_sub          ; Here Accb = 7F7F, EXPb =
;
0006 E009          call    loadAB          ; result of multiplying ACC
0007 E03B          call    F_mpy         ; Here ACCb = FF7E, EXPb =
;
0008 C008          self    goto    self
;
;      Load constant values to (ACCa, EXPa) & (ACCb, EXPb) fo
;
;      loadAB
0009 B001          movlw   0x01
000A 0121          movwf  ACCaHI
000B B0FF          movlw   0xff          ; loads ACCa = 01FF EXP(4
;
000C 0120          movwf  ACCaLO
000D B004          movlw   0x04
000E 0122          movwf  EXPa
;
000F B07F          movlw   0x7f
0010 0124          movwf  ACCbHI
0011 B0FF          movlw   0xff          ; loads ACCb = 7fff EXP(6
;
0012 0123          movwf  ACCbLO
0013 B006          movlw   0x06
0014 0125          movwf  EXPb
0015 0002          return
;
;*****
;      Floating Point Subtraction ( ACCb - ACCa -> ACCb )
;
;      Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16
;
;      (a) Load the 1st operand in location ACCaLO & ACCaHI
;
;      the 8 bit exponent in EXPa .
;      (b) Load the 2nd operand in location ACCbLO & ACCbHI
;
;      the 8 bit exponent in EXPb .
;      (c) CALL F_sub
;      (d) The result is in location ACCbLO & ACCbHI ( 16 b
;
;      the 8 bit exponent in EXPb.
;
;*****
;
;      F_sub
0016 B020          movlw   ACCaLO
0017 0101          movwf  fsr0
0018 E075          call    negate          ; At first negate ACCa; Th
;
;*****
;      Floating Point Addition ( ACCb + ACCa -> ACCb )
;
;      Addition : ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bi
;
;      (a) Load the 1st operand in location ACCaLO & ACCaHI
;
;      the 8 bit exponent in EXPa.

```

```

;      (b) Load the 2nd operand in location ACCbLO & ACCbHI
;
;      the 8 bit exponent in EXPb.
;      (c) CALL F_add
;      (d) The result is in location ACCbLO & ACCbHI ( 16 b
;
;      the 8 bit exponent in EXPb
;
;*****
;
;
F_add
0019 6022      movfp    EXPa,wreg
001A 3125      cpfseq   EXPb
001B C01D      goto     Lbl1
001C C026      goto     noAddNorm      ; if exponents are equal

Lbl1
001D 3025      cpfslt   EXPb
001E E09E      call    F_swap      ; if B > A then swap ( A<->

001F 6022      movfp    EXPa,wreg
0020 0525      subwf   EXPb

sloop
0021 E030      call    addNorm
0022 1F25      incfsz  EXPb
0023 C021      goto     sloop
0024 6022      movfp    EXPa,wreg
0025 0125      movwf   EXPb

noAddNorm
0026 6021      movfp    ACCaHI,wreg
0027 0824      iorwf   ACCbHI,w
0028 012B      movwf   sign      ; save the sign ( MSB sta

0029 E036      call    D_add      ; compute double precision

002A 972B      btfs    sign,MSB
002B 9724      btfs    ACCbHI,MSB
002C 0002      return
002D 1525      incf   EXPb
002E 8804      bcf    _carry
002F C033      goto   shftr

;
addNorm
0030 8804      bcf    _carry
0031 9F24      btfs   ACCbHI,MSB
0032 8004      bsf   _carry      ; set carry if < 0

shftr
0033 1924      rrcf   ACCbHI
0034 1923      rrcf   ACCbLO
0035 0002      return

;
;*****
;
;      Double Precision Addition
;
;
D_add
0036 6020      movfp    ACCaLO,wreg
0037 0F23      addwf   ACCbLO      ;addwf lsb
0038 6021      movfp    ACCaHI,wreg
0039 1124      addwfc  ACCbHI      ;addwf msb with carry
003A 0002      return

;
;*****
;
;      Binary Floating Point Multiplication
;
;
;      Multiplication :
;
;      ACCb(16 bits)EXP(b) * ACCa(16

```

Math Routines

```

; where, EXP(x) represents an 8 bit exponent.
; (a) Load the 1st operand in location ACCaLO & ACCaHI
;
; an 8 bit exponent in locati
; (b) Load the 2nd operand in location ACCbLO & ACCbHI
;
; an 8 bit exponent in locati
; (c) CALL F_mpy
; (d) The 16 bit result overwrites ACCb(ACCbLO & ACCbH
;
; is stored in EXPb and the results are norma
;
; NOTE : If one needs to get a 32 bit product ( & an 8 bit e
;
; re assemble the program after changing the line "
;
; to " Model6 equ FALSE ".
; If this option is chosen, then the 32 bit result i
;
; ( ACCbHI, ACCbLO, ACCcHI, ACCcLO ) and the 8 bit e
;
; This method ( with " Model6 equ FALSE " ) is NOT R
;
; If a 32 bit mantissa is desired, set MODEL6 equ FALSE
;*****
;
; F_mpy
003B E07D call S_SIGN
003C E064 call setup
;
mloop
003D 8804 bcf _carry
003E 1929 rrcf ACCdHI ;rotate d right
003F 1928 rrcf ACCdLO
0040 9804 btfsc _carry
0041 E036 call D_add
0042 1924 rrcf ACCbHI
0043 1923 rrcf ACCbLO
0044 1927 rrcf ACCcHI
0045 1926 rrcf ACCcLO
0046 172A decfsz temp ;loop until all bits ch
;
0047 C03D goto mloop
;
0048 6022 movfp EXPa,wreg
0049 0F25 addwf EXPb
;
; if Model6
004A 3324 tstfsz ACCbHI
004B C05B goto finup ; if ACCbHI != 0
004C 3323 tstfsz ACCbLO
004D C055 goto Shft08 ; if ACCbLO != 0 && ACCbHI
;
004E 6027 movfp ACCcHI,wreg
004F 0124 movwf ACCbHI ; if ACCb == 0, then move
;
0050 6026 movfp ACCcLO,wreg
0051 0123 movwf ACCbLO
0052 B010 movlw 16
0053 0F25 addwf EXPb
0054 C05B goto finup
;
; Shft08
0055 6023 movfp ACCbLO,wreg

```

Math Routines

```

0056 0124          movwf   ACCbHI
0057 6027          movfp   ACCcHI,wreg
0058 0123          movwf   ACCbLO
0059 B008          movlw   8
005A 0F25          addwf   EXPb
;
;          endif          ; matching endif for IF Model16
;
; finup
005B 972B          btfss   sign,MSB
005C C08B          goto   F_norm
;
005D B026          movlw   ACCcLO
005E 0101          movwf   fsr0
005F E075          call   negate
0060 B023          movlw   ACCbLO
0061 0101          movwf   fsr0
0062 E075          call   negate
0063 C08B          goto   F_norm          ; normalize floating point
;
;*****
;
; setup
0064 292A          clrfs   temp
0065 842A          bsf    temp,4          ; set temp = 16
0066 6024          movfp   ACCbHI,wreg          ;move ACCb to ACCd
0067 0129          movwf   ACCdHI
0068 6023          movfp   ACCbLO,wreg
0069 0128          movwf   ACCdLO
006A 2924          clrfs   ACCbHI
006B 2923          clrfs   ACCbLO          ; clear ACCb ( ACCbLO & AC
006C 0002          return
;
;*****
;          Double Precision Negate Routines
;
negateAlt
006D 6000          movfp   indf0,wreg
006E 8D04          bcf    _fs1
006F 2D00          negw   indf0
0070 8504          bsf    _fs1
0071 6000          movfp   indf0,wreg
0072 2900          clrfs   indf0
0073 0300          subwfb indf0
0074 0002          return
;
negate
0075 1300          comf   indf0
0076 8D04          bcf    _fs1
0077 1500          incf   indf0
0078 8504          bsf    _fs1
0079 9A04          btfsc  _z
007A 0700          decf   indf0
007B 1300          comf   indf0
007C 0002          return
;
;*****
;          Check Sign of the Number, if so negate and set the SIGN
;
; S_SIGN
007D 6021          movfp   ACCaHI,wreg
007E 0C24          xorwf   ACCbHI,w

```

Math Routines

```
007F 012B      movwf    sign
0080 9724      btfss   ACCbHI,MSB      ; if MSB set go & nega

0081 C085      goto    chek_A
;
0082 B023      movlw   ACCbLO
0083 0101      movwf   fsr0
0084 E075      call   negate
;
0085 9721      btfss   ACCaHI,MSB      ; if MSB set go & nega

0086 0002      return
0087 B020      movlw   ACCaLO
0088 0101      movwf   fsr0
0089 E075      call   negate
008A 0002      return
;
;*****
;      Normalize Routine
; Normalizes ACCb for use in floating point calculations.
; Call this routine as often as possible to minimize the lo
; of precision. This routine normalizes ACCb so that the
; mantissa is maximized and the exponent minimized.
;
;*****
;
; F_norm      ; normalize ACCb
008B 3324      tstfsz   ACCbHI
008C C090      goto    C_norm
008D 3323      tstfsz   ACCbLO
008E C090      goto    C_norm
008F 0002      return
;
; C_norm
0090 9E24      btfsz   ACCbHI,MSB-1
0091 0002      return
0092 E095      call   shftSL
0093 0725      decf   EXPb
0094 C090      goto    C_norm
;
; shftSL
0095 8804      bcf     _carry
;
; if Model6
0096 1B26      rlc    ACCcLO
0097 1B27      rlc    ACCcHI
; endif
;
0098 1B23      rlc    ACCbLO
0099 1B24      rlc    ACCbHI
009A 8F24      bcf    ACCbHI,MSB
009B 9804      btfsz   _carry
009C 8724      bsf    ACCbHI,MSB
009D 0002      return
;
;*****
; Swap ACCa & ACCb [ (ACCa,EXPa) <-> (ACCb,EXPb) ]
;
; F_swap
009E 6021      movfp   ACCaHI,wreg
009F 012A      movwf   temp
00A0 6024      movfp   ACCbHI,wreg      ; ACCaHI <-> ACCbHI
00A1 0121      movwf   ACCaHI
00A2 602A      movfp   temp,wreg
```


Math Routines

```

00A3 0124          movwf    ACCbHI
;
00A4 6020          movfp   ACCaLO,wreg
00A5 012A          movwf   temp
00A6 6023          movfp   ACCbLO,wreg    ; ACCaLO <-> ACCbLO
00A7 0120          movwf   ACCaLO
00A8 602A          movfp   temp,wreg
00A9 0123          movwf   ACCbLO
;
00AA 6025          movfp   EXPb,wreg
00AB 012A          movwf   temp
00AC 6022          movfp   EXPa,wreg    ; EXPa <-> EXPb
00AD 0125          movwf   EXPb
00AE 602A          movfp   temp,wreg
00AF 0122          movwf   EXPa
00B0 0002          return
;
;*****
;
;           Normalizes A Floating Point Number
;           The number is assumed to be (LowByte, HighByte, Ex
; consecutive locations. Before calling this routine, the a
; of the LowByte should be loaded into FSR0 (indirect regis
;*****
;
; Normalize
00B1 1501          incf   fsr0
00B2 3300          tstfsz indf0
00B3 C0B9          goto  NextNorm1
00B4 0701          decf   fsr0
00B5 3300          tstfsz indf0
00B6 C0B8          goto  NextNorm2
00B7 0002          return
NextNorm2
00B8 1501          incf   fsr0
NextNorm1
00B9 9E00          btfsz  indf0,MSB-1
00BA 0002          return
00BB E0C0          call  shiftNorm
00BC 1501          incf   fsr0
00BD 0700          decf   indf0
00BE 0701          decf   fsr0
00BF C0B9          goto  NextNorm1
;
shiftNorm
00C0 8804          bcf   _carry
00C1 1501          incf   fsr0
00C2 1B00          rlc   indf0
00C3 1501          incf   fsr0
00C4 1B00          rlc   indf0
00C5 0701          decf   fsr0
00C6 0701          decf   fsr0
00C7 0701          decf   fsr0
00C8 1B00          rlc   indf0
00C9 1501          incf   fsr0
00CA 1B00          rlc   indf0
00CB 8F00          bcf   indf0,MSB
00CC 9804          btfsz  _carry
00CD 8700          bsf   indf0,MSB
00CE 0002          return
;
END
Errors   :    0
Warnings :    0

```

Math Routines

NOTES:

Implementing IIR Digital Filters

INTRODUCTION

This application note describes the implementation of various digital filters using the PIC17C42, the first member of Microchip's 2nd generation 8-bit microcontrollers. The PIC17C42 is a very high speed 8-bit microcontroller with an instruction cycle time of 250ns (@ 16 MHz input clock). Even though PIC17C42 is an 8-bit machine, it's high speed & efficient instruction set allows implementation of digital filters for practical applications. Traditionally digital filters are implemented using expensive Digital Signal Processors (DSPs). In a system the DSP is normally a slave processor being controlled by a either an 8- or 16-bit microcontroller. Where sampling rates are not high (esp. in mechanical control systems), a single chip solution is possible using the PIC17C42.

This application note provides a few examples of implementing digital filters. Example code for 2nd order Infinite Impulse Response (IIR) filters is given. The following type of filters are implemented:

- Low Pass
- High Pass
- Band Pass
- Band Stop (notch) filter

This application note does not explain how to design a filter. Filter design theory is well established and is beyond the scope of this application note. It is assumed that a filter is designed according to the desired specifications. The desired digital filters may be designed using either standard techniques or using commonly available digital filter design software packages.

Finite Impulse Response (FIR) filters have many advantages over IIR filters, but are much more resource intensive (both in terms of execution time and RAM). On the other hand, IIR filters are quite attractive for implementing with the PIC17C42 resources. Especially where, phase information is not so important, IIR filters are a good choice (FIR filters have a linear phase response). Of the various forms used for realizing digital filters (like, Direct form, Direct II form, Cascade form, Parallel, Lattice structure, etc.) the Direct II form is used in this application note. It is easy to understand and simple macros can be built using these structures.

THEORY OF OPERATION

Digital filters in most cases assume the following form of relationship between the output and input sequences.

$$y(n) = - \sum_{i=0}^M a_i y(n-i) + \sum_{j=0}^N b_j x(n-j)$$

The above equation basically states that the present output is a weighted sum of the past inputs and past outputs. In case of FIR filters, the weighted constants $a_i=0$ and in case of IIR filters, at least one of the a_i constant is non zero. In case of IIR, the above formula may be re written in terms of Z transform as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^M b_k z^{-k}}{1 + \sum_{k=1}^N a_k z^{-k}}$$

The above equation can further be rewritten in difference equation format as follows:

$$y(n) = - \sum_{i=1}^M a_i y(n-i) + \sum_{j=0}^N b_j x(n-j)$$

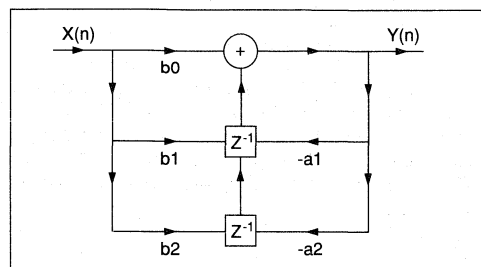
Realization of the above equation is called as the Direct Form II structure. For example, in case of a second order structure, $M=N=2$, gives the following difference equations :

$$d(n) = x(n) + a_1 d(n-1) + a_2 d(n-2) \quad (1)$$

$$y(n) = b_0 d(n) + b_1 d(n-1) + b_2 d(n-2) \quad (2)$$

The above difference equations may be represented as shown in Figure 1.

FIGURE 1 - 2ND ORDER DIRECT FORM II STRUCTURE (TRANPOSED)



Implementing IIR Digital Filters

The structure as shown in Figure 1 may be cascaded to attain a higher order filter. For example, if two stages are cascaded together, a 4th Order IIR Filter is obtained. This way, the output of the 1st stage becomes the input to the second stage. Multiple order filters are thus implemented by cascading a 2nd order filter structure as shown in Figure 1.

IMPLEMENTATION

A 4th order IIR Filter is implemented by cascading two structures shown in Figure 1. The output Y (output of each filter stage) is computed by direct implementation of Equations (1) & (2). Since each stage is similar algorithmically, it is implemented as a Macro using Microchip's Assembler/Linker for PIC17C42. This Macro (labelled "BIQUAD") is called twice for implementing a 4th order filter. The output of the 1st stage is directly fed to the input of the second stage without any scaling.

Scaling is required depending on the particular application. The user can modify the code very easily without any penalty on speed. Also saturation arithmetic is not used. Overflows can be avoided by limiting the input sequence amplitude. All numbers are assumed to be 16 bits in Q15 format (15 decimal points, MSB is sign bit). Thus the user must scale & sign extend the input sequence accordingly. For example, if the input is from a 12 bit A/D converter, the user must sign extend the 12 bit input if bit 11 is a one.

The BIQUAD macro is a generic macro and can be used for all IIR filters whether it is Low Pass, High Pass, Band Pass or Band Stop. A general purpose 16x16 multiplier routine is also provided. This routine is implemented as a straight line code for speed considerations.

The 4th order IIR filter implemented is a Low Pass Filter with the specifications as shown in Table 1.

TABLE 1 - FILTER CONSTANTS

	BAND1	BAND2
Lower Band Edge	0.0	600 Hz
Upper Band Edge	500 Hz	1 KHz
Nominal Gain	1.0	0.0
Nominal Ripple	0.01	0.05
Maximum Ripple	0.00906	0.04601
Ripple in dB	0.07830	-26.75
Sampling Frequency = 2 KHz		

The Low Pass Filter is designed using a digital filter design package (DFDP by Atlanta Signal Processors Inc.). The filter package produces filter constants of the structure shown in Table 1. Table 2 shows the filter coefficients that are obtained for the above Low Pass filter specification.

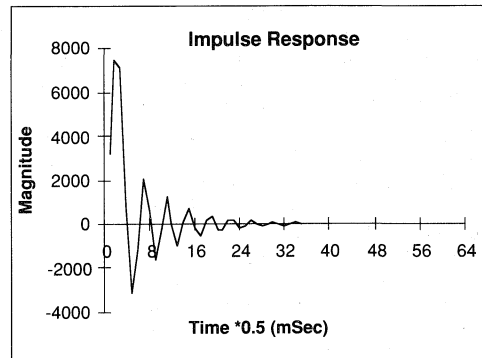
TABLE 2 - FILTER COEFFICIENTS

Co-efficients	a1	a2	b0	b1	b2
Stage 1	-0.133331	0.167145	0.285431	0.462921	0.285431
Stage 2	0.147827	0.765900	0.698273	0.499908	0.698273

The above filter co-efficients (5 per stage) are quantized to Q15 format (i.e they are multiplied by 32768) and saved in program memory (starting at label "_coeff_lpass"). The constants for both the stages are read into data memory using TLRD & TABLRD instructions in the Initialization Routine (labelled "initFilter"). The user may read the coefficients of only one stage at a time and save some RAM at the expense of speed.

The sample 4th order Low Pass IIR Filter is tested by analyzing the impulse response of the filter. An impulse signal is fed as input to the filter. This is simulated by forcing the input to the filter by a large quantity (say 7F00h) on the first input sample, and the all zeros from the 2nd sample onwards. The output sequence is the filter's impulse response and is captured into the PICMASTER's (Microchip's Universal In-Circuit Emulator) real time trace buffer. This captured data from PICMASTER is saved to file and analyzed. Analysis was done using MathCad for Windows and DSP Analysis program from Burr-Brown (DSPLAY). The Fourier Transform of this Impulse response of the filter should display the Filter's frequency response, in this case being a Low Pass type. The plots of the impulse response and the frequency response is shown in figures below.

FIGURE 2 - IMPULSE RESPONSE CAPTURED FROM PIC-MASTER



DSPLAY is a trademark of Burr-Brown
 DFDP is a trademark of Atlanta Signal Processing Inc.
 Windows is a trademark of Microsoft Corporation
 MathCad is a registered trademark of MathSoft, Inc.

Implementing IIR Digital Filters

FIGURE 3 - SPECTRUM COMPUTED FROM IMPULSE RESPONSE

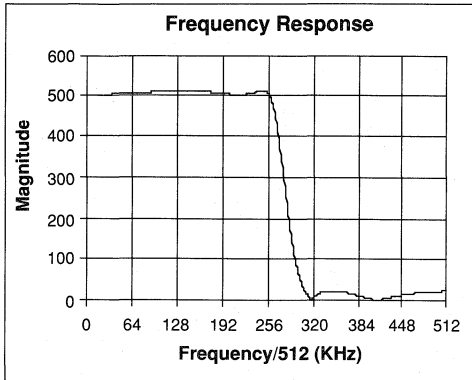
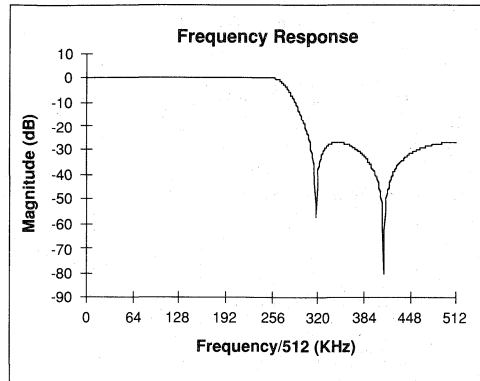


FIGURE 4 - LOG MAGNITUDE SPECTRUM OF IMPULSE RESPONSE



PERFORMANCE

The resource requirements for filter implementations using PIC17C42 is given below. These numbers can be used to determine whether a higher order filter can be executed in real time. The same information may be used to determine the highest sampling rate possible.

FILTER APPLICATIONS

Digital filters find applications in many areas especially involving processing of real world signals. In some applications like ABS systems in an automobile, digital filtering becomes a must. In this case elimination of noise (especially glitches and false readings of sensors) is very critical and thus the requirement of digital signal processing.

Digital filters are also needed in Process Control where notch filters and low pass filters are desired because the signals from sensors are transmitted over long lines, especially in a very noisy environment. In this case typically a notch filter (centering 50Hz or 60Hz) is used. In case of eliminating background noise, a band stop filter (e.g. 40Hz to 120Hz) is used. The sample code given in this application note can be used to design a feedback control system's digital compensator. For example, a typical DC Motor's digital compensator (like dead beat compensator) is of second order and has the same filter structure as is implemented in this application note.

TABLE 3 - RESOURCE REQUIREMENTS

Timing (Cycles)†	#of Filter Stages*775 + 16
Program Memory (locations)†	#of Filter Stages*68 + 290
RAM (File Registers)	#of Filter Stages*16+16

†: The above numbers do not include the initialization routine.

Author: Amar Palacherla
Logic Products Division

TABLE 4 - RESOURCE REQUIREMENTS

Filter Order	Cycles	Real Time (@ 16 MHz)	Maximum Sampling	Program Memory†	RAM
2	791	197.75 µSec	5.05 KHz	358	32
4	1566	391.5 µSec	2.55 KHz	426	48
6	2341	585.25 µSec	1.7 KHz	494	64
8	3116	779.0 µSec	1.28 KHz	562	80
10	3891	972.75 µSec	1.0 KHz	630	96

†: The above numbers do not include the initialization routine.

Implementing IIR Digital Filters

APPENDIX A: IIR.ASM

MPASM B0.54
Digital IIR Filter Using PIC17C42

PAGE 1

```
TITLE "Digital IIR Filter Using PIC17C42"
LIST P=17C42, C=L20, T=ON, R=DEC, N=0

;
include "17c42.h"

;
include "17c42.mac"

;
include "17c42iir.mac"
;*****
; PIC17C42 MACRO
;
; Macro For A Bi-Quad IIR Filter
; 2nd order Direct Form (Transposed) Type
;
; Filter co-efficients B0 & B2 are assumed equal
;
; The difference equations for each cascade section is given by :
;  $Y(n) = B0*D(n) + B1*D(n-1) + B2*D(n-2)$ 
;  $D(n) = X(n) - A1*D(n-1) - A2*D(n-2)$ 
; where X(n) = input sample, Y(n) = output of filter
; and A1, A2, B0, B1, B2 are the Filter Co-efficients
;
; The above difference equations are only for 1 section of a
; 2nd order Direct_Form II Filter structure (IIR)
;
; NOTE :
; It is possible to design the above structures
; such that the co-efficients B0 = B2. If this is the
; case,
;  $Y(n) = B0*[D(n) + D(n-2)] = B2*[D(n) + D(n-2)]$ 
; This way, one multiplication can be avoided
;
; If a 4th order filter is to be implemented, the output of
; the 1st structure should be input to the 2nd cascade section
;
; Timing (WORST CASE) :
;
; 59+4*179 = 775 Cycles
; (194 uS @ 16 Mhz)
;
; Program Memory :
;
; 63 locations
;
;*****
; The sample filters are desined so that B0=B2
; This saves 1 multiplication
;
0001 B0_EQUALS_B2 equ TRUE
;
;*****
; Parameters to BIQUAD Macro :
; Filter Constants A1, A2, B0, B1, B2
; & D(n), D(n-1), D(n-2), filter stage #
;
BIQUAD MACRO Ax1,Ax2,Bx0,Bx1,Dn,Dn_1,Dn_2,stage
;
; Compute Ax2*D(n-2)
```

Implementing IIR Digital Filters

```

;
MOVFP16 Dn_2,AARG ; D(n-2) = multiplier
MOVFP16 Ax2,BARG ; A2 = multiplicand
call DblMult ; (ACCd,ACCc) = A2*D(n-2)
;
; Add product to output of 1st section
; Save result in 32 bit Accumulator
;
ADD32 DPX,ACC
;
; Compute A1*D(n-1)
;
MOVFP16 Dn_1,AARG ; AARG = D(n-2) = multiplier
MOVFP16 Ax1,BARG ; BARG = A2 = multiplicand
call DblMult ; (ACCd,ACCc) = A1*D(n-1)
;
; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
; multiplications already done, so simply perform a 32 bit add
; of previously obtained multiplication results
;
ADD32 DPX,ACC ; ACC = A1*D(n-1) + A2*D(n-2) + (output of 1st
;
; save the upper 16 bits of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication
;
rlcf ACC+B1,w
rlcf ACC+B2,w
movwf Dn
rlcf ACC+B3,w ; decimal adjust ( mult by 2)
movwf Dn+B1
;
; Compute B2 * [D(n) + D(n-2)]
;
if B0_EQUALS_B2
ADD16ACC Dn_2,Dn,AARG ; AARG = Dn + D(n-2) = multiplier
MOVFP16 Bx0,BARG ; BARG = A2 = multiplicand
call DblMult ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
MOVFP32 DPX,ACC
else
MOVFP16 Bx0,BARG
MOVFP16 Dn,AARG
call DblMult ; B0*D(n)
MOVFP32 DPX,ACC
MOVFP16 Bx2,BARG
MOVFP16 Dn_2,AARG
call DblMult ; B2*D(n-2)
ADD32 DPX,ACC
endif
;
; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
; This way in the next iteration D(n-2) is equal to the present D(n-1)
;
movfp Dn_1,AARG+B0
movfp AARG+B0,Dn_2 ; Shift down D(n-1)
movfp Dn_1+B1,AARG+B1
movfp AARG+B1,Dn_2+B1 ; AARG = D(n-1) = multiplier
MOVFP16 Bx1,BARG ; BARG = B1 = multiplicand
call DblMult ; (ACCd,ACCc) = B1*D(n-1)
;
; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
; = B1*D(n-1) + B0*[D(n) + D(n-2)]

```

Implementing IIR Digital Filters

```
; Since all multiplications are already done, simply perform a
; 32 bit addition
;
ADD32   DPX,ACC           ; ACC = B1*D(n-1) + B2*D(n-2) + B0*D(n)
;
; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;
MOV16   Dn,Dn_1          ; Shift down D(n) to D(n-1)
;
ENDM

;
;*****
;                               Second Order Direct Form IIR Filter
;
;
; In the code given below, a 4th order IIR Elliptic Lowpass Filter
; is implemented. Other order filters may be implemented by
; taking the following example code as a basis.
;
; The specifications of the filter are :
;
; Sampling Frequency = 2.0 Khz
;
; Filter Type - 4th Order Elliptic Lowpass Filter
;
;
; Lower Band Edge           Band1           Band2
; Upper Band Edge           500 Hz           1 Khz
; Nominal Gain               1.0             0.0
; Nominal Ripple             0.01            0.05
; Maximum Ripple             0.00906         0.04601
; Ripple in dB               0.07830         -26.75
;
; The Filter Co-efficients for the above specifications
; of the filter are computed as follows :
;
; 1st Section :
;
; A11 = -0.133331
; A12 = 0.167145
; B10 = 0.285431
; B11 = 0.462921
; B12 = 0.285431
;
; 2nd Section
;
; A21 = 0.147827
; A22 = 0.765900
; B20 = 0.698273
; B21 = 0.499908
; B22 = 0.698273
;
; Performance (WORST Case):
;
; Cycles = #of Filter Stages*775 + 16
;          = 2*775+16 = 1566 Cycles
;          ( 391 uSec)
;          per each sample. Initialization
;          time after reset is not counted
;          Timing measured with B0_EQUALS_B2
;          set to TRUE (see BIQUAD Macro for
;          explanation
;
; Program Memory :
;
;          = 16+ # of FilterStages * (BIQUAD
;          + filter co-efficients)
;          + multiplier
;          = 16+2*(63+5)+274 = 421 locations
;          (excluding initialization)
```


Implementing IIR Digital Filters

```

;
;
;          RAM usage = 48 file registers
;          RAM usage/each additional stage = 16 file regs
;
;
;          This time is less than 2 Khz (500 uSec),
;          which means real time filtering is possible
;
;*****
;*****
;
;
;          CBLOCK 0
0000 0004      B0,B1,B2,B3
;          ENDC
;*****
;
;          CBLOCK 0x18
0018 0004      DPX,DPX1,DPX2,DPX3          ; arithmetic accumulator
001C 0004      AARG,AARG1,BARG,BARG1      ; multiply arguments
;          ENDC
;
;          CBLOCK
0020 0002      Dn1, Dn1_Hi
0022 0002      Dn1_1, Dn1_1_Hi
0024 0002      Dn1_2, Dn1_2_Hi
0026 0000
0026 0002      Dn2, Dn2_Hi
0028 0002      Dn2_1, Dn2_1_Hi
002A 0002      Dn2_2, Dn2_2_Hi
;          ENDC

;          CBLOCK
002C 0002      A11, A11_Hi
002E 0002      A12, A12_Hi
0030 0002      B10, B10_Hi
0032 0002      B11, B11_Hi          ; 1st Section Filter Co-efficients
0034 0002      B12, B12_Hi
0036 0000
0036 0002      A21, A21_Hi
0038 0002      A22, A22_Hi
003A 0002      B20, B20_Hi
003C 0002      B21, B21_Hi          ; 2nd Section Filter Co-efficients
003E 0002      B22, B22_Hi
;          ENDC

;          CBLOCK
0040 0002      X, X1          ; 16 bits of input stream
0042 0002      Y, Y1          ; 16 bits of filter output
0044 0000
0044 0004      ACC, ACC1, ACC2, ACC3      ; 32 bit accumulator for computations
;          ENDC

;          FltStage      .set      2
000A      NumCoeff      equ      (5*FltStage) ; 5 Co-eff per stage
;
;          LPASS      .set      TRUE
0000      HPASS      .set      FALSE
0000      BPASS      .set      FALSE          ; select the desired filter type
0000      BSTOP      .set      FALSE
;
;          SIGNED      equ      TRUE          ; Set This To 'TRUE' for signed multi
;          ; and 'FALSE' for unsigned.
;
;*****
;          Test Program For Low Pass Filter
;*****

```

Implementing IIR Digital Filters

```

                ORG     0x0000
;
start
0000 E00D      call    initFilter
0001 B000      movlw   0x00
0002 0140      movwf  X
0003 B07F      movlw   0x7f          ; set initial Xn = X(0) = 0x7f00
0004 0141      movwf  X+B1        ; test for impulse response
;
NextPoint
0005 E022      call    IIR_Filter
0006 A442      tltwt   _LOW,Y
tracePoint
0007 AE43      tablwt  _HIGH,0,Y+B1
0008 0000      nop
0009 2940      clrfs  X          ; set X(n) = 0 , n <> 0
000A 2941      clrfs  X+B1        ; for simulating an Impulse
000B C005      goto   NextPoint
;
000C C00C      self   goto   self
;
;*****
initFilter
;
; At first read the Filter Co-efficients from Prog. Mem to Data RAM
;
        if     LPASS
000D B0C1      movlw   _coeff_lpass
000E 010D      movwf  tblptrl
000F B001      movlw   page _coeff_lpass
0010 010E      movwf  tblptrh
        endif
        if     HPASS
        movlw   _coeff_hpass
        movwf  tblptrl
        movlw   page _coeff_hpass
        movwf  tblptrh
        endif
        if     BPASS
        movlw   _coeff_bpass
        movwf  tblptrl
        movlw   page _coeff_bpass
        movwf  tblptrh
        endif
        if     BSTOP
        movlw   _coeff_bstop
        movwf  tblptrl
        movlw   page _coeff_bstop
        movwf  tblptrh
        endif
;
0011 B02C      movlw   All
0012 0101      movwf  fsr0
0013 8404      bsf    _fs0
0014 8D04      bcf    _fsl          ; auto increment
;
; Read Filter Co-efficients from Program Memory
;
0015 B00A      movlw   NumCoeff
0016 A92C      tablrld _LOW,_INC,All        ; garbage
NextCoeff
0017 A000      tlrld  _LOW,indf0
0018 AB00      tablrld _HIGH,_INC,indf0
0019 1700      decfsz wreg
001A C017      goto   NextCoeff

```

Implementing IIR Digital Filters

```

;
; Initialize "Dn"s to zero
;
001B B020          movlw   Dn1
001C 0101          movwf   fsr0
001D B00C          movlw   6*FltStage
NextClr
001E 2900          clrfsz  indf0
001F 1700          decfsz  wreg
0020 C01E          goto    NextClr
;
0021 0002          return
;
;*****
;                      1st Cascade Section
;*****
;
IIR_Filter
;
; Compute  $D(n) = X(n) + A1*D(n-1) + A2*D(n-2)$ 
; Since the filter constants are computed in Q15 format,
;  $X(n)$  must be multiplied by  $2^{**15}$  and then added to the
; other terms.
;
; Move Input to accumulator after proper scaling
;
0022 8804          bcf     _carry
0023 1941          rrcf    X+B1
0024 1940          rrcf    X
0025 2900          clrfsz  wreg          ; Scale the input X
0026 1900          rrcf    wreg
0027 0145          movwf   ACC+B1
0028 6040          movf    X,wreg
0029 0146          movwf   ACC+B2
002A 6041          movf    X+B1,wreg
002B 0147          movwf   ACC+B3          ; ACC = scaled input :  $X*(2^{**15})$ 
;
; 1st Biquad filter section
;
BIQUAD A11,A12,B10,B11,Dn1,Dn1_1,Dn1_2,1
;
; Compute  $A2*D(n-2)$ 
;
002C 7C24          MOVFP   Dn1_2+B0,AARG+B0          ; move Dn1_2(B0) to AARG(B0)
002D 7D25          MOVFP   Dn1_2+B1,AARG+B1          ; move Dn1_2(B1) to AARG(B1)
;
002E 7E2E          MOVFP   A12+B0,BARG+B0          ; move A12(B0) to BARG(B0)
002F 7F2F          MOVFP   A12+B1,BARG+B1          ; move A12(B1) to BARG(B1)
;
0030 E0AF          call    DblMult          ; (ACCd,ACCc) =  $A2*D(n-2)$ 
;
; Add product to output of 1st section
; Save result in 32 bit Accumulator
;
0031 6018          MOVFP   DPX+B0,WREG          ; get lowest byte of DPX into w
0032 0F44          ADDWF   ACC+B0          ; add lowest byte of ACC, save in

```

Implementing IIR Digital Filters

```
ACC(B0)
0033 6019      MOVFDP DPX+B1,WREG      ; get 2nd byte of DPX into w
0034 1145      ADDWFC ACC+B1      ; add 2nd byte of ACC, save in
ACC(B1)
0035 601A      MOVFDP DPX+B2,WREG      ; get 3rd byte of DPX into w
0036 1146      ADDWFC ACC+B2      ; add 3rd byte of ACC, save in
ACC(B2)
0037 601B      MOVFDP DPX+B3,WREG      ; get 4th byte of DPX into w
0038 1147      ADDWFC ACC+B3      ; add 4th byte of ACC, save in
ACC(B3)

;
; Compute A1*D(n-1)
;

0039 7C22      MOVFDP Dn1_1+B0,AARG+B0    ; move Dn1_1(B0) to AARG(B0)
003A 7D23      MOVFDP Dn1_1+B1,AARG+B1    ; move Dn1_1(B1) to AARG(B1)

003B 7E2C      MOVFDP A11+B0,BARG+B0    ; move A11(B0) to BARG(B0)
003C 7F2D      MOVFDP A11+B1,BARG+B1    ; move A11(B1) to BARG(B1)

003D E0AF      call   DblMult      ; (ACCd,ACCc) = A1*D(n-1)
;
; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
; multiplications already done, so simply perform a 32 bit add
; of previously obtained multiplication results
;

003E 6018      MOVFDP DPX+B0,WREG      ; get lowest byte of DPX into w
003F 0F44      ADDWFC ACC+B0      ; add lowest byte of ACC, save in
ACC(B0)
0040 6019      MOVFDP DPX+B1,WREG      ; get 2nd byte of DPX into w
0041 1145      ADDWFC ACC+B1      ; add 2nd byte of ACC, save in
ACC(B1)
0042 601A      MOVFDP DPX+B2,WREG      ; get 3rd byte of DPX into w
0043 1146      ADDWFC ACC+B2      ; add 3rd byte of ACC, save in
ACC(B2)
0044 601B      MOVFDP DPX+B3,WREG      ; get 4th byte of DPX into w
0045 1147      ADDWFC ACC+B3      ; add 4th byte of ACC, save in
ACC(B3)

;
;
; save the upper 16 bits of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication
;

0046 1A45      rlcfc  ACC+B1,w
0047 1A46      rlcfc  ACC+B2,w
0048 0120      movwfc Dn1
0049 1A47      rlcfc  ACC+B3,w      ; decimal adjust ( mult by 2)
004A 0121      movwfc Dn1+B1

;
; Compute B2 * [D(n) + D(n-2)]
;

if B0_EQUALS_B2

004B 6024      movwfc Dn1_2+B0,wreg
004C 0E20      addwfc Dn1+B0,w
004D 011C      movwfc AARG+B0
004E 6025      movwfc Dn1_2+B1,wreg
004F 1021      addwfc Dn1+B1,w
0050 011D      movwfc AARG+B1
```

Implementing IIR Digital Filters

```

0051 7E30          MOVFP  B10+B0,BARG+B0          ; move B10 (B0) to BARG (B0)
0052 7F31          MOVFP  B10+B1,BARG+B1          ; move B10 (B1) to BARG (B1)

0053 E0AF          call    DblMult                ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]

0054 5844          MOVFP  DPX+B0,ACC+B0          ; move DPX (B0) to ACC (B0)
0055 5945          MOVFP  DPX+B1,ACC+B1          ; move DPX (B1) to ACC (B1)
0056 5A46          MOVFP  DPX+B2,ACC+B2          ; move DPX (B2) to ACC (B2)
0057 5B47          MOVFP  DPX+B3,ACC+B3          ; move DPX (B3) to ACC (B3)

                else

                MOVFP16 B10,BARG
                MOVFP16 Dn1,AARG

                call    DblMult                ; B0*D(n)

                MOVFP32 DPX,ACC

                MOVFP16 Bx2,BARG

                MOVFP16 Dn1_2,AARG

                call    DblMult                ; B2*D(n-2)

                ADD32  DPX,ACC

                endif

                ;
                ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                ;
0058 7C22          movfp  Dn1_1,AARG+B0
0059 5C24          movfp  AARG+B0,Dn1_2          ; Shift down D(n-1)
005A 7D23          movfp  Dn1_1+B1,AARG+B1
005B 5D25          movfp  AARG+B1,Dn1_2+B1        ; AARG = D(n-1) = multiplier

005C 7E32          MOVFP  B11+B0,BARG+B0          ; move B11 (B0) to BARG (B0)
005D 7F33          MOVFP  B11+B1,BARG+B1          ; move B11 (B1) to BARG (B1)

005E E0AF          call    DblMult                ; (ACCd,ACCc) = B1*D(n-1)

                ;
                ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                ; = B1*D(n-1) + B0*[D(n) + D(n-2)]
                ; Since all multiplications are already done, simply perform a
                ; 32 bit addition
                ;

005F 6018          MOVFP  DPX+B0,WREG          ; get lowest byte of DPX into w
0060 0F44          ADDWF  ACC+B0                ; add lowest byte of ACC, save in
ACC (B0)
0061 6019          MOVFP  DPX+B1,WREG          ; get 2nd byte of DPX into w
0062 1145          ADDWFC ACC+B1                ; add 2nd byte of ACC, save in

```

Implementing IIR Digital Filters

```
ACC(B1)
0063 601A      MOVFP  DPX+B2,WREG      ; get 3rd byte of DPX into w
0064 1146      ADDWFC ACC+B2           ; add 3rd byte of ACC, save in
ACC(B2)
0065 601B      MOVFP  DPX+B3,WREG      ; get 4th byte of DPX into w
0066 1147      ADDWFC ACC+B3           ; add 4th byte of ACC, save in
ACC(B3)

;
; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;

0067 6020      MOVFP  Dn1+B0,WREG      ; get byte of Dn1 into w
0068 0122      MOVWF  Dn1_1+B0        ; move to Dn1_1(B0)
0069 6021      MOVFP  Dn1+B1,WREG      ; get byte of Dn1 into w
006A 0123      MOVWF  Dn1_1+B1        ; move to Dn1_1(B1)

;

;
; 2nd Biquad filter section
;
BIQUAD  A21,A22,B20,B21,Dn2,Dn2_1,Dn2_2,2

;
; Compute A22*D(n-2)
;

006B 7C2A      MOVFP  Dn2_2+B0,AARG+B0    ; move Dn2_2(B0) to AARG(B0)
006C 7D2B      MOVFP  Dn2_2+B1,AARG+B1    ; move Dn2_2(B1) to AARG(B1)

;

006D 7E38      MOVFP  A22+B0,BARG+B0    ; move A22(B0) to BARG(B0)
006E 7F39      MOVFP  A22+B1,BARG+B1    ; move A22(B1) to BARG(B1)

006F E0AF      call   DblMult                ; (ACCd,ACCc) = A2*D(n-2)

;
; Add product to output of 1st section
; Save result in 32 bit Accumulator
;

0070 6018      MOVFP  DPX+B0,WREG      ; get lowest byte of DPX into w
0071 0F44      ADDWF  ACC+B0           ; add lowest byte of ACC, save in
ACC(B0)
0072 6019      MOVFP  DPX+B1,WREG      ; get 2nd byte of DPX into w
0073 1145      ADDWFC ACC+B1           ; add 2nd byte of ACC, save in
ACC(B1)
0074 601A      MOVFP  DPX+B2,WREG      ; get 3rd byte of DPX into w
0075 1146      ADDWFC ACC+B2           ; add 3rd byte of ACC, save in
ACC(B2)
0076 601B      MOVFP  DPX+B3,WREG      ; get 4th byte of DPX into w
0077 1147      ADDWFC ACC+B3           ; add 4th byte of ACC, save in
ACC(B3)

;
; Compute A1*D(n-1)
;

0078 7C28      MOVFP  Dn2_1+B0,AARG+B0    ; move Dn2_1(B0) to AARG(B0)
0079 7D29      MOVFP  Dn2_1+B1,AARG+B1    ; move Dn2_1(B1) to AARG(B1)
```

Implementing IIR Digital Filters

```

007A 7E36          MOVFP  A21+B0,BARG+B0          ; move A21(B0) to BARG(B0)
007B 7F37          MOVFP  A21+B1,BARG+B1          ; move A21(B1) to BARG(B1)

007C E0AF          call   DblMult                ; (ACCd,ACCc) = A1*D(n-1)
;
; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
; multiplications already done, so simply perform a 32 bit add
; of previously obtained multiplication results
;

007D 6018          MOVFP  DPX+B0,WREG                ; get lowest byte of DPX into w
007E 0F44          ADDWF  ACC+B0                    ; add lowest byte of ACC, save in
ACC(B0)
007F 6019          MOVFP  DPX+B1,WREG                ; get 2nd byte of DPX into w
0080 1145          ADDWFC ACC+B1                    ; add 2nd byte of ACC, save in
ACC(B1)
0081 601A          MOVFP  DPX+B2,WREG                ; get 3rd byte of DPX into w
0082 1146          ADDWFC ACC+B2                    ; add 3rd byte of ACC, save in
ACC(B2)
0083 601B          MOVFP  DPX+B3,WREG                ; get 4th byte of DPX into w
0084 1147          ADDWFC ACC+B3                    ; add 4th byte of ACC, save in
ACC(B3)

;
;
; save the upper 16 bits of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication
;
0085 1A45          rlcfc  ACC+B1,w
0086 1A46          rlcfc  ACC+B2,w
0087 0126          movwf  Dn2
0088 1A47          rlcfc  ACC+B3,w                ; decimal adjust (mult by 2)
0089 0127          movwf  Dn2+B1

;
; Compute B2 * [D(n) + D(n-2)]
;
        if      B0_EQUALS_B2

008A 602A          movfpl Dn2_2+B0,wreg
008B 0E26          addwfl Dn2+B0,w
008C 011C          movwfl AARG+B0
008D 602B          movfpl Dn2_2+B1,wreg
008E 1027          addwfl Dn2+B1,w
008F 011D          movwfl AARG+B1

0090 7E3A          MOVFP  B20+B0,BARG+B0          ; move B20(B0) to BARG(B0)
0091 7F3B          MOVFP  B20+B1,BARG+B1          ; move B20(B1) to BARG(B1)

0092 E0AF          call   DblMult                ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]

0093 5844          MOVFP  DPX+B0,ACC+B0          ; move DPX(B0) to ACC(B0)
0094 5945          MOVFP  DPX+B1,ACC+B1          ; move DPX(B1) to ACC(B1)
0095 5A46          MOVFP  DPX+B2,ACC+B2          ; move DPX(B2) to ACC(B2)
0096 5B47          MOVFP  DPX+B3,ACC+B3          ; move DPX(B3) to ACC(B3)

        else

MOVFP16 B20,BARG

```

Implementing IIR Digital Filters

```

MOVFP16 Dn2,AARG

call   DblMult           ; B0*D(n)

MOVFP32 DPX,ACC

MOVFP16 Bx2,BARG
MOVFP16 Dn2_2,AARG
call   DblMult           ; B2*D(n-2)
ADD32  DPX,ACC

endif

;
; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
; This way in the next iteration D(n-2) is equal to the present D(n-1)
;
0097 7C28      movfp   Dn2_1,AARG+B0
0098 5C2A      movfpf  AARG+B0,Dn2_2           ; Shift down D(n-1)
0099 7D29      movfpf  Dn2_1+B1,AARG+B1
009A 5D2B      movfpf  AARG+B1,Dn2_2+B1       ; AARG = D(n-1) = multiplier

009B 7E3C      MOVFP   B21+B0,BARG+B0         ; move B21(B0) to BARG(B0)
009C 7F3D      MOVFP   B21+B1,BARG+B1         ; move B21(B1) to BARG(B1)

009D E0AF      call    DblMult           ; (ACCd,ACCc) = B1*D(n-1)
;
; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
;                   = B1*D(n-1) + B0*[D(n) + D(n-2)]
; Since all multiplications are already done, simply perform a
; 32 bit addition
;

009E 6018      MOVFP   DPX+B0,WREG           ; get lowest byte of DPX into w
009F 0F44      ADDWF   ACC+B0                 ; add lowest byte of ACC, save in
ACC(B0)
00A0 6019      MOVFP   DPX+B1,WREG           ; get 2nd byte of DPX into w
00A1 1145      ADDWF   ACC+B1                 ; add 2nd byte of ACC, save in
ACC(B1)
00A2 601A      MOVFP   DPX+B2,WREG           ; get 3rd byte of DPX into w
00A3 1146      ADDWF   ACC+B2                 ; add 3rd byte of ACC, save in
ACC(B2)
00A4 601B      MOVFP   DPX+B3,WREG           ; get 4th byte of DPX into w
00A5 1147      ADDWF   ACC+B3                 ; add 4th byte of ACC, save in
ACC(B3)

;
; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;

00A6 6026      MOVFP   Dn2+B0,WREG           ; get byte of Dn2 into w
00A7 0128      MOVWF   Dn2_1+B0              ; move to Dn2_1(B0)
00A8 6027      MOVFP   Dn2+B1,WREG           ; get byte of Dn2 into w
00A9 0129      MOVWF   Dn2_1+B1              ; move to Dn2_1(B1)

;

```


Implementing IIR Digital Filters

```

;
; The filter output is now computed
; Save the Upper 16 Bits of 32 bit Accumulator into Y after
; adjusting the decimal point
;
MOV16   ACC+B2,Y

00AA 6046      MOVFP   ACC+B2+B0,WREG      ; get byte of ACC+B2 into w
00AB 0142      MOVWF   Y+B0              ; move to Y(B0)
00AC 6047      MOVFP   ACC+B2+B1,WREG      ; get byte of ACC+B2 into w
00AD 0143      MOVWF   Y+B1              ; move to Y(B1)

;
00AE 0002      ;           return           ; Output Y(n) computed
;
;*****
; Set SIGNED/UNSIGNED Flag Before Including 17c42MPY.mac
;
include       "17c42MPY.mac"

;*****
; Low Pass Filter Co-efficients
;
;
;           ELLIPTIC   LOWPASS FILTER
;
;           FILTER ORDER =      4
;           Sampling frequency =  2.000 KiloHertz
;
;           BAND 1      BAND 2
;
; LOWER BAND EDGE      .00000      .60000
; UPPER BAND EDGE      .50000      1.00000
; NOMINAL GAIN          1.00000      .00000
; NOMINAL RIPPLE        .01000      .05000
; MAXIMUM RIPPLE        .00906      .04601
; RIPPLE IN dB          .07830      -26.74235
;
; I           A(I,1)      A(I,2)      B(I,0)      B(I,1)      B(I,2)
;
; 1           -.133331     .167145     .285431     .462921     .285431
; 2           .147827     .765900     .698273     .499908     .698273
;
;_coeff_lpass      ; co-efficients for 1st Cascade
01C1 1111      data      4369      ; -A11
01C2 EA9B      data      -5477     ; -A12
01C3 2489      data      9353      ; B10
01C4 3B41      data      15169     ; B11
01C5 2489      data      9353      ; B12
;
;           ; co-efficients for 2nd Cascade
01C6 ED14      data      -4844     ; -A21
01C7 9DF7      data      -25097    ; -A22
01C8 5961      data      22881     ; B20
01C9 3FFD      data      16381     ; B21
01CA 5961      data      22881     ; B22
;
;*****
;
;
; High Pass Filter Co-efficients
;
;

```

Implementing IIR Digital Filters

```

;
;           ELLIPTIC   HIGHPASS FILTER
;
;           FILTER ORDER =   4
;           Sampling frequency =   2.000 KiloHertz
;
;           BAND 1      BAND 2
;
;
; LOWER BAND EDGE      .00000      .50000
; UPPER BAND EDGE      .40000      1.00000
; NOMINAL GAIN          .00000      1.00000
; NOMINAL RIPPLE        .04000      .02000
; MAXIMUM RIPPLE        .03368      .01686
; RIPPLE IN dB          -29.45335    .14526
;
; I           A(I,1)      A(I,2)      B(I,0)      B(I,1)      B(I,2)
;
; 1           .276886     .195648     .253677     -.411407     .253677
; 2           -.094299    .780396     .678650     -.485840     .678650
;
;
;_coeff_hpass          ; co-efficients for 1st Cascade section
01CB DC8F             data      -9073          ; -A11
01CC E6F5             data      -6411          ; -A12
01CD 2079             data      8313           ; B10
01CE CB57             data      -13481         ; B11
01CF 2079             data      8313           ; B12
;
;                               ; co-efficients for 2nd Cascade section
01D0 0C12             data      3090           ; -A21
01D1 9C1C             data      -25572          ; -A22
01D2 56DE             data      22238          ; B20
01D3 C1D0             data      -15920          ; B21
01D4 56DE             data      22238          ; B22
;
;*****
;
;*****
; Band Pass Filter Co-efficients
;
;
;           ELLIPTIC   BANDPASS FILTER
;
;           FILTER ORDER =   4
;           Sampling frequency =   2.000 KiloHertz
;
;           BAND 1      BAND 2      BAND 3
;
;
; LOWER BAND EDGE      .00000      .30000      .90000
; UPPER BAND EDGE      .10000      .70000      1.00000
; NOMINAL GAIN          .00000      1.00000     .00000
; NOMINAL RIPPLE        .05000      .05000      .05000
; MAXIMUM RIPPLE        .03644      .03867      .03641
; RIPPLE IN dB          -28.76779     .32956     -28.77647
;
;
; I           A(I,1)      A(I,2)      B(I,0)      B(I,1)      B(I,2)
;
; 1           -.936676     .550568     .444000     -.865173     .444000
; 2           .936707     .550568     .615540     1.199402     .615540
;
;
;_coeff_bpss          ; co-efficients for 1st Cascade section
01D5 3BF2             data      30693/2         ; -A11
01D6 DCC4             data      -18041/2        ; -A12
01D7 1C6A             data      14549/2         ; B10
01D8 C8A1             data      -28350/2        ; B11
01D9 1C6A             data      14549/2         ; B12
;
;                               ; co-efficients for 2nd Cascade section

```

Implementing IIR Digital Filters

```

01DA C40D          data    -30694/2      ; -A21
01DB DCC4          data    -18041/2      ; -A22
01DC 2765          data     20170/2      ; B20
01DD 4CC3          data     39302/2      ; B21
01DE 2765          data     20170/2      ; B22
;
;*****
;
;*****
;   Band Stop Filter Co-efficients
;
;
;
;           ELLIPTIC   BANDSTOP FILTER
;
;   FILTER ORDER =      4
;   Sampling frequency =  2.000 KiloHertz
;
;           BAND  1          BAND  2          BAND  3
;
;
;LOWER BAND EDGE          .00000          .45000          .70000
;UPPER BAND EDGE          .30000          .55000          1.00000
;NOMINAL GAIN              1.00000          .00000          1.00000
;NOMINAL RIPPLE            .05000          .05000          .05000
;MAXIMUM RIPPLE            .03516          .03241          .03517
;RIPPLE IN dB              .30015          -29.78523         .30027
;
;
;   I           A(I,1)      A(I,2)      B(I,0)      B(I,1)      B(I,2)
;
;   1           .749420     .583282      .392685     .087936     .392685
;   2           -.749390     .583282     1.210022     -2.270935     1.210022
;
;_coeff_bstop          ; co-efficients for 1st Cascade section
;   data          -24557/2      ; -A11
01E0 DAAC          data          -19113/2      ; -A12
01E1 1922          data          12868/2      ; B10
01E2 05A0          data          2881/2      ; B11
01E3 1922          data          12868/2      ; B12
;   ; co-efficients for 2nd Cascade section
01E4 2FF6          data          24556/2      ; -A21
01E5 DAAC          data          -19113/2      ; -A22
01E6 4D71          data          39650/2      ; B20
01E7 EEA9          data          -8878/2      ; B21
01E8 4D71          data          39650/2      ; B22
;
;*****

```

END

```

Errors   :    0
Warnings :

```



Implementing IIR Digital Filters

NOTES:

Implementation of Fast Fourier Transforms

INTRODUCTION

Fourier transforms are one of the fundamental operations in signal processing. In digital computations, Discrete Fourier Transforms (DFT) are used to describe, represent and analyze discrete-time signals. However, direct implementation of DFT is computationally very inefficient. Of the various available high speed algorithms to compute DFT, the Cooley-Tukey algorithm is the simplest and most commonly used. These efficient algorithms to compute DFTs are called Fast Fourier Transforms (FFTs).

This application note provides the source code to compute the FFT using PIC17C42. The theory behind the FFT algorithms is well established and described in the literature and hence not described in this application note. A Radix-2 Cooley-Tukey FFT is implemented with no limits on the length of FFT. The length is only limited by the amount of available program memory space. All computations are done using double precision arithmetic.

IMPLEMENTATION

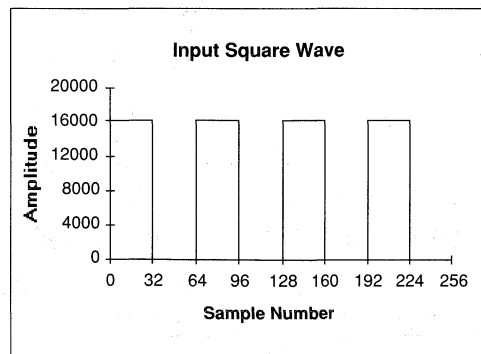
Since the PIC17C42 has only 232 x 8 general purpose RAM (equivalent of 116 x 16), at most a 32 point FFT (16-bit REAL & IMAGINARY data) can be implemented using on chip RAM. To compute higher point FFT, the data is stored in program memory space of PIC17C42. The PIC17C42 has instructions (`TABLRD` & `TABLWT`) to transfer data between program memory space and on chip file registers. In extended microcontroller mode, the PIC17C42 has 2K x 16 (0000h:07FFh) on chip program memory space and 62K x 16 (0800h:0FFFFh) of external program memory space. In this mode, the code (in this case, the FFT code) may reside on the on chip EPROM and the data to be analyzed may be stored in external RAMs (up to 62K). A suggested method of connecting external RAMs (appropriate EEPROMs may also be used) is shown in Figure 3.

If PIC17C42 is used in extended microcontroller mode and if all the code resides on chip, then the cost may further be reduced by using only one external SRAM instead of two. The block diagram is shown in Figure 2. The 16-bit data stored in the external RAM is organized as low byte followed by high byte. To achieve this, the code presented in this application note needs minor modifications, especially where `TABLRD` and `TABLWR` instructions are used. Address indexing must be incremented by two since 2 reads/writes must be performed to access a 16-bit data.

The FFT is implemented with Decimation In Frequency. Thus the input data before calling the FFT routine ("R2FFT") should be in normal order and the transformed data is in scrambled order. The original data is overwritten by the transformed data to conserve memory. This is achieved by use of in-place calculations. This in-place calculations causes the order of the DFT terms to be permuted. So at the end of the transform, all the data needs to be unscrambled to get the right order of the DFT terms. In some applications the order of terms is not necessary. Keeping this in mind, the unscrambling code is written as a separate subroutine ("Unscramble") and may be called if necessary.

Before implementing the FFT using PIC17C42, a "C" program was written and tested. This high level programming helps in writing the assembly code and the results of both programs can be compared against while debugging the assembly code. The "C" source code for the Radix-2 FFT is shown in Appendix A. The assembly code source file of the FFT program is shown in Appendix B. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note ANw00540.

FIGURE 1 - TEST WAVE FORM



Implementing FFT

TESTING

The assembly code was developed and debugged using Microchip's PICMASTER In-Circuit Emulator System. A main program generates a test pattern (like a square wave) and calls the FFT routines "R2FFT" & "Unscramble". After the DFT terms are computed, the results are captured into PICMASTER's real time trace buffer by putting a trace point on a dummy TABLRD instruction and capturing only the 2nd cycle (data cycle of TABLRD) of the instruction. The data from Trace buffer was hot linked to Microsoft Excel using DDE and then the graphs were plotted and analyzed.

The code was tested on various wave forms (a rectangular pulse, a triangular wave, square wave and a sine wave) and using FFT lengths of 64, 256 & 1024. The results of a 256 Point FFT on a square wave is shown below. The test wave form is shown in Figure 1 and the frequency spectrum of it computed by PIC17C42 is shown in Figure 2. As expected, the spectra appears at the odd harmonics of the input wave form's fundamental frequency (At $N*256/64$, $N = 0,1,3,5,\dots$).

PERFORMANCE

The performance of FFT using PIC17C42 is quite impressive noting that for an 8 bit machine with no hardware multiplier. Also note that all computations are performed using double precision arithmetic (16- & 32- bits) which is the case in most of the low end DSPs. Table 1 provides the real time performance in total number of Instruction cycles for both the "R2FFT" & "Unscramble" routines using 64, 256 & 1024 Point FFT. Note that the timings are in a worst case situation and in general will be a lot better than shown in the table. The worst case situation arises because the 16 x 16 software multiplier ("DbMult") does not have a uniform timing and depends on the input data. The worst case timing of the multiplier is used in the computation of performance.

FIGURE 2 - FFT (MAGNITUDE SPECTRUM) OF FIGURE 1 COMPUTED BY PIC17C42

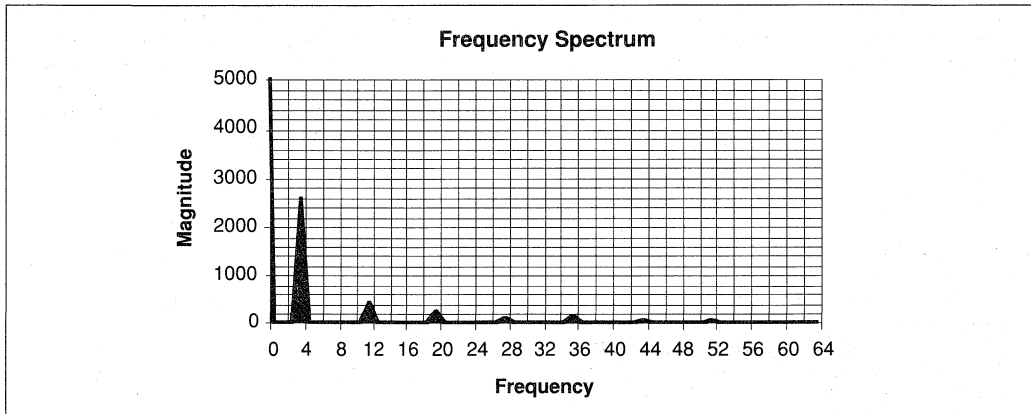


TABLE 1 - WORST CASE PERFORMANCE OF FFT IN INSTRUCTION CYCLES AND REAL TIME @ 16 MHZ

N (FFT Length)	64 Point	256 Point	1024 Point
R2FFT	34116 + 768*Mult = 171588	178024 + 4096*Mult= 911208	878752 + 20480*Mult = 4544672
Unscramble	5143	22495	93525
Total	176731 (44.18 mS)	933703 (233.42 mS)	4638197 (1159.55 mS)

Table 2 shows the Program Memory & Data RAM requirements for an N Point FFT. The multiplier routine and other general purpose macro requirements are included in the memory requirements. The speed performance for the square wave test data differs from the Table 1 since worst case timings is not used and reflects a more reasonable data.

FFT APPLICATIONS

Although the FFT does not find a place in many microcontroller applications, it is very useful in providing a benchmark of the processor. As can be seen from Table 2, the performance is very satisfactory, considering the fact that PIC17C42 is a Microcontroller and not a DSP. Also it should be borne in mind that all computations are performed in 16/32 bit arithmetic and that PIC17C42 is a low cost 8 bit machine unlike the DSPs which are relatively expensive.

In applications like Instrumentation, where real time FFT computation is not required, PIC17C42 can be used as a single chip solution instead of a Microcontroller and a Digital Signal Processor.

Suggested Reading :

- [1] L. R. Rabiner and B. Gold, Theory and Application Of Digital Signal Processing.
Englewood Cliffs, NJ : Prentice-Hall, 1975.
- [2] C. S. Burrs and T. W. Parks, DFT/FFT and Convolution Algorithms.
New York : Wiley, 1985.
- [3] Jeffrey J. Rodriguez, An Improved FFT Digit-Reversal Algorithm.
IEEE Transactions On Acoustics, Speech, And Signal Processing,
Vol. 37, No. 8, Aug 1989.

*Author: Amar Palacherla
Logic Products Division*

TABLE 2 - REQUIREMENTS FOR RADIX-2 FFT

N (FFT Length)	64 Point	256 Point	1024 Point
Code Space (locations)	$603 + 0.75 * N = 651$	$603 + 0.75 * N = 795$	$603 + 0.75 * N = 1371$
Data Storage in Program Memory Space	$2 * N = 128$	$2 * N = 512$	$2 * N = 2048$
Scratch RAM	49	49	49
Performance (Square Wave Input Data)	122384 (30.6 mS)	644416 (161.1 mS)	3192176 (798 mS)

FIGURE 3 - EXTERNAL MEMORY CONNECTION

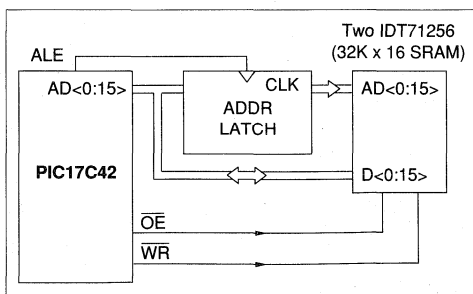
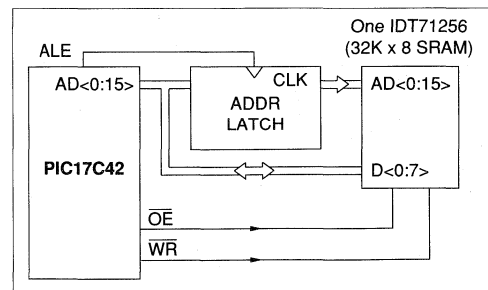


FIGURE 4 - ALTERNATE EXTERNAL MEMORY CONNECTION



Implementing FFT

APPENDIX A: FFT ALGORITHM

MPASM B0.54

PAGE 1

```
*****  
; A Cooley-Tukey Radix-2 DIF FFT  
;  
; Radix-2 implementation  
; Decimation In Frequency  
; Single Butterfly  
; Table Lookup of Twiddle Factors  
; Complex Input & Complex Output  
;  
; All data is assumed to be 16 bits and the intermediate  
; results are stored in 32 bits  
;  
; Length Of FFT must be a Power Of 2  
; Max Length Possible is 2**15  
;  
; The input/output complex data is organized as a single array  
; of real data followed by imaginary data  
; Data is stored in External Memory and is accessed by  
; TABLRD & TABLWT Instructions  
;  
*****  
LIST P=17C42, C=120, T=ON, R=DEC, N=0  
include "17c42.h"  
;  
include "17c42.mac"  
;  
*****  
; RLC16AB  
;  
; DESCRIPTION:  
; 16 bit rotate left A into B  
;  
; ARGUMENTS:  
; 2*a => b  
;  
; TIMING (in cycles):  
; 3  
;  
;  
RLC16AB MACRO a,b  
  
BCF _carry  
RLCF a+B0,W  
MOVWF b+B0  
RLCF a+B1,W  
MOVWF b+B1  
  
ENDM  
;  
*****  
; TBLADDR  
;  
; DESCRIPTION:  
; Load 16 bit table pointer with specified label  
;  
;
```


Implementing FFT

```

; TIMING (in cycles):
;     4
;

TBLADDR MACRO    label

MOVLW    (label) & 0xff
MOVWF    tblptrl
MOVLW    page    (label)
MOVWF    tblptrh

ENDM

;*****
;     ADDLBL
;
; DESCRIPTION:
;     Add A Label (16 bit constant) To A File Register (16 bit)
;
; TIMING (in cycles):
;     4
;

ADDLBL MACRO    label,f

MOVLW    (label) & 0xff
ADDWF    f+B0
MOVLW    page    (label)
ADDWFC   f+B1

ENDM

;*****
;
0100      FftLen .set    256           ; FFT Length
0008      Power .set    8             ; (2**Power = FftLen)
00EF      DigitRevCount .set    239   ; (FftLen-1) - (2**((Power+1)/2))

0001      SCALE_BUTTERFLY .set    TRUE ; intermediate scaling performed

0800      EXT_RAM_START_ADDR .set    0x0800 ; External Memory Data Storage Start
Addr

;*****
;
0000 0004      CBLOCK 0
ENDC          B0,B1,B2,B3           ; RAM offset constants

;
0018 0002      CBLOCK 0x18
001A 0002      AARG,AARG1          ; 16 bit multiplier A
001C 0004      BARG,BARG1          ; 16 bit multiplicand B
001C 0004      DPX,DPX1,DPX2,DPX3  ; 32 bit multiplier result = A*B
ENDC

;
0020 0004      CBLOCK
ENDC          ACC, ACC1, ACC2, ACC3 ; 32 bit accumulator for computa

;
0024 0002      CBLOCK
0026 0002      count1,count11      ; N1
0028 0002      count2,count22      ; N2
0028 0002      QuartLen,QuartLen1  ; FftLen/4
ENDC

;
002A 0002      CBLOCK
002C 0002      TF_Offset,TF_Offset1 ; twiddle factor address computa
002C 0002      TF_Addr,TF_Addr1

```

Implementing FFT

```
002E 0002      Cos,Cos1,
0030 0002      Sin,Sin1
      ENDC

;
      CBLOCK
0032 0002      VarIloop,VarIloop1
0034 0002      VarJloop,VarJloop1
0036 0001      VarKloop
0037 0002      VarL,VarL1
      ENDC

;
      CBLOCK
0039 0002      Xi,Xi1
003B 0002      Yi,Yi1
003D 0002      Xl,Xl1
003F 0002      Yl,Yl1
      ENDC

;
      CBLOCK
0041 0002      Xt,Xt1
0043 0002      Yt,Yt1
      ENDC

      CBLOCK
0045 0004      temp,temp1,temp2,temp3
0049 0002      testCount,testCount1
004B 0002      PulseCount, PulseCount1
      ENDC

;
;
;*****
;          Test Program For FFT Subroutine
;*****
;
      ORG      0x0000
;
include "square.asm"      ; Generate Test Vector Data
;*****
;          Test Routine For FFT
; FFT Of Square Wave Pulse
;*****

0008          PulseWidthFactor      .set      8
;
      testFft
      MOVK16  2*PulseWidthFactor,testCount

0000 B010          MOVLW      (2*PulseWidthFactor) & 0xff
0001 0149          MOVWF      testCount+B0
0002 B000          MOVLW      (2*PulseWidthFactor)/256
0003 014A          MOVWF      testCount+B1

      CLR16      Yi

0004 293B          CLRWF      Yi+B0
0005 293C          CLRWF      Yi+B1

      TBLADDR ExtRamAddr      ; load table pointers with data start addr

0006 B000          MOVLW      (ExtRamAddr) & 0xff
0007 010D          MOVWF      tblptrl
0008 B008          MOVLW      page      (ExtRamAddr)
0009 010E          MOVWF      tblptrh

;
      nextPulse
```

Implementing FFT

```
MOVK    FftLen/PulseWidthFactor,PulseCount

000A B020          MOVLW  FftLen/PulseWidthFactor
000B 014B          MOVWF  PulseCount

MOVK    FftLen/PulseWidthFactor,PulseCount1

000C B020          MOVLW  FftLen/PulseWidthFactor
000D 014C          MOVWF  PulseCount1

;
MOVK16  0x3FFF,Xi

000E B0FF          MOVLW  (0x3FFF) & 0xff
000F 0139          MOVWF  Xi+B0
0010 B03F          MOVLW  (0x3FFF)/256
0011 013A          MOVWF  Xi+B1

LX1
0012 E034          call   write
0013 174B          decfsz PulseCount
0014 C012          goto   LX1
;
CLR16  Xi

0015 2939          CLRFB  Xi+B0
0016 293A          CLRFB  Xi+B1

LX2
0017 E034          call   write
0018 174C          decfsz PulseCount1
0019 C017          goto   LX2
;
DEC16  testCount

001A 2900          CLRFB  WREG
001B 0749          DECF  testCount+B0
001C 034A          SUBWFB testCount+B1

TFSZ16 testCount

001D 6049          MOVFP  testCount+B0,WREG
001E 084A          IORWF  testCount+B1,W
001F 3300          TSTFSZ WREG

0020 C00A          goto   nextPulse

;
0021 E039          call   R2FFT           ; Compute Fourier Transform
0022 E116          call   Unscramble     ; Digit Reverse the scrambled data
;
; Fourier Transform Completed
;
; capture data to PIC-MASTER Trace Buffer
MOVK16  FftLen*2,temp

0023 B000          MOVLW  (FftLen*2) & 0xff
0024 0145          MOVWF  temp+B0
0025 B002          MOVLW  (FftLen*2)/256
0026 0146          MOVWF  temp+B1
```

Implementing FFT

```

TBLADDR ExtRamAddr                ; load table pointers with data start addr

0027 B000          MOVWL  (ExtRamAddr) & 0xff
0028 010D          MOVWF  tblptrl
0029 B008          MOVWL  page      (ExtRamAddr)
002A 010E          MOVWF  tblptrh

capture
002B A930          tblrld  0,1,Sin ; table latch = mem(tblptr)
DEC16  temp

002C 2900          CLRWF  WREG
002D 0745          DECF   temp+B0
002E 0346          SUBWFB temp+B1

TFSZ16  temp

002F 6045          MOVFP  temp+B0,WREG
0030 0846          IORWF  temp+B1,W
0031 3300          TSTFSZ  WREG

0032 C02B          goto   capture
;
0033 C033          self   goto  self
;
write
0034 A439          t1wt   0,Xi
0035 AF3A          tblwt  1,1,Xi+B1      ; auto increment for Imag Data
0036 A43B          t1wt   0,Yi
0037 AF3C          tblwt  1,1,Yi+B1
0038 0002          return
;
;*****
;                               RADIX-2 FFT
;
;       Decimation In Frequency
;
; Input Data should be unscrambled
; Output Data at the end is in scrambled form
; To obtain the unscrambled form, the digit reverse counter
; subroutine, "Unscramble" should be called (see the example)
;
;*****

R2FFT
MOVK16  FftLen,count2                ; count2 = N

0039 B000          MOVWL  (FftLen) & 0xff
003A 0126          MOVWF  count2+B0
003B B001          MOVWL  (FftLen)/256
003C 0127          MOVWF  count2+B1

MOVK16  FftLen/4,QuartLen            ; QuartLen = FftLen/4

003D B040          MOVWL  (FftLen/4) & 0xff
003E 0128          MOVWF  QuartLen+B0
003F B000          MOVWL  (FftLen/4)/256
0040 0129          MOVWF  QuartLen+B1

0041 292B          clrf   TF_Offset+B1
MOVK    1,TF_Offset                ; Init TF_Offset = 1

0042 B001          MOVWL  1
0043 012A          MOVWF  TF_Offset

```

Implementing FFT

```

MOVK    Power,VarKloop                ; Kloop
0044 B008                MOVLW    Power
0045 0136                MOVWF    VarKloop

                Kloop                    ; for K = 1 to Power-1
MOV16   count2,count1                ; count1 = count2

0046 6026                MOVFP   count2+B0,WREG    ; get byte of count2 into w
0047 0124                MOVWF   count1+B0          ; move to count1(B0)
0048 6027                MOVFP   count2+B1,WREG    ; get byte of count2 into w
0049 0125                MOVWF   count1+B1          ; move to count1(B1)

                RRC16   count2                ; count2 = count2/2

004A 1A27                RLCF   count2+B1,W        ; move sign into carry bit
004B 1927                RRCF   count2+B1
004C 1926                RRCF   count2+B0

                CLR16   VarJloop                ; J = 0

004D 2934                CLRF   VarJloop+B0
004E 2935                CLRF   VarJloop+B1

                CLR16   TF_Addr                ; TF_Addr = 0

004F 292C                CLRF   TF_Addr+B0
0050 292D                CLRF   TF_Addr+B1

                Jloop
                ;
                ; Read Twiddle factors from Sine/Cosine Table from Prog Mem
                ;
MOVFP16 TF_Addr,tblptrl                ; load sine table address to table

0051 6D2C                MOVFP   TF_Addr+B0,tblptrl+B0 ; move TF_Addr(B0) to tblptrl(B0)
0052 6E2D                MOVFP   TF_Addr+B1,tblptrl+B1 ; move TF_Addr(B1) to tblptrl(B1)

                ADDLBL  SineTable,tblptrl        ; add table offset

0053 B094                MOVLW   (SineTable) & 0xff
0054 0F0D                ADDWF  tblptrl+B0
0055 B002                MOVLW   page(SineTable)
0056 110E                ADDWFC tblptrl+B1

0057 A830                tblrld  0,0,Sin                ; Read Sine Value from lookup table
0058 A030                tlrld   0,Sin
0059 A231                tlrld   1,Sin+B1
                ADD16   QuartLen,tblptrl

005A 6028                MOVFP   QuartLen+B0,WREG    ; get lowest byte of QuartLen into w
005B 0F0D                ADDWF  tblptrl+B0          ; add lowest byte of tblptrl, save in
005C 6029                MOVFP   QuartLen+B1,WREG    ; get 2nd byte of QuartLen into w
005D 110E                ADDWFC tblptrl+B1          ; add 2nd byte of tblptrl, save in

005E A82E                tblrld  0,0,Cos                ; Read Cosine Value from table
005F A02E                tlrld   0,Cos
0060 A22F                tlrld   1,Cos+B1

```

Implementing FFT

```

;
ADD16 TF_Offset,TF_Addr ; TF_Addr = TF_Addr + TF_Offset

0061 602A MOVFP TF_Offset+B0,WREG ; get lowest byte of TF_Offset into w
0062 0F2C ADDWF TF_Addr+B0 ; add lowest byte of TF_Addr, save in
0063 602B MOVFP TF_Offset+B1,WREG ; get 2nd byte of TF_Offset into w
0064 112D ADDWFC TF_Addr+B1 ; add 2nd byte of TF_Addr, save in

;
RLC16AB VarJloop,VarIloop ; I = J*2 since Real followed by Imag

0065 8804 BCF _carry
0066 1A34 RLCF VarJloop+B0,W
0067 0132 MOVWF VarIloop+B0
0068 1A35 RLCF VarJloop+B1,W
0069 0133 MOVWF VarIloop+B1

;
Iloop

RLC16AB count2,VarL ; VarL = count2*2

006A 8804 BCF _carry
006B 1A26 RLCF count2+B0,W
006C 0137 MOVWF VarL+R0
006D 1A27 RLCF count2+B1,W
006E 0138 MOVWF VarL+B1

ADD16 VarIloop,VarL ; VarL = (I+count2)*2

006F 6032 MOVFP VarIloop+B0,WREG ; get lowest byte of VarIloop into w
0070 0F37 ADDWF VarL+B0 ; add lowest byte of VarL, save in
0071 6033 MOVFP VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
0072 1138 ADDWFC VarL+B1 ; add 2nd byte of VarL, save in VarL(B1)

;
; Get Real & Imag Data from external RAMs (Program Memory)
; load table pointers with data start addr
;
MOVFP16 VarL,tblptr1 ; read data(L)

0073 6D37 MOVFP VarL+B0,tblptr1+B0 ; move VarL(B0) to tblptr1(B0)
0074 6E38 MOVFP VarL+B1,tblptr1+B1 ; move VarL(B1) to tblptr1(B1)

ADDLBL ExtRamAddr,tblptr1 ; add data addr offset

0075 B000 MOVLW (ExtRamAddr) & 0xff
0076 0F0D ADDWF tblptr1+B0
0077 B008 MOVLW page (ExtRamAddr)
0078 110E ADDWFC tblptr1+B1

0079 A93D tablrd 0,1,X1 ; auto increment for Imag Data
007A A03D tlrld 0,X1
007B A23E tlrld 1,X1+B1 ; real data XL
007C A83F tablrd 0,0,Y1
007D A03F tlrld 0,Y1
007E A240 tlrld 1,Y1+B1 ; imag data YL

MOVFP16 VarIloop,tblptr1 ; read data(I)

007F 6D32 MOVFP VarIloop+B0,tblptr1+B0 ; move VarIloop(B0) to tblptr1(B0)
0080 6E33 MOVFP VarIloop+B1,tblptr1+B1 ; move VarIloop(B1) to tblptr1(B1)

```

Implementing FFT

```
ADDLBL ExtRamAddr,tblptrl ; add data addr offset

0081 B000 MOV LW (ExtRamAddr) & 0xff
0082 0F0D ADD WF tblptrl+B0
0083 B008 MOV LW page (ExtRamAddr)
0084 110E ADD WF tblptrl+B1

0085 A939 tablrd 0,1,Xi ; auto increment for Imag Data
0086 A039 tlrdr 0,Xi
0087 A23A tlrdr 1,Xi+B1 ; real data XI
0088 A83B tablrd 0,0,Yi
0089 A03B tlrdr 0,Yi
008A A23C tlrdr 1,Yi+B1 ; imag data YI

;
; Real & Imag Data is fetched
; Compute Butterfly
;
SUB16ACC X1,Xi,Xt ; Xt = Xi - X1

008B 603D MOV FP X1+B0,WREG ; get lowest byte of X1 into w
008C 0439 SUB WF Xi+B0,W ; sub lowest byte of Xi, save in Xi(B0)
008D 0141 MOV WF Xt+B0
008E 603E MOV FP X1+B1,WREG ; get 2nd byte of X1 into w
008F 023A SUB WFB Xi+B1,W ; sub 2nd byte of Xi, save in Xi(B1)
0090 0142 MOV WF Xt+B1

ADD16 X1,Xi ; Xi = Xi + X1

0091 603D MOV FP X1+B0,WREG ; get lowest byte of X1 into w
0092 0F39 ADD WF Xi+B0 ; add lowest byte of Xi, save in Xi(B0)
0093 603E MOV FP X1+B1,WREG ; get 2nd byte of X1 into w
0094 113A ADD WFC Xi+B1 ; add 2nd byte of Xi, save in Xi(B1)

SUB16ACC Y1,Yi,Yt ; Yt = Yi - Y1

0095 603F MOV FP Y1+B0,WREG ; get lowest byte of Y1 into w
0096 043B SUB WF Yi+B0,W ; sub lowest byte of Yi, save in Yi(B0)
0097 0143 MOV WF Yt+B0
0098 6040 MOV FP Y1+B1,WREG ; get 2nd byte of Y1 into w
0099 023C SUB WFB Yi+B1,W ; sub 2nd byte of Yi, save in Yi(B1)
009A 0144 MOV WF Yt+B1

ADD16 Y1,Yi ; Yi = Yi + Y1

009B 603F MOV FP Y1+B0,WREG ; get lowest byte of Y1 into w
009C 0F3B ADD WF Yi+B0 ; add lowest byte of Yi, save in Yi(B0)
009D 6040 MOV FP Y1+B1,WREG ; get 2nd byte of Y1 into w
009E 113C ADD WFC Yi+B1 ; add 2nd byte of Yi, save in Yi(B1)

;
if SCALE_BUTTERFLY
RRC16 X1

009F 1A3A RLCF Xi+B1,W ; move sign into carry bit
00A0 193A RRCF Xi+B1
00A1 1939 RRCF Xi+B0

RRC16 Yi

00A2 1A3C RLCF Yi+B1,W ; move sign into carry bit
```

Implementing FFT

```

00A3 193C          RRCF   Yi+B1
00A4 193B          RRCF   Yi+B0

                RRC16   Xt

00A5 1A42          RLCF   Xt+B1,W      ; move sign into carry bit
00A6 1942          RRCF   Xt+B1
00A7 1941          RRCF   Xt+B0

                RRC16   Yt

00A8 1A44          RLCF   Yt+B1,W      ; move sign into carry bit
00A9 1944          RRCF   Yt+B1
00AA 1943          RRCF   Yt+B0

                endif
;

                MOVFP16 Cos,AARG

00AB 782E          MOVFP   Cos+B0,AARG+B0    ; move Cos(B0) to AARG(B0)
00AC 792F          MOVFP   Cos+B1,AARG+B1    ; move Cos(B1) to AARG(B1)

                MOVFP16 Yt,BARG

00AD 7A43          MOVFP   Yt+B0,BARG+B0    ; move Yt(B0) to BARG(B0)
00AE 7B44          MOVFP   Yt+B1,BARG+B1    ; move Yt(B1) to BARG(B1)

00AF E182          Call    DblMult ; COS*Yt
                MOVFP32 DPX,ACC

00B0 5C20          MOVFP   DPX+B0,ACC+B0    ; move DPX(B0) to ACC(B0)
00B1 5D21          MOVFP   DPX+B1,ACC+B1    ; move DPX(B1) to ACC(B1)
00B2 5E22          MOVFP   DPX+B2,ACC+B2    ; move DPX(B2) to ACC(B2)
00B3 5F23          MOVFP   DPX+B3,ACC+B3    ; move DPX(B3) to ACC(B3)

                MOVFP16 Sin,AARG

00B4 7830          MOVFP   Sin+B0,AARG+B0    ; move Sin(B0) to AARG(B0)
00B5 7931          MOVFP   Sin+B1,AARG+B1    ; move Sin(B1) to AARG(B1)

                MOVFP16 Xt,BARG

00B6 7A41          MOVFP   Xt+B0,BARG+B0    ; move Xt(B0) to BARG(B0)
00B7 7B42          MOVFP   Xt+B1,BARG+B1    ; move Xt(B1) to BARG(B1)

00B8 E182          Call    DblMult ; SIN*Xt, Scale if necessary

                ADD32   ACC,DPX

00B9 6020          MOVFP   ACC+B0,WREG      ; get lowest byte of ACC into w
00BA 0F1C          ADDWF  DPX+B0          ; add lowest byte of DPX, save in DPX(B0)
00BB 6021          MOVFP   ACC+B1,WREG      ; get 2nd byte of ACC into w
00BC 111D          ADDWFC DPX+B1          ; add 2nd byte of DPX, save in DPX(B1)
00BD 6022          MOVFP   ACC+B2,WREG      ; get 3rd byte of ACC into w
00BE 111E          ADDWFC DPX+B2          ; add 3rd byte of DPX, save in DPX(B2)
00BF 6023          MOVFP   ACC+B3,WREG      ; get 4th byte of ACC into w
00C0 111F          ADDWFC DPX+B3          ; add 4th byte of DPX, save in DPX(B3)

```


Implementing FFT

```

MOVFP16 DPX+B2,Y1 ; Y1 = COS*Yt + SIN*Xt, Scale if neces
00C1 5E3F MOVFP DPX+B2+B0,Y1+B0 ; move DPX+B2 (B0) to Y1 (B0)
00C2 5F40 MOVFP DPX+B2+B1,Y1+B1 ; move DPX+B2 (B1) to Y1 (B1)

;
MOVFP16 Yt,BARG ; AARG = SIN, BARG = Yt
00C3 7A43 MOVFP Yt+B0,BARG+B0 ; move Yt (B0) to BARG (B0)
00C4 7B44 MOVFP Yt+B1,BARG+B1 ; move Yt (B1) to BARG (B1)

00C5 E182 Call DblMult ; SIN*Yt
MOVFP32 DPX,ACC

00C6 5C20 MOVFP DPX+B0,ACC+B0 ; move DPX (B0) to ACC (B0)
00C7 5D21 MOVFP DPX+B1,ACC+B1 ; move DPX (B1) to ACC (B1)
00C8 5E22 MOVFP DPX+B2,ACC+B2 ; move DPX (B2) to ACC (B2)
00C9 5F23 MOVFP DPX+B3,ACC+B3 ; move DPX (B3) to ACC (B3)

MOVFP16 Cos,AARG
00CA 782E MOVFP Cos+B0,AARG+B0 ; move Cos (B0) to AARG (B0)
00CB 792F MOVFP Cos+B1,AARG+B1 ; move Cos (B1) to AARG (B1)

MOVFP16 Xt,BARG
00CC 7A41 MOVFP Xt+B0,BARG+B0 ; move Xt (B0) to BARG (B0)
00CD 7B42 MOVFP Xt+B1,BARG+B1 ; move Xt (B1) to BARG (B1)

00CE E182 Call DblMult ; COS*Xt, Scale if necessary
SUB32 ACC,DPX ; DPX = COS*Xt - SIN*Yt

00CF 6020 MOVFP ACC+B0,WREG ; get lowest byte of ACC into w
00D0 051C SUBWF DPX+B0 ; sub lowest byte of DPX, save in DPX (B0)
00D1 6021 MOVFP ACC+B1,WREG ; get 2nd byte of ACC into w
00D2 031D SUBWFB DPX+B1 ; sub 2nd byte of DPX, save in DPX (B1)
00D3 6022 MOVFP ACC+B2,WREG ; get 3rd byte of ACC into w
00D4 031E SUBWFB DPX+B2 ; sub 3rd byte of DPX, save in DPX (B2)
00D5 6023 MOVFP ACC+B3,WREG ; get 4th byte of ACC into w
00D6 031F SUBWFB DPX+B3 ; sub 4th byte of DPX, save in DPX (B3)

MOVFP16 DPX+B2,X1 ; X1 = COS*Xt - SIN*Yt, Scale if neces
00D7 5E3D MOVFP DPX+B2+B0,X1+B0 ; move DPX+B2 (B0) to X1 (B0)
00D8 5F3E MOVFP DPX+B2+B1,X1+B1 ; move DPX+B2 (B1) to X1 (B1)

;
;
; Store results of butterfly
;
DEC16 tblptrl ; table pointer already loaded with I

00D9 2900 CLRF WREG
00DA 070D DECF tblptrl+B0
00DB 030E SUBWFB tblptrl+B1

00DC A439 t1wt 0,Xi
00DD AF3A tab1wt 1,1,Xi+B1 ; auto increment for Imag Data

```

Implementing FFT

```
00DE A43B          tlwt    0,Yi
00DF AE3C          tablwt  1,0,Yi+B1          ; Xi & Yi stored

        MOVFP16 VarL,tblptrl          ; read data(L)

00E0 6D37          MOVFP   VarL+B0,tblptrl+B0 ; move VarL(B0) to tblptrl(B0)
00E1 6E38          MOVFP   VarL+B1,tblptrl+B1 ; move VarL(B1) to tblptrl(B1)

        ADDLBL  ExtRamAddr,tblptrl          ; add data addr offset

00E2 B000          MOVLW   (ExtRamAddr) & 0xff
00E3 0F0D          ADDWF  tblptrl+B0
00E4 B008          MOVLW   page      (ExtRamAddr)
00E5 110E          ADDWFC  tblptrl+B1

00E6 A43D          tlwt    0,X1
00E7 AF3E          tablwt  1,1,X1+B1          ; auto increment for Imag Data
00E8 A43F          tlwt    0,Y1
00E9 AE40          tablwt  1,0,Y1+B1          ; X(L) & Y(L) stored
;
; Increment for next Iloop
;
        RLC16AB count1,temp          ; temp = count1*2

00EA 8804          BCF    _carry
00EB 1A24          RLCF  count1+B0,W
00EC 0145          MOVWF  temp+B0
00ED 1A25          RLCF  count1+B1,W
00EE 0146          MOVWF  temp+B1

        ADD16   temp,VarIloop          ; I = I + temp

00EF 6045          MOVFP  temp+B0,WREG          ; get lowest byte of temp into w
00F0 0F32          ADDWF  VarIloop+B0          ; add lowest byte of VarIloop, save in
VarIloop(B0)

00F1 6046          MOVFP  temp+B1,WREG          ; get 2nd byte of temp into w
00F2 1133          ADDWFC  VarIloop+B1          ; add 2nd byte of VarIloop, save in
VarIloop(B1)
;

        MOVK16  (FftLen*2),temp

00F3 B000          MOVLW  ((FftLen*2)) & 0xff
00F4 0145          MOVWF  temp+B0
00F5 B002          MOVLW  ((FftLen*2))/256
00F6 0146          MOVWF  temp+B1

        SUB16   VarIloop,temp          ; temp = 2*FftLen - I

00F7 6032          MOVFP  VarIloop+B0,WREG          ; get lowest byte of VarIloop into w
00F8 0545          SUBWF  temp+B0          ; sub lowest byte of temp, save in
temp(B0)
00F9 6033          MOVFP  VarIloop+B1,WREG          ; get 2nd byte of VarIloop into w
00FA 0346          SUBWFB  temp+B1          ; sub 2nd byte of temp, save in temp(B1)

        DEC16   temp

00FB 2900          CLRF  WREG
00FC 0745          DECF  temp+B0
00FD 0346          SUBWFB  temp+B1
```

Implementing FFT

```
00FE 9746          btfss  temp+B1,MSB
00FF C06A          goto   Iloop           ; while I < 2*FftLen
;
; I Loop end
;
; increment for next J Loop
;
        INC16  VarJloop           ; J = J + 1

0100 2900          CLRF   WREG
0101 1534          INCF   VarJloop+B0
0102 1135          ADDWFC VarJloop+B1

        MOV16  count2,temp

0103 6026          MOVFP  count2+B0,WREG      ; get byte of count2 into w
0104 0145          MOVWF  temp+B0              ; move to temp(B0)
0105 6027          MOVFP  count2+B1,WREG      ; get byte of count2 into w
0106 0146          MOVWF  temp+B1              ; move to temp(B1)

        SUB16  VarJloop,temp           ; temp = count2 - J

0107 6034          MOVFP  VarJloop+B0,WREG    ; get lowest byte of VarJloop into w
0108 0545          SUBWF  temp+B0              ; sub lowest byte of temp, save in
temp(B0)
0109 6035          MOVFP  VarJloop+B1,WREG    ; get 2nd byte of VarJloop into w
010A 0346          SUBWFB temp+B1              ; sub 2nd byte of temp, save in temp(B1)

        DEC16  temp

010B 2900          CLRF   WREG
010C 0745          DECF   temp+B0
010D 0346          SUBWFB temp+B1

010E 9746          btfss  temp+B1,MSB
010F C051          goto   Jloop           ; while J < count2
;
; J Loop end
;
; increment for next K Loop
;
        RLC16  TF_Offset           ; TF_Offset = 2 * TF_Offset

0110 8804          BCF   _carry
0111 1B2A          RLCF   TF_Offset+B0
0112 1B2B          RLCF   TF_Offset+B1

0113 1736          decfsz VarKloop
0114 C046          goto   Kloop           ; while K < Power
;
0115 0002          return           ; FFT complete
;
; K Loop End
; FFT Computation Over with data scrambled
; Descramble the data using "Unscramble" Routine
;
;*****
; Unscramble Data Order Sequence
; A digit reverse counter
;*****
```

Implementing FFT

```

include "reverse.asm"
;*****
;
;           A digit reverse counter
;
;   Unscramble Data Order Sequence Of Radix-2 FFT
;   Length (must be a power of 2) is limited only by
;   the amount of External RAM available and must be
;   a number less than 2**15
;
;*****

Unscramble

CLR16  VarJloop      ; J = 0

0116 2934          CLRf  VarJloop+B0
0117 2935          CLRf  VarJloop+B1

0118 2933          clrf  VarIloop+B1
MOVk  1,VarIloop   ; I = 1

0119 B001          MOVlW  1
011A 0132          MOVWf  VarIloop

                nextI
MOVk16 FftLen/2,VarKloop

011B B080          MOVlW  (FftLen/2) & 0xff
011C 0136          MOVWf  VarKloop+B0
011D B000          MOVlW  (FftLen/2)/256
011E 0137          MOVWf  VarKloop+B1

011F C127          goto   testK
                KlessJ
SUB16  VarKloop,VarJloop      ; J = J - K

0120 6036          MOVFP  VarKloop+B0,WREG ; get lowest byte of VarKloop into w
0121 0534          SUBWf  VarJloop+B0      ; sub lowest byte of VarJloop, save in
VarJloop(B0)

0122 6037          MOVFP  VarKloop+B1,WREG ; get 2nd byte of VarKloop into w
0123 0335          SUBWFB VarJloop+B1      ; sub 2nd byte of VarJloop, save in
VarJloop(B1)

RRC16  VarKloop      ; K = K/2

0124 1A37          RLCf  VarKloop+B1,W      ; move sign into carry bit
0125 1937          RRCf  VarKloop+B1
0126 1936          RRCf  VarKloop+B0

                testK
MOV16  VarJloop,temp

0127 6034          MOVFP  VarJloop+B0,WREG ; get byte of VarJloop into w
0128 0145          MOVWf  temp+B0          ; move to temp(B0)
0129 6035          MOVFP  VarJloop+B1,WREG ; get byte of VarJloop into w
012A 0146          MOVWf  temp+B1          ; move to temp(B1)

SUB16  VarKloop,temp      ; temp = J - K

012B 6036          MOVFP  VarKloop+B0,WREG ; get lowest byte of VarKloop into w
012C 0545          SUBWf  temp+B0          ; sub lowest byte of temp, save in
temp(B0)

```

Implementing FFT

```
012D 6037      MOVFP  VarKloop+B1,WREG ; get 2nd byte of VarKloop into w
012E 0346      SUBWFB temp+B1           ; sub 2nd byte of temp, save in temp(B1)

012F 9746      btfs  temp+B1,MSB
0130 C120      goto   KlessJ           ; while K < J

                ADD16  VarKloop,VarJloop           ; J = J + K

0131 6036      MOVFP  VarKloop+B0,WREG ; get lowest byte of VarKloop into w
0132 0F34      ADDWF  VarJloop+B0     ; add lowest byte of VarJloop, save in
VarJloop(B0)

0133 6037      MOVFP  VarKloop+B1,WREG ; get 2nd byte of VarKloop into w
0134 1135      ADDWFC VarJloop+B1     ; add 2nd byte of VarJloop, save in
VarJloop(B1)

                ;
                ; if (i < j) then swap data(i) & data(j)
                ;
MOV16  VarJloop,temp

0135 6034      MOVFP  VarJloop+B0,WREG ; get byte of VarJloop into w
0136 0145      MOVWF  temp+B0           ; move to temp(B0)
0137 6035      MOVFP  VarJloop+B1,WREG ; get byte of VarJloop into w
0138 0146      MOVWF  temp+B1           ; move to temp(B1)

                SUB16  VarIloop,temp           ; temp = J - I

0139 6032      MOVFP  VarIloop+B0,WREG ; get lowest byte of VarIloop into w
013A 0545      SUBWF  temp+B0           ; sub lowest byte of temp, save in
temp(B0)
013B 6033      MOVFP  VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
013C 0346      SUBWFB temp+B1           ; sub 2nd byte of temp, save in temp(B1)

                DEC16  temp

013D 2900      CLRF  WREG
013E 0745      DECF  temp+B0
013F 0346      SUBWFB temp+B1

0140 9F46      btfs  temp+B1,MSB
0141 C174      goto   incI

                ;
                ; swap data
                ; read data(i)

                RLC16AB VarIloop,tblptrl           ; add twice the addr, since Real Data

0142 8804      BCF  _carry
0143 1A32      RLCF  VarIloop+B0,W
0144 010D      MOVWF tblptrl+B0
0145 1A33      RLCF  VarIloop+B1,W
0146 010E      MOVWF tblptrl+B1

                ADDLBL ExtRamAddr,tblptrl           ; is followed by Imag Data

0147 B000      MOVLW (ExtRamAddr) & 0xff
0148 0F0D      ADDWF tblptrl+B0
0149 B008      MOVLW page (ExtRamAddr)
014A 110E      ADDWFC tblptrl+B1
```

Implementing FFT

```
014B A939          tblrld 0,1,Xi ; auto increment for Imag Data
014C A039          tlrld  0,Xi
014D A23A          tlrld  1,Xi+B1 ; real data XI
014E A83B          tblrld 0,0,Yi
014F A03B          tlrld  0,Yi
0150 A23C          tlrld  1,Yi+B1 ; imag data YI
;
; read data(j)
;
      RLC16AB VarJloop,tblptrl      ; add twice the addr, since Real Data

0151 8804          BCF      _carry
0152 1A34          RLCF    VarJloop+B0,W
0153 010D          MOVWF   tblptrl+B0
0154 1A35          RLCF    VarJloop+B1,W
0155 010E          MOVWF   tblptrl+B1

      ADDLBL ExtRamAddr,tblptrl      ; is followed by Imag Data

0156 B000          MOVLW   (ExtRamAddr) & 0xff
0157 0F0D          ADDWF   tblptrl+B0
0158 B008          MOVLW   page      (ExtRamAddr)
0159 110E          ADDWF   tblptrl+B1

015A A93D          tblrld 0,1,X1 ; auto increment for Imag Data
015B A03D          tlrld  0,X1
015C A23E          tlrld  1,X1+B1 ; real data XL
015D A83F          tblrld 0,0,Y1
015E A03F          tlrld  0,Y1
015F A240          tlrld  1,Y1+B1 ; imag data YL
;
; Interchange data(I) & data(J)
;
; J addr already loaded into table pointers, bu autoincremented
;
      DEC16  tblptrl

0160 2900          CLRF    WREG
0161 070D          DECF    tblptrl+B0
0162 030E          SUBWF   tblptrl+B1

0163 A439          tlwt   0,Xi
0164 AF3A          tblwt  1,1,Xi+B1      ; auto increment for Imag Data
0165 A43B          tlwt   0,Yi
0166 AE3C          tblwt  1,0,Yi+B1      ; X(I) & Y(I) stored

      RLC16AB VarIloop,tblptrl      ; add twice the addr, since Real Data

0167 8804          BCF      _carry
0168 1A32          RLCF    VarIloop+B0,W
0169 010D          MOVWF   tblptrl+B0
016A 1A33          RLCF    VarIloop+B1,W
016B 010E          MOVWF   tblptrl+B1

      ADDLBL ExtRamAddr,tblptrl      ; is followed by imag Data

016C B000          MOVLW   (ExtRamAddr) & 0xff
016D 0F0D          ADDWF   tblptrl+B0
016E B008          MOVLW   page      (ExtRamAddr)
016F 110E          ADDWF   tblptrl+B1
```

Implementing FFT

```

0170 A43D          tlwt    0,X1
0171 AF3E          tablwt  1,1,X1+B1      ; auto increment for Imag Data
0172 A43F          tlwt    0,Y1
0173 AE40          tablwt  1,0,Y1+B1      ; X(L) & Y(L) stored
                ;
                ; increment I
                ;
                incl
    INC16   VarIloop

0174 2900          CLRF    WREG
0175 1532          INCF    VarIloop+B0
0176 1133          ADDWFC  VarIloop+B1

    MOVK16  DigitRevCount,temp

0177 B0EF          MOVLW   (DigitRevCount) & 0xff
0178 0145          MOVWF   temp+B0
0179 B000          MOVLW   (DigitRevCount)/256
017A 0146          MOVWF   temp+B1

    SUB16   VarIloop,temp          ; temp = DigitRevCount - I

017B 6032          MOVFP   VarIloop+B0,WREG ; get lowest byte of VarIloop into w
017C 0545          SUBWF   temp+B0          ; sub lowest byte of temp, save in
temp(B0)
017D 6033          MOVFP   VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
017E 0346          SUBWFB  temp+B1          ; sub 2nd byte of temp, save in temp(B1)

017F 9746          btfs    temp+B1,MSB
0180 C11B          goto    nextI          ; while i < DigitRevCount
0181 0002          return

                ;
                ; End digit reverse counter
                ;
                ;*****

                ;*****
                ;          Include Double Precision Multiplication Routine
                ;*****
0001          SIGNED  equ    TRUE

    include "17c42mpy.mac"

                ;
                ;*****
                ;          Sine-Cosine Tables
                ;*****
                ;
    include "fft256.tbl"

                ;
                ;          256 Point FFT Sine Table
                ; coefficient table (size of table is 3n/4).
                ;
    SineTable
                ;
0294 0000          data    0
0295 0324          data    804
0296 0648          data    1608
0297 096A          data    2410
0298 0C8C          data    3212
0299 0FAB          data    4011

```



Implementing FFT

029A 12C8	data	4808
029B 15E2	data	5602
029C 18F9	data	6393
029D 1C0B	data	7179
029E 1F1A	data	7962
029F 2223	data	8739
02A0 2528	data	9512
02A1 2826	data	10278
02A2 2B1F	data	11039
02A3 2E11	data	11793
02A4 30FB	data	12539
02A5 33DF	data	13279
02A6 36BA	data	14010
02A7 398C	data	14732
02A8 3C56	data	15446
02A9 3F17	data	16151
02AA 41CE	data	16846
02AB 447A	data	17530
02AC 471C	data	18204
02AD 49B4	data	18868
02AE 4C3F	data	19519
02AF 4EBF	data	20159
02B0 5133	data	20787
02B1 539B	data	21403
02B2 55F5	data	22005
02B3 5842	data	22594
02B4 5A82	data	23170
02B5 5CB3	data	23731
02B6 5ED7	data	24279
02B7 60EB	data	24811
02B8 62F1	data	25329
02B9 64E8	data	25832
02BA 66CF	data	26319
02BB 68A6	data	26790
02BC 6A6D	data	27245
02BD 6C23	data	27683
02BE 6DC9	data	28105
02BF 6F5E	data	28510
02C0 70E2	data	28898
02C1 7254	data	29268
02C2 73B5	data	29621
02C3 7504	data	29956
02C4 7641	data	30273
02C5 776B	data	30571
02C6 7884	data	30852
02C7 7989	data	31113
02C8 7A7C	data	31356
02C9 7B5C	data	31580
02CA 7C29	data	31785
02CB 7CE3	data	31971
02CC 7D89	data	32137
02CD 7E1D	data	32285
02CE 7E9C	data	32412
02CF 7F09	data	32521
02D0 7F61	data	32609
02D1 7FA6	data	32678
02D2 7FD8	data	32728
02D3 7FF5	data	32757
	;	
	CosTable	
	;	
02D4 7FFF	data	32767
02D5 7FF5	data	32757
02D6 7FD8	data	32728
02D7 7FA6	data	32678
02D8 7F61	data	32609
02D9 7F09	data	32521
02DA 7E9C	data	32412
02DB 7E1D	data	32285

Implementing FFT

02DC 7D89	data	32137
02DD 7CE3	data	31971
02DE 7C29	data	31785
02DF 7B5C	data	31580
02E0 7A7C	data	31356
02E1 7989	data	31113
02E2 7884	data	30852
02E3 776B	data	30571
02E4 7641	data	30273
02E5 7504	data	29956
02E6 73B5	data	29621
02E7 7254	data	29268
02E8 70E2	data	28898
02E9 6F5E	data	28510
02EA 6DC9	data	28105
02EB 6C23	data	27683
02EC 6A6D	data	27245
02ED 68A6	data	26790
02EE 66CF	data	26319
02EF 64E8	data	25832
02F0 62F1	data	25329
02F1 60EB	data	24811
02F2 5ED7	data	24279
02F3 5CB3	data	23731
02F4 5A82	data	23170
02F5 5842	data	22594
02F6 55F5	data	22005
02F7 539B	data	21403
02F8 5133	data	20787
02F9 4EBF	data	20159
02FA 4C3F	data	19519
02FB 49B4	data	18868
02FC 471C	data	18204
02FD 447A	data	17530
02FE 41CE	data	16846
02FF 3F17	data	16151
0300 3C56	data	15446
0301 398C	data	14732
0302 36BA	data	14010
0303 33DF	data	13279
0304 30FB	data	12539
0305 2E11	data	11793
0306 2B1F	data	11039
0307 2826	data	10278
0308 2528	data	9512
0309 2223	data	8739
030A 1F1A	data	7962
030B 1C0B	data	7179
030C 18F9	data	6393
030D 15E2	data	5602
030E 12C8	data	4808
030F 0FAB	data	4011
0310 0C8C	data	3212
0311 096A	data	2410
0312 0648	data	1608
0313 0324	data	804
0314 0000	data	0
0315 FCDC	data	-804
0316 F9B8	data	-1608
0317 F696	data	-2410
0318 F374	data	-3212
0319 F055	data	-4011
031A ED38	data	-4808
031B EA1E	data	-5602
031C E707	data	-6393
031D E3F5	data	-7179
031E E0E6	data	-7962
031F DDDD	data	-8739
0320 DAD8	data	-9512

Implementing FFT

```
0321 D7DA      data      -10278
0322 D4E1      data      -11039
0323 D1EF      data      -11793
0324 CF05      data      -12539
0325 CC21      data      -13279
0326 C946      data      -14010
0327 C674      data      -14732
0328 C3AA      data      -15446
0329 C0E9      data      -16151
032A BE32      data      -16846
032B BB86      data      -17530
032C B8E4      data      -18204
032D B64C      data      -18868
032E B3C1      data      -19519
032F B141      data      -20159
0330 AECD      data      -20787
0331 AC65      data      -21403
0332 AA0B      data      -22005
0333 A7BE      data      -22594
0334 A57E      data      -23170
0335 A34D      data      -23731
0336 A129      data      -24279
0337 9F15      data      -24811
0338 9D0F      data      -25329
0339 9B18      data      -25832
033A 9931      data      -26319
033B 975A      data      -26790
033C 9593      data      -27245
033D 93DD      data      -27683
033E 9237      data      -28105
033F 90A2      data      -28510
0340 8F1E      data      -28898
0341 8DAC      data      -29268
0342 8C4B      data      -29621
0343 8AFC      data      -29956
0344 89BF      data      -30273
0345 8895      data      -30571
0346 877C      data      -30852
0347 8677      data      -31113
0348 8584      data      -31356
0349 84A4      data      -31580
034A 83D7      data      -31785
034B 831D      data      -31971
034C 8277      data      -32137
034D 81E3      data      -32285
034E 8164      data      -32412
034F 80F7      data      -32521
0350 809F      data      -32609
0351 805A      data      -32678
0352 8028      data      -32728
0353 800B      data      -32757
```

```
;  
;*****  
  
;  
;*****  
;          FFT Input/Output Data Stored In External RAM  
; Operate Processor In Extended Microcontroller Mode  
; External Data Starts at Address 0x0800, with 2 bytes of  
; Real Data followed by 2 bytes of Imaginary Data.  
;*****  
          ORG          EXT_RAM_START_ADDR  
  
;  
ExtRamAddr  
;  
  
          END
```

```
Errors      :      0  
Warnings    :      0
```

Tone Generation

INTRODUCTION

A general purpose resonator routine is implemented using PIC17C42. This routine is used to generate multiple tones. A tone signal is normally generated using extensive table lookup schemes. When a multiple tone signal is desired, each tone must have its own lookup table, thus requiring a large amount of storage space, especially when various frequencies are to be generated. This application note implements a tone generation using recursive techniques. The algorithm for a resonator is developed and implemented using PIC17C42.

THEORY

Generation of a single tone basically implies generating samples of a sine/cosine wave. The Z-Transform of a sine wave is given as follows :

$$Z\{\sin(wt)\} = \frac{Y(z)}{X(z)} = \frac{z \cdot \sin(wT)}{z^2 - 2z \cdot \cos(wT) + 1}$$

The impulse response of the above transform (i.e. for $X(z) = 1$) will generate a sine wave of frequency w sampled at a rate of $T (= 1/fs)$. Thus the above equation is translated to:

$$Y(z) = \frac{z^{-1} \cdot \sin(wT)}{1 - 2z^{-1} \cdot \cos(wT) + z^{-2}}$$

The above equation can be rewritten in a difference equation form as follows:

$$y(n) - 2y(n-1)\cos(wT) + y(n-2) = x(n-1)\sin(wT)$$

Rearranging the above equation and setting, $x(n)$ as an impulse sequence, the following recursive equations are obtained.

$$y(n) = 2 \cdot K_1 \cdot y(n-1) - y(n-2)$$

$$y(n-2) = y(n-1)$$

$$y(n-1) = y(n)$$

with the following conditions:

$$K_1 = \cos(wT)$$

$$K_2 = \text{initial } y(n-1) = \sin(wT)$$

$$K_3 = \text{initial } y(n-2) = 0$$

IMPLEMENTATION

The above developed algorithm is implemented as a subroutine using PIC17C42. All computations are performed using double precision arithmetic (16/32-bits). The recursive tone generation algorithm is implemented as a subroutine (labelled as "Resonator"). This subroutine generates samples of a single tone. To generate multiple frequencies, simply call this resonator routine for the desired frequencies and sum the outputs. The three tone co-efficients are stored in program memory and are read into the data memory using `TABLRD` instructions.

The fully commented code is listed in Appendix A. The timing and memory requirements are included in the comment sections of the code. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note ANDS00540. This code can be easily modified and used in various applications like DTMF generation, sound generation, etc. The tones generated can easily be output to an on chip PWM channel which in turn can drive a speaker for producing various sounds. If using a PWM channel, it is suggested to set the PWM frequency much higher than the sampling frequency used (in the example code, for 8 KHz sampling frequency, use at least 20 KHz PWM frequency).

As an example, a dual tone is generated and the resulting digital wave form is analyzed. The main program calls the function "Resonator" twice for generating the two desired tones and the two outputs are summed. A sampling frequency of 8 KHz was used to generate a dual tone of 800 Hz and 1.10 KHz. The resulting wave form is shown in Figure 1. The spectrum of the signal shown in Figure 2 shows 2 peaks corresponding to the two desired tones (800 Hz & 1.10 KHz). The assembly code was tested using PICMASTER™ (Microchip's Universal In-Circuit Emulator System).

The generated tones were captured into the PICMASTER's trace buffer and then transferred to Microsoft Excel using Dynamic Data Exchange (DDE). Once the data is in Excel it is analyzed using Excel's FFT utility.



Tone Generation

FIGURE 1 - DUAL TONE WAVE FORM CAPTURED BY PICMASTER

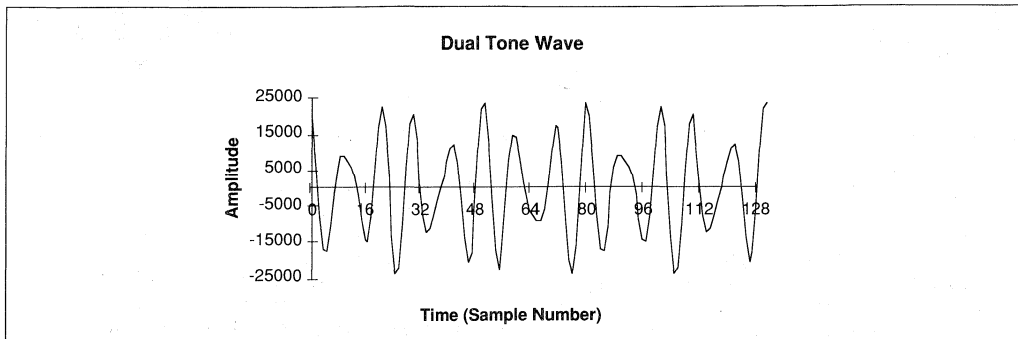
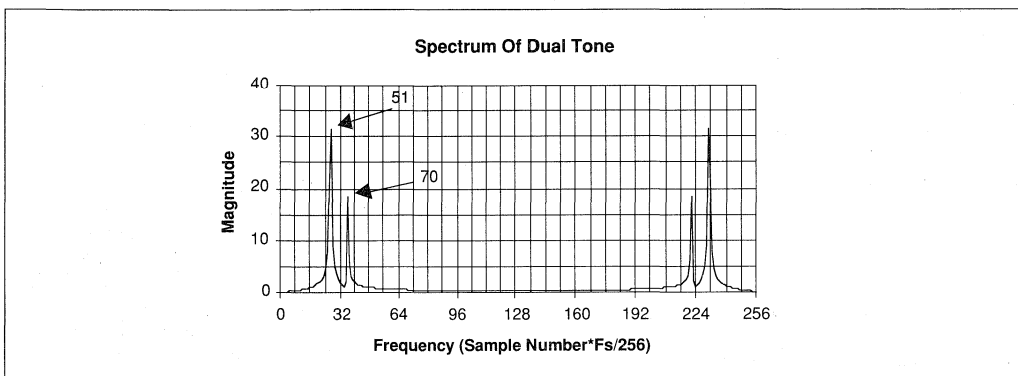


FIGURE 2 - FREQUENCY SPECTRUM OF FIG. 1.1 SHOWING THE DUAL TONE FREQUENCIES



PERFORMANCE

Table 1 below provides the performance of the resonator (labeled in the source code as "Resonator") in terms of both timing and memory requirements. Since a double precision multiplier is used (software implementation), the multiplier timing is not always constant. Therefore the timings are given for the worst case. Note that irrespective of the frequency and the sampling (resolution) of the tone, program memory requirements is only 54 locations which in case of table lookup could be very large.

TABLE 1

Cycles	235 Cycles
Time @ 16 MHz	58.75 μ s
Time @ 25 MHz	47 μ s
Data Memory	9 + 9*(# of tones to be generated)
Program memory	54 locations

APPLICATIONS

Tone generation is required in many application. The code provided in this application note may be used as a general purpose routine to generate desired tones. It can be used in applications involving secure off-site control, where commands/data in the format of tones are transmitted over a telephone line. The tone generation finds applications involving signal modulations as well. The routine can be used to generate audible tones and output to a speaker connected to an I/O Port or a PWM channel.

Author: Amar Palacherla
Logic Products Division

APPENDIX A: TONE.LST

MPASM B0.54

PAGE 1

```
*****
;
;           Dual Tone Generation
;
;   A generic resonator subroutine is implemented to generate
;   tones. Samples Of A Sin/Cos Wave are generated using
;   recursive techniques. Table Lookups are thus avoided
;   This is especially useful in generating multiple tones
;   (e.g. DTMF tone generation, tone signalling, etc), or programmable
;   tone generation which may vary for each application unit.
;
;*****
LIST      P=17C42, C=80, T=ON, R=DEC, N=0

include "17c42.h"

include "17c42.mac"

;
;*****
;           TBLADDR
;
;   DESCRIPTION:
;       Load 16 bit table pointer with specified label
;
;   TIMING (in cycles):
;       4
;
TBLADDR MACRO  label

MOVLW  (label)
MOVWF  tblptrl
MOVLW  page  (label)
MOVWF  tblptrh

ENDM

;
;*****
;           ADDLBL
;
;   DESCRIPTION:
;       Add A Label (16 bit constant) To A File Register (1
;
;   TIMING (in cycles):
;       4
;
ADDLBL MACRO  label,f

MOVLW  (label)
ADDWF  f+B0
MOVLW  page  (label)
ADDWFC f+B1

ENDM
```

Tone Generation

```

;*****
;
;          CBLOCK 0
0000 0004      B0,B1,B2,B3      ; RAM offset constants
          ENDC
;
          CBLOCK 0x18
0018 0002      AARG,AARG1      ; 16 bit multiplier A
001A 0002      BARG,BARG1      ; 16 bit multiplicand B
001C 0004      DPX,DPX1,DPX2,DPX3 ; 32 bit multiplier result = A*B
          ENDC
;
          CBLOCK
          ; Generic Resonator
0020 0002      K1,K11          ; K1
0022 0003      SinCos_2C,SinCos_2,SinCos1_2 ; y(n-2), Init Value = K3
0025 0002      SinCos_1,SinCos1_1 ; y(n-1), Init Value = K2
0027 0002      SinCos,SinCos1  ; y(n)
0029 0000
          ENDC
;
          CBLOCK
0029 0002      tone1K1,tone1K11 ; Tone 1 variables
002B 0003      tone1_2C,tone1_2,tone1_1_2
002E 0002      tone1_1,tone1_1_1
0030 0002      tone1,tone1
0032 0000
0032 0002      tone2K1,tone2K11 ; Tone 2 variables
0034 0003      tone2_2C,tone2_2,tone2_1_2
0037 0002      tone2_1,tone2_1_1
0039 0002      tone2,tone2_1
003B 0000
003B 0002      dualTone,dualTone1 ; tone1+tone2
          ENDC
;*****
;
;          #define ADD_OFFSET FALSE
005A
;
;          #define coeff_addr 0x0030 ; prog mem addr of
005B
;
;          #define DummyAddr 0x0A00
005C
;*****
;
;          Test Routine For Sin Wave Generation
;
;          (a) Call Initialization Subroutine to read desired s
;
;          freq from prog mem to data mem.
;
;          (b) Call the resonator subroutine to generate sampl
;
;          sine wave.
;
;          (c) Perform a Dummy Table Write of The Sine Wave Da
;
;          PIC-MASTER emulator can capture the data in
;
;*****
```

Tone Generation

```

0000 C040          ORG    0x0000
                  goto   start

;*****

;Resonator Co-efficients For Desired Tones

          ORG    coeff_addr

          ; Tone 1 Resonator Constants
          ; Sample Rate = fs = 8 khz, Tone Freq = f = 0.800 Khz

0030 678E          DATA  26510 ; K1 = COS( $\omega T$ ) = COS(360*f/)
0031 259E          DATA  9630  ; K2 = SIN( $\omega T$ ) = SIN(360*f/)
0032 0000          DATA  0      ; K3 is init value of y(n-2

          ; Tone 2 Resonator Constants
          ; Sample Rate = fs = 8 khz, Tone Freq = f = 1.10 Khz

0033 5321          DATA  21281 ; K1 = COS( $\omega T$ ) = COS(360*f/)
0034 1855          DATA  6229  ; K2 = SIN( $\omega T$ ) = SIN(360*f/)
0035 0000          DATA  0      ; K3 is init value of y(n-2

;*****

          ORG    0x0040
start
0040 E057          call   Coeff_Read

; load table pointers with a dummy addr

          MOVK16 DummyAddr,tblptr1

0041 B000          MOVLW  (0x0A00) & 0xff
0042 010D          MOVWF  tblptr1+B0
0043 B00A          MOVLW  ((0x0A00) >> 8)
0044 010E          MOVWF  tblptr1+B1

          NextSample
0045 0000          nop
0046 A43B          tlwt   0,dualTone
          capture
0047 AE3C          tablwt 1,0,dualTone+B1 ; for PIC-MASTER tr
0048 0000          nop
0049 B029          movlw  tone1K1 ; load indirect add
004A 0101          movwf  fsr0 ; for Tone 1
004B E06C          call   Resonator ; Compute next samp

004C B032          movlw  tone2K1 ; load indirect add
004D 0101          movwf  fsr0 ; for Tone 2
004E E06C          call   Resonator ; compute next samp

; Compute Tone1 + Tone2 for dual tone

          ADD16ACC tone1,tone2,dualTone

```

Tone Generation

```
                                ; dualTone = tone1

004F 6030          movfp  tone1+B0,wreg
0050 0E39          addwf  tone2+B0,w
0051 013B          movwf  dualTone+B0
0052 6031          movfp  tone1+B1,wreg
0053 103A          addwfc tone2+B1,w
0054 013C          movwf  dualTone+B1

0055 C045          goto   NextSample

0056 C056          self   goto   self

;*****
; Initialization routine :
; Read Tone 1 & Tone 2 Resonator Frequencies from Program M
; Data Memory
;
; Program Memory :      3 + 8*(#of tones to be gene
; Timing         :      4 + 11*(#of tones to be gen
;*****

Coeff_Read

    TBLADDR coeff_addr

0057 B030          MOVLW  (0x0030)
0058 010D          MOVWF  tblptrl
0059 B000          MOVLW  page   (0x0030)
005A 010E          MOVWF  tblptrh

005B A929          tablrd  0,1,tone1K1
005C A029          tlrld  0,tone1K1
005D AB2A          tablrd  1,1,tone1K1+B1 ; read K1
005E A02E          tlrld  0,tone1_1
005F AB2F          tablrd  1,1,tone1_1+B1 ; read K2
0060 A02C          tlrld  0,tone1_2
0061 A22D          tlrld  1,tone1_2+B1   ; read K3
0062 292B          clrf   tone1_2C

0063 A932          tablrd  0,1,tone2K1
0064 A032          tlrld  0,tone2K1
0065 AB33          tablrd  1,1,tone2K1+B1 ; read K1
0066 A037          tlrld  0,tone2_1
0067 AB38          tablrd  1,1,tone2_1+B1 ; read K2
0068 A035          tlrld  0,tone2_2
0069 A236          tlrld  1,tone2_2+B1   ; read K3
006A 2934          clrf   tone2_2C

006B 0002          return

;*****
; Resonator Subroutine
;
; Before calling this routine, load the indirect register,
; with the starting RAM address of the desired Tone Variabl
; ( eg. For Tone1 Generation, load FSR0 with "Tone1K1" addr
```


Tone Generation

```
;
;
;
; Timing (worst case) :
;      20 + 36 + (worst case multiplier time)
;      = 56 + 179 = 235 cycles
;      = 58.75 uS @ 16Mhz
;      = 47.00 uS @ 20Mhz
;      = 37.60 uS @ 25 Mhz
;
; Memory Requirements :
;      Program Memory : 54 locations
;      Data Memory    : 9 + 9*(#of tones to be ge
;
;*****
;
Resonator
;
; Transfer tone variables to resonator's variables using i
; This is necessary for making the "Resonator" a generic
; subroutine, so that the same code can be called for vario
; tones.
; Indirect addressing mode can be used throught the code i
; subroutine, but is less efficient.
;
006C 8D04          bcf     _fs1
006D 8404          bsf     _fs0          ; auto increment FSR0
006E 4020          movpf  indf0,K1+B0
006F 4021          movpf  indf0,K1+B1
0070 4022          movpf  indf0,SinCos_2C
0071 4023          movpf  indf0,SinCos_2+B0
0072 4024          movpf  indf0,SinCos_2+B1
0073 4025          movpf  indf0,SinCos_1+B0
0074 4026          movpf  indf0,SinCos_1+B1
0075 4027          movpf  indf0,SinCos+B0
0076 4028          movpf  indf0,SinCos+B1
;
; Compute 2*K1*y(n-1)
;
MOVFP16 K1,BARG

0077 7A20          MOVFP  K1+B0,BARG+B0      ; move K1(B0) t
0078 7B21          MOVFP  K1+B1,BARG+B1      ; move K1(B1) t

MOVFP16 SinCos_1,AARG

0079 7825          MOVFP  SinCos_1+B0,AARG+B0; move Si
007A 7926          MOVFP  SinCos_1+B1,AARG+B1; move Si

007B E0A2          call   DblMult
RLC32 DPX          ; DPX = 2*K1*y(n-1)

007C 8804          BCF   _carry
007D 1B1C          RLCF  DPX+B0
```

Tone Generation

```

007E 1B1D          RLCF   DPX+B1
007F 1B1E          RLCF   DPX+B2
0080 1B1F          RLCF   DPX+B3

;
; subtract y(n-2)*(2**15)
;
0081 2922          RRC24   SinCos_2C      clrf   SinCos_2C

0082 1A24          RLCF   SinCos_2C+B2,W      ; move sign

0083 1924          RRCF   SinCos_2C+B2
0084 1923          RRCF   SinCos_2C+B1
0085 1922          RRCF   SinCos_2C+B0

SUB24   SinCos_2C,DPX1          ; DPX = 2*K1*y(n-1) - y(n-2)

0086 6022          MOVFP  SinCos_2C+B0,wreg      ; get lowest
0087 051D          SUBWF  DPX1+B0          ; sub lowest byt
0088 6023          MOVFP  SinCos_2C+B1,wreg      ; get 2nd b
0089 031E          SUBWFB DPX1+B1          ; sub 2nd byte o
008A 6024          MOVFP  SinCos_2C+B2,wreg      ; get 3rd b
008B 031F          SUBWFB DPX1+B2          ; sub 3rd byte o

RLC24   DPX1          ; adjust decimal point

008C 8804          BCF   _carry
008D 1B1D          RLCF   DPX1+B0
008E 1B1E          RLCF   DPX1+B1
008F 1B1F          RLCF   DPX1+B2

;
; update past samples with newly computed values
;
MOVFP16 DPX2,SinCos          ; y(n) = 2*K1*y(n-1) - y(n-2)

0090 5E27          MOVFP  DPX2+B0,SinCos+B0      ; move DPX2
0091 5F28          MOVFP  DPX2+B1,SinCos+B1      ; move DPX2

MOV16   SinCos_1,SinCos_2          ; y(n-2) = y(n-1)

0092 6025          MOVFP  SinCos_1+B0,wreg      ; get byte o
0093 0123          MOVWF  SinCos_2+B0          ; move to Si
0094 6026          MOVFP  SinCos_1+B1,wreg      ; get byte o
0095 0124          MOVWF  SinCos_2+B1          ; move to Si

```

Tone Generation

```
MOVPF16 DPX2,SinCos_1      ; y(n-1) = y(n)
0096 5E25                  MOVPF  DPX2+B0,SinCos_1+B0; move DP
0097 5F26                  MOVPF  DPX2+B1,SinCos_1+B1; move DP

;
; Generation Of The Next Sample Of The Resonator (sine wave
; The 16 bit result is stored in location "SinCos" (low Byte
; "SinCos+1" (High Byte)
;
; write back all the computed values to respective tone var
;
0098 0701                  decf   fsr0
0099 8D04                  bcf   _fs1
009A 8C04                  bcf   _fs0
009B 6028                  movfp SinCos+B1,indf0
009C 6027                  movfp SinCos+B0,indf0
009D 6026                  movfp SinCos_1+B1,indf0
009E 6025                  movfp SinCos_1+B0,indf0
009F 6024                  movfp SinCos_2C+B2,indf0
00A0 6023                  movfp SinCos_2C+B1,indf0
;
00A1 0002                  return
;
;*****
;      Include Double Precision Multiplication Routine
;*****

0001      SIGNED equ      TRUE

include "17c42mpy.mac"
;      NOLIST
;*****
;
;      Double Precision Multiplier For PIC17C42
;
;      Dmult
;
;      DESCRIPTION:
;
;      Multiplication : AARG (16 bits) * BARG (16 bits) -> DPX
;
;      (a) Load the 1st operand in locations AARG+B0 & AARG
;
;      (b) Load the 2nd operand in locations BARG+B0 & BARG
;
;      (c) CALL Dmult
;      (d) The 32 bit result is in locations ( DPX+B0,DPX+B
;
;      In the signed case, a savings of 9 clks can be real
;
;      BARG as the positive factor in the product when pos
;
;
;      TIMING (worst case):
;      unsigned:          17
```

Tone Generation

```

;
;           signed:           BARG+ 17
;
;           BARG- 17
;
;   NOTE :   Define SIGNED/UNSIGNED To 1/0 before including
;
;           this file in your program
;
;*****
;
;           Multiplication Macro
;*****
;
; TIMING:   unsigned:   11+7*10+8*11   = 169 clks
; (worst case) signed:   11+7*10+7*11+5 = 163 clks
;
MULTMAC  MACRO

variable i

i = 0

if  SIGNED

while i < 15

else

while i < 16

endif

if i < 8           ; test low byte

btfsc  BARG+B0,i

else           ; test high byte

btfsc  BARG+B1,i-8

fi

goto  add#v(i)

if i < 8

rlcf  DPX+B3,W           ; rotate sign into carry bit
rrcf  DPX+B3           ; for i < 8, no meaningful bits
rrcf  DPX+B2           ; are in DPX+B0
rrcf  DPX+B1

else

rlcf  DPX+B3,W           ; rotate sign into carry bit
rrcf  DPX+B3
rrcf  DPX+B2
rrcf  DPX+B1
rrcf  DPX+B0

fi

i = i+1
endw

clr  DPX+B0           ; if we get here, BARG = 0
return

```

Tone Generation

```
        add0
movfp  AARG+B0,WREG
addwf  DPX+B2          ;add lsb
movfp  AARG+B1,WREG
addwfc DPX+B3          ;add msb
rlcf   AARG+B1,W      ; rotate sign into carry bit
rrcf   DPX+B3          ; for i < 8, no meaningful bits
rrcf   DPX+B2          ; are in DPX+B0
rrcf   DPX+B1

i = 1

if  SIGNED

    while i < 15

else

    while i < 16

endif

if i < 8

    btfss  BARG+B0,i    ;test low byte

else

    btfss  BARG+B1,i-8  ; test high byte

fi

goto  noadd#v(i)
add#v(i)
movfp  AARG+B0,WREG
addwf  DPX+B2          ;add lsb
movfp  AARG+B1,WREG
addwfc DPX+B3          ;add msb

noadd#v(i)
if i < 8

    rlcf   AARG+B1,W      ; rotate sign into carry bit
    rrcf   DPX+B3          ; for i < 8, no meaningful bits
    rrcf   DPX+B2          ; are in DPX+B0
    rrcf   DPX+B1

else

    rlcf   AARG+B1,W      ; rotate sign into carry bit
    rrcf   DPX+B3
    rrcf   DPX+B2
    rrcf   DPX+B1
    rrcf   DPX+B0

fi

i = i+1
endw

if  SIGNED

    rlcf   AARG+B1,W      ; since BARG is always made posit
    rrcf   DPX+B3          ; the last bit is known to be zer

    rrcf   DPX+B2
    rrcf   DPX+B1
    rrcf   DPX+B0
```

Tone Generation

```
endif

ENDM

;           Double Precision Multiply ( 16x16 -> 32 )
;           ( AARG*BARG -> : 32 bit output in DPX
;
; DblMult
; if SIGNED

00A2 971B           btfss  BARG+B1,MSB           ; test sign of BARG
00A3 C0AE           goto   argsok               ; if positive, ok
      NEG16  AARG+B0           ; if negative, then negate

00A4 1318           COMF   AARG+B0+B0
00A5 1319           COMF   AARG+B0+B1
00A6 2900           CLRF   wreg
00A7 1518           INCF   AARG+B0+B0
00A8 1119           ADDWFC AARG+B0+B1

      NEG16  BARG+B0           ; AARG and BARG

00A9 131A           COMF   BARG+B0+B0
00AA 131B           COMF   BARG+B0+B1
00AB 2900           CLRF   wreg
00AC 151A           INCF   BARG+B0+B0
00AD 111B           ADDWFC BARG+B0+B1

endif

      argsok
CLR16  DPX+B2           ; clear initial partial pr

00AE 291E           CLRF   DPX+B2+B0
00AF 291F           CLRF   DPX+B2+B1

MULTMAC           ; use macro for multiplica

0000           variable i
0000           i = 0
           if SIGNED
           while i < 15
else
           while i < 16

endif

if i < 8           ; test low byte
      btfsc  BARG+B0,i
```

Tone Generation

```

        else
            ; test high byte
        btfsc    BARG+B1,i-8
    fi
        goto    add#v(i)
    if i < 8
        rlcfc   DPX+B3,W      ; rotate sign into carry bit
        rrcfc   DPX+B3      ; for i < 8, no meaningful bits

        rrcfc   DPX+B2      ; are in DPX+B0
        rrcfc   DPX+B1

    else
        rlcfc   DPX+B3,W      ; rotate sign into carry bit
        rrcfc   DPX+B3
        rrcfc   DPX+B2
        rrcfc   DPX+B1
        rrcfc   DPX+B0

    fi

        i = i+1
    endw

        if i < 8
            ; test low byte
00B0 981A        btfsc    BARG+B0,i
                else
                    ; test high byte

                btfsc    BARG+B1,i-8

        fi

00B1 C113        goto    add0

        if i < 8
00B2 1A1F        rlcfc   DPX+B3,W      ; rotate sign into
00B3 191F        rrcfc   DPX+B3      ; for i < 8, no mea
00B4 191E        rrcfc   DPX+B2      ; are in DPX+B0
00B5 191D        rrcfc   DPX+B1

        else

            rlcfc   DPX+B3,W      ; rotate sign into carry bit

            rrcfc   DPX+B3

            rrcfc   DPX+B2

            rrcfc   DPX+B1

            rrcfc   DPX+B0
```

Tone Generation

```
fi

0001          i = i+1

          if i < 8          ; test low byte
00B6 991A      btfsc      BARG+B0,i
          else          ; test high byte

          btfsc      BARG+B1,i-8

fi

00B7 C11D      goto      add1

          if i < 8
00B8 1A1F      rlcfc      DPX+B3,W      ; rotate sign into
00B9 191F      rrcfc      DPX+B3      ; for i < 8, no mea
00BA 191E      rrcfc      DPX+B2      ; are in DPX+B0
00BB 191D      rrcfc      DPX+B1

          else

          rlcfc      DPX+B3,W      ; rotate sign into carry bit
          rrcfc      DPX+B3
          rrcfc      DPX+B2
          rrcfc      DPX+B1
          rrcfc      DPX+B0

fi

0002          i = i+1

          if i < 8          ; test low byte
00BC 9A1A      btfsc      BARG+B0,i
          else          ; test high byte

          btfsc      BARG+B1,i-8

fi
```


Tone Generation

```
00BD C127          goto    add2
                   if i < 8
00BE 1A1F          rlcfc   DPX+B3,W ; rotate sign into
00BF 191F          rrcfc   DPX+B3 ; for i < 8, no mea
00C0 191E          rrcfc   DPX+B2 ; are in DPX+B0
00C1 191D          rrcfc   DPX+B1
                   else
                   rlcfc   DPX+B3,W ; rotate sign into carry bit
                   rrcfc   DPX+B3
                   rrcfc   DPX+B2
                   rrcfc   DPX+B1
                   rrcfc   DPX+B0
                   fi
0003              i = i+1
                   if i < 8 ; test low byte
00C2 9B1A          btfsc   BARG+B0,i
                   else ; test high byte
                   btfsc   BARG+B1,i-8
                   fi
00C3 C131          goto    add3
                   if i < 8
00C4 1A1F          rlcfc   DPX+B3,W ; rotate sign into
00C5 191F          rrcfc   DPX+B3 ; for i < 8, no mea
00C6 191E          rrcfc   DPX+B2 ; are in DPX+B0
00C7 191D          rrcfc   DPX+B1
                   else
                   rlcfc   DPX+B3,W ; rotate sign into carry bit
                   rrcfc   DPX+B3
                   rrcfc   DPX+B2
                   rrcfc   DPX+B1
```

Tone Generation

```

                                rrcf    DPX+B0

                                fi

0004                                i = i+1

                                if i < 8          ; test low byte
0008 9C1A                            btfscc BARG+B0,i
                                else                ; test high byte

                                btfscc BARG+B1,i-8

                                fi

0009 C13B                            goto    add4

                                if i < 8
000CA 1A1F                            rlcfc  DPX+B3,W    ; rotate sign into
000CB 191F                            rrcfc  DPX+B3      ; for i < 8, no mea
000CC 191E                            rrcfc  DPX+B2      ; are in DPX+B0
000CD 191D                            rrcfc  DPX+B1

                                else

                                rlcfc  DPX+B3,W    ; rotate sign into carry bit
                                rrcfc  DPX+B3
                                rrcfc  DPX+B2
                                rrcfc  DPX+B1
                                rrcfc  DPX+B0

                                fi

0005                                i = i+1

                                if i < 8          ; test low byte
000CE 9D1A                            btfscc BARG+B0,i
                                else                ; test high byte

                                btfscc BARG+B1,i-8
```

```
fi
00CF C145      goto    add5
               if i < 8
00D0 1A1F      rlcfc   DPX+B3,W    ; rotate sign into
00D1 191F      rrcfc   DPX+B3      ; for i < 8, no mea
00D2 191E      rrcfc   DPX+B2      ; are in DPX+B0
00D3 191D      rrcfc   DPX+B1
               else
               rlcfc   DPX+B3,W    ; rotate sign into carry bit
               rrcfc   DPX+B3
               rrcfc   DPX+B2
               rrcfc   DPX+B1
               rrcfc   DPX+B0
               fi
0006           i = i+1
               if i < 8           ; test low byte
00D4 9E1A      btfscc  BARG+B0,i
               else               ; test high byte
               btfscc  BARG+B1,i-8
               fi
00D5 C14F      goto    add6
               if i < 8
00D6 1A1F      rlcfc   DPX+B3,W    ; rotate sign into
00D7 191F      rrcfc   DPX+B3      ; for i < 8, no mea
00D8 191E      rrcfc   DPX+B2      ; are in DPX+B0
00D9 191D      rrcfc   DPX+B1
               else
               rlcfc   DPX+B3,W    ; rotate sign into carry bit
               rrcfc   DPX+B3
```

Tone Generation

```

                                rrcf    DPX+B2
                                rrcf    DPX+B1
                                rrcf    DPX+B0

                                fi

0007                                i = i+1

                                if i < 8          ; test low byte
00DA 9F1A                            btfsc    BARG+B0,i
                                else                ; test high byte

                                btfsc    BARG+B1,i-8

                                fi

00DB C159                            goto    add7

                                if i < 8
00DC 1A1F                            rlcfc   DPX+B3,W    ; rotate sign into
00DD 191F                            rrcfc   DPX+B3     ; for i < 8, no mea
00DE 191E                            rrcfc   DPX+B2     ; are in DPX+B0
00DF 191D                            rrcfc   DPX+B1

                                else

                                rlcfc   DPX+B3,W    ; rotate sign into carry bit
                                rrcfc   DPX+B3
                                rrcfc   DPX+B2
                                rrcfc   DPX+B1
                                rrcfc   DPX+B0

                                fi

0008                                i = i+1

                                if i < 8          ; test low byte

                                btfsc    BARG+B0,i
```

Tone Generation

```

                                else                ; test high byte

00E0 981B                        btfsc      BARG+B1,i-8
                                fi
00E1 C163                        goto     add8
                                if i < 8

                                rlcfc      DPX+B3,W ; rotate sign into carry bit
                                rrcfc      DPX+B3 ; for i < 8, no meaningful bits
                                rrcfc      DPX+B2 ; are in DPX+B0
                                rrcfc      DPX+B1

                                else

00E2 1A1F                        rlcfc      DPX+B3,W ; rotate sign into
00E3 191F                        rrcfc      DPX+B3
00E4 191E                        rrcfc      DPX+B2
00E5 191D                        rrcfc      DPX+B1
00E6 191C                        rrcfc      DPX+B0

                                fi
0009                              i = i+1

                                if i < 8                ; test low byte

                                btfsc      BARG+B0,i

                                else                ; test high byte

00E7 991B                        btfsc      BARG+B1,i-8
                                fi
00E8 C16E                        goto     add9
                                if i < 8

                                rlcfc      DPX+B3,W ; rotate sign into carry bit
                                rrcfc      DPX+B3 ; for i < 8, no meaningful bits
                                rrcfc      DPX+B2 ; are in DPX+B0
                                rrcfc      DPX+B1

                                else
```

Tone Generation

```
00E9 1A1F          rlcf   DPX+B3,W   ; rotate sign into
00EA 191F          rrcf   DPX+B3
00EB 191E          rrcf   DPX+B2
00EC 191D          rrcf   DPX+B1
00ED 191C          rrcf   DPX+B0

                    fi

000A              i = i+1

                    if i < 8          ; test low byte

                    btfsc   BARG+B0,i

                    else              ; test high byte

00EE 9A1B          btfsc   BARG+B1,i-8

                    fi

00EF C179          goto   add10

                    if i < 8

                    rlcf   DPX+B3,W   ; rotate sign into carry bit
                    rrcf   DPX+B3     ; for i < 8, no meaningful bits
                    rrcf   DPX+B2     ; are in DPX+B0
                    rrcf   DPX+B1

                    else

00F0 1A1F          rlcf   DPX+B3,W   ; rotate sign into
00F1 191F          rrcf   DPX+B3
00F2 191E          rrcf   DPX+B2
00F3 191D          rrcf   DPX+B1
00F4 191C          rrcf   DPX+B0

                    fi

000B              .i = i+1

                    if i < 8          ; test low byte

                    btfsc   BARG+B0,i

                    else              ; test high byte
```

Tone Generation

```
00F5 9B1B          btfsc    BARG+B1,i-8
                   fi
00F6 C184          goto     add11
                   if i < 8
                   rlcfc   DPX+B3,W    ; rotate sign into carry bit
                   rrcfc   DPX+B3      ; for i < 8, no meaningful bits
                   rrcfc   DPX+B2      ; are in DPX+B0
                   rrcfc   DPX+B1
                   else
00F7 1A1F          rlcfc   DPX+B3,W    ; rotate sign into
00F8 191F          rrcfc   DPX+B3
00F9 191E          rrcfc   DPX+B2
00FA 191D          rrcfc   DPX+B1
00FB 191C          rrcfc   DPX+B0
                   fi
000C              i = i+1
                   if i < 8          ; test low byte
                   btfsc   BARG+B0,i
                   else          ; test high byte
00FC 9C1B          btfsc   BARG+B1,i-8
                   fi
00FD C18F          goto     add12
                   if i < 8
                   rlcfc   DPX+B3,W    ; rotate sign into carry bit
                   rrcfc   DPX+B3      ; for i < 8, no meaningful bits
                   rrcfc   DPX+B2      ; are in DPX+B0
                   rrcfc   DPX+B1
                   else
00FE 1A1F          rlcfc   DPX+B3,W    ; rotate sign into
```

Tone Generation

```
00FF 191F          rrcf    DPX+B3
0100 191E          rrcf    DPX+B2
0101 191D          rrcf    DPX+B1
0102 191C          rrcf    DPX+B0

                    fi

000D              i = i+1

                    if i < 8          ; test low byte

                    btfsc    BARG+B0,i

                    else              ; test high byte

0103 9D1B          btfsc    BARG+B1,i-8

                    fi

0104 C19A          goto    addl3

                    if i < 8

                    rlcfc    DPX+B3,W  ; rotate sign into carry bit

                    rrcfc    DPX+B3    ; for i < 8, no meaningful bits

                    rrcfc    DPX+B2    ; are in DPX+B0

                    rrcfc    DPX+B1

                    else

0105 1A1F          rlcfc    DPX+B3,W  ; rotate sign into

0106 191F          rrcfc    DPX+B3
0107 191E          rrcfc    DPX+B2
0108 191D          rrcfc    DPX+B1
0109 191C          rrcfc    DPX+B0

                    fi

000E              i = i+1

                    if i < 8          ; test low byte

                    btfsc    BARG+B0,i

                    else              ; test high byte

010A 9E1B          btfsc    BARG+B1,i-8

                    fi
```


Tone Generation

```
010B C1A5          goto    add14
                   if i < 8
                   rlcfc   DPX+B3,W    ; rotate sign into carry bit
                   rrcfc   DPX+B3      ; for i < 8, no meaningful bits
                   rrcfc   DPX+B2      ; are in DPX+B0
                   rrcfc   DPX+B1

                   else

010C 1A1F          rlcfc   DPX+B3,W    ; rotate sign into
010D 191F          rrcfc   DPX+B3
010E 191E          rrcfc   DPX+B2
010F 191D          rrcfc   DPX+B1
0110 191C          rrcfc   DPX+B0

                   fi

000F              i = i+1

0111 291C          clrfc   DPX+B0      ; if we get here, B
0112 0002          return

                   add0
0113 6018          movfpc  AARG+B0,WREG
0114 0F1E          addwfc  DPX+B2      ;add lsb
0115 6019          movfpc  AARG+B1,WREG
0116 111F          addwfc  DPX+B3      ;add msb
0117 1A19          rlcfc   AARG+B1,W    ; rotate sign into
0118 191F          rrcfc   DPX+B3      ; for i < 8, no mea
0119 191E          rrcfc   DPX+B2      ; are in DPX+B0
011A 191D          rrcfc   DPX+B1

0001              i = 1
                   if    SIGNED
                   while i < 15
                   else
                   while i < 16
                   endif

                   if i < 8
                   btfss  BARG+B0,i    ;test low byte
                   else
```

Tone Generation

```
        btfss      BARG+B1,i-8 ; test high byte

fi

        goto      noadd#v(i)
        add#v(i)
        movfp     AARG+B0,WREG
        addwf     DPX+B2      ;add lsb
        movfp     AARG+B1,WREG
        addwfc    DPX+B3      ;add msb

        noadd#v(i)
if i < 8

        rlcfc    AARG+B1,W      ; rotate sign into carry bit
        rrcfc    DPX+B3        ; for i < 8, no meaningful bits

        rrcfc    DPX+B2        ; are in DPX+B0
        rrcfc    DPX+B1

else

        rlcfc    AARG+B1,W      ; rotate sign into carry bit
        rrcfc    DPX+B3
        rrcfc    DPX+B2
        rrcfc    DPX+B1
        rrcfc    DPX+B0

        fi

        i = i+1
        endw

        if i < 8      ;test low byte

011B 911A          btfss      BARG+B0,i

        else
        ; test high byte
        btfss      BARG+B1,i-8

        fi

011C C121          goto      noadd1
        addl
011D 6018          movfp     AARG+B0,WREG
011E 0F1E          addwf     DPX+B2      ;add lsb
011F 6019          movfp     AARG+B1,WREG
0120 111F          addwfc    DPX+B3      ;add msb

        noadd1

        if i < 8

0121 1A19          rlcfc    AARG+B1,W      ; rotate sign into
0122 191F          rrcfc    DPX+B3        ; for i < 8, no mea
0123 191E          rrcfc    DPX+B2        ; are in DPX+B0
0124 191D          rrcfc    DPX+B1

        else
```

Tone Generation

```

                                rlcfc  AARG+B1,W ; rotate sign into carry bit
                                rrcfc  DPX+B3
                                rrcfc  DPX+B2
                                rrcfc  DPX+B1
                                rrcfc  DPX+B0

                                fi

0002                                i = i+1

                                if i < 8 ;test low byte

0125 921A                            btffs  BARG+B0,i
                                else ; test high byte

                                btffs  BARG+B1,i-8

                                fi

0126 C12B                                goto  noadd2
add2                                movfpc AARG+B0,WREG
0127 6018                                addwf  DPX+B2 ;add lsb
0128 0F1E                                movfpc AARG+B1,WREG
0129 6019                                addwfc DPX+B3 ;add msb
012A 111F

noadd2                                if i < 8

012B 1A19                                rlcfc  AARG+B1,W ; rotate sign into
012C 191F                                rrcfc  DPX+B3 ; for i < 8, no mea
012D 191E                                rrcfc  DPX+B2 ; are in DPX+B0
012E 191D                                rrcfc  DPX+B1

                                else

                                rlcfc  AARG+B1,W ; rotate sign into carry bit

                                rrcfc  DPX+B3
                                rrcfc  DPX+B2
                                rrcfc  DPX+B1
                                rrcfc  DPX+B0

                                fi
```

Tone Generation

```
0003                                     i = i+1

                                     if i < 8           ;test low byte

012F 931A                               btfss     BARG+B0,i
                                     else
                                     ; test high byte
                                     btfss     BARG+B1,i-8

                                     fi

0130 C135                               goto     noadd3
add3
0131 6018                               movf    AARG+B0,WREG
0132 0F1E                               addwf   DPX+B2      ;add lsb
0133 6019                               movf    AARG+B1,WREG
0134 111F                               addwfc  DPX+B3      ;add msb

noadd3
                                     if i < 8

0135 1A19                               rlc    AARG+B1,W  ; rotate sign into
0136 191F                               rrcf   DPX+B3     ; for i < 8, no mea
0137 191E                               rrcf   DPX+B2     ; are in DPX+B0
0138 191D                               rrcf   DPX+B1

                                     else

                                     rlc    AARG+B1,W  ; rotate sign into carry bit
                                     rrcf   DPX+B3
                                     rrcf   DPX+B2
                                     rrcf   DPX+B1
                                     rrcf   DPX+B0

                                     fi

0004                                     i = i+1

                                     if i < 8           ;test low byte

0139 941A                               btfss     BARG+B0,i
                                     else
                                     ; test high byte
                                     btfss     BARG+B1,i-8
```

Tone Generation

```

fi
013A C13F          goto    noadd4
                    add4
013B 6018          movfp   AARG+B0,WREG
013C 0F1E          addwf  DPX+B2      ;add lsb
013D 6019          movfp   AARG+B1,WREG
013E 111F          addwfc DPX+B3      ;add msb

                    noadd4
                    if i < 8
013F 1A19          rlcfc   AARG+B1,W  ; rotate sign into
0140 191F          rrcfc   DPX+B3      ; for i < 8, no mea
0141 191E          rrcfc   DPX+B2      ; are in DPX+B0
0142 191D          rrcfc   DPX+B1

                    else

                    rlcfc   AARG+B1,W  ; rotate sign into carry bit
                    rrcfc   DPX+B3
                    rrcfc   DPX+B2
                    rrcfc   DPX+B1
                    rrcfc   DPX+B0

                    fi

0005              i = i+1

                    if i < 8              ;test low byte
0143 951A          btfss   BARG+B0,i
                    else                  ; test high byte
                    btfss   BARG+B1,i-8

                    fi

0144 C149          goto    noadd5
                    add5
0145 6018          movfp   AARG+B0,WREG
0146 0F1E          addwf  DPX+B2      ;add lsb
0147 6019          movfp   AARG+B1,WREG
0148 111F          addwfc DPX+B3      ;add msb

                    noadd5
                    if i < 8
0149 1A19          rlcfc   AARG+B1,W  ; rotate sign into
014A 191F          rrcfc   DPX+B3      ; for i < 8, no mea
```

Tone Generation

```
014B 191E          rrcf   DPX+B2   ; are in DPX+B0
014C 191D          rrcf   DPX+B1

                    else

                    rlcfc  AARG+B1,W ; rotate sign into carry bit

                    rrcf   DPX+B3

                    rrcf   DPX+B2

                    rrcf   DPX+B1

                    rrcf   DPX+B0

                    fi

0006              i = i+1

                    if i < 8          ;test low byte

014D 961A          btfss  BARG+B0,i

                    else              ; test high byte

                    btfss  BARG+B1,i-8

                    fi

014E C153          goto   noadd6
                    add6

014F 6018          movfpc AARG+B0,WREG
0150 0F1E          addwfc DPX+B2   ;add lsb
0151 6019          movfpc AARG+B1,WREG
0152 111F          addwfc DPX+B3   ;add msb

                    noadd6

                    if i < 8

0153 1A19          rlcfc  AARG+B1,W ; rotate sign into
0154 191F          rrcf   DPX+B3   ; for i < 8, no mea
0155 191E          rrcf   DPX+B2   ; are in DPX+B0
0156 191D          rrcf   DPX+B1

                    else

                    rlcfc  AARG+B1,W ; rotate sign into carry bit

                    rrcf   DPX+B3

                    rrcf   DPX+B2

                    rrcf   DPX+B1

                    rrcf   DPX+B0
```

```

                                fi
0007                                i = i+1

                                if i < 8                ;test low byte
0157 971A                btfss    BARG+B0,i
                                else                    ; test high byte

                                btfss    BARG+B1,i-8

                                fi

0158 C15D                goto    noadd7
                                add7
0159 6018                movfp   AARG+B0,WREG
015A 0F1E                addwf   DPX+B2        ;add lsb
015B 6019                movfp   AARG+B1,WREG
015C 111F                addwfc  DPX+B3        ;add msb

                                noadd7
                                if i < 8
015D 1A19                rlcfc   AARG+B1,W    ; rotate sign into
015E 191F                rrcfc   DPX+B3        ; for i < 8, no mea
015F 191E                rrcfc   DPX+B2        ; are in DPX+B0
0160 191D                rrcfc   DPX+B1

                                else

                                rlcfc   AARG+B1,W    ; rotate sign into carry bit
                                rrcfc   DPX+B3
                                rrcfc   DPX+B2
                                rrcfc   DPX+B1
                                rrcfc   DPX+B0

                                fi

0008                                i = i+1

                                if i < 8                ;test low byte

                                btfss    BARG+B0,i
```

Tone Generation

```

                                else                ; test high byte

0161 901B                        btfss      BARG+B1,i-8

                                fi

0162 C167                        goto     noadd8
                                add8

0163 6018                        movfp   AARG+B0,WREG
0164 0F1E                        addwf   DPX+B2      ;add lsb
0165 6019                        movfp   AARG+B1,WREG
0166 111F                        addwfc  DPX+B3      ;add msb

                                noadd8

                                if i < 8

                                rlcf   AARG+B1,W ; rotate sign into carry bit

                                rrcf   DPX+B3      ; for i < 8, no meaningful bits

                                rrcf   DPX+B2      ; are in DPX+B0

                                rrcf   DPX+B1

                                else

0167 1A19                        rlcf   AARG+B1,W ; rotate sign into

0168 191F                        rrcf   DPX+B3
0169 191E                        rrcf   DPX+B2
016A 191D                        rrcf   DPX+B1
016B 191C                        rrcf   DPX+B0

                                fi

0009                            i = i+1

                                if i < 8                ;test low byte

                                btfss   BARG+B0,i

                                else                ; test high byte

016C 911B                        btfss   BARG+B1,i-8

                                fi

016D C172                        goto     noadd9
                                add9

016E 6018                        movfp   AARG+B0,WREG
016F 0F1E                        addwf   DPX+B2      ;add lsb
0170 6019                        movfp   AARG+B1,WREG
0171 111F                        addwfc  DPX+B3      ;add msb

                                noadd9

                                if i < 8
```


Tone Generation

```

                                rlcfc  AARG+B1,W ; rotate sign into carry bit
                                rrcfc  DPX+B3   ; for i < 8, no meaningful bits
                                rrcfc  DPX+B2   ; are in DPX+B0
                                rrcfc  DPX+B1

                                else

0172 1A19                        rlcfc  AARG+B1,W ; rotate sign into
0173 191F                        rrcfc  DPX+B3
0174 191E                        rrcfc  DPX+B2
0175 191D                        rrcfc  DPX+B1
0176 191C                        rrcfc  DPX+B0

                                fi

000A                              i = i+1

                                if i < 8 ; test low byte

                                btfssc BARG+B0,i

                                else ; test high byte

0177 921B                        btfssc BARG+B1,i-8

                                fi

0178 C17D                        goto   noadd10
                                add10
0179 6018                        movfpc AARG+B0,WREG
017A 0F1E                        addwfc DPX+B2 ;add lsb
017B 6019                        movfpc AARG+B1,WREG
017C 111F                        addwfc DPX+B3 ;add msb

                                noadd10

                                if i < 8

                                rlcfc  AARG+B1,W ; rotate sign into carry bit
                                rrcfc  DPX+B3   ; for i < 8, no meaningful bits
                                rrcfc  DPX+B2   ; are in DPX+B0
                                rrcfc  DPX+B1

                                else

017D 1A19                        rlcfc  AARG+B1,W ; rotate sign into
017E 191F                        rrcfc  DPX+B3
017F 191E                        rrcfc  DPX+B2
```

Tone Generation

```
0180 191D          rrcf    DPX+B1
0181 191C          rrcf    DPX+B0

                    fi

000B              i = i+1

                    if i < 8
                    ;test low byte

                    btfss   BARG+B0,i

                    else
                    ; test high byte

0182 931B          btfss   BARG+B1,i-8

                    fi

0183 C188          goto    noadd11
add11
0184 6018          movfp   AARG+B0,WREG
0185 0F1E          addwf   DPX+B2    ;add 1sb
0186 6019          movfp   AARG+B1,WREG
0187 111F          addwfc  DPX+B3    ;add msb

noadd11
                    if i < 8

                    rlcfc   AARG+B1,W ; rotate sign into carry bit

                    rrcf    DPX+B3    ; for i < 8, no meaningful bits

                    rrcf    DPX+B2    ; are in DPX+B0

                    rrcf    DPX+B1

                    else

0188 1A19          rlcfc   AARG+B1,W ; rotate sign into

0189 191F          rrcf    DPX+B3
018A 191E          rrcf    DPX+B2
018B 191D          rrcf    DPX+B1
018C 191C          rrcf    DPX+B0

                    fi

000C              i = i+1

                    if i < 8
                    ;test low byte

                    btfss   BARG+B0,i

                    else
                    ; test high byte
```

Tone Generation

```
018D 941B          btfss    BARG+B1,i-8
                   fi
018E C193          goto     noadd12
                   add12
018F 6018          movfp   AARG+B0,WREG
0190 0F1E          addwf   DPX+B2      ;add lsb
0191 6019          movfp   AARG+B1,WREG
0192 111F          addwfc  DPX+B3      ;add msb
                   noadd12
                   if i < 8
                           rlcfc   AARG+B1,W ; rotate sign into carry bit
                           rrcfc   DPX+B3      ; for i < 8, no meaningful bits
                           rrcfc   DPX+B2      ; are in DPX+B0
                           rrcfc   DPX+B1
                   else
0193 1A19          rlcfc   AARG+B1,W ; rotate sign into
0194 191F          rrcfc   DPX+B3
0195 191E          rrcfc   DPX+B2
0196 191D          rrcfc   DPX+B1
0197 191C          rrcfc   DPX+B0
                           fi
000D              i = i+1
                   if i < 8 ;test low byte
                           btfss   BARG+B0,i
                   else ; test high byte
0198 951B          btfss   BARG+B1,i-8
                   fi
0199 C19E          goto     noadd13
                   add13
019A 6018          movfp   AARG+B0,WREG
019B 0F1E          addwf   DPX+B2      ;add lsb
019C 6019          movfp   AARG+B1,WREG
019D 111F          addwfc  DPX+B3      ;add msb
                   noadd13
                   if i < 8
                           rlcfc   AARG+B1,W ; rotate sign into carry bit
```

Tone Generation

```

rrcf DPX+B3 ; for i < 8, no meaningful bits
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

019E 1A19 rrcf AARG+B1,W ; rotate sign into
019F 191F rrcf DPX+B3
01A0 191E rrcf DPX+B2
01A1 191D rrcf DPX+B1
01A2 191C rrcf DPX+B0

fi

000E i = i+1

if i < 8 ;test low byte

btfss BARG+B0,i

else ; test high byte

01A3 961B btfss BARG+B1,i-8

fi

01A4 C1A9 goto noadd14
add14 movfp AARG+B0,WREG
01A5 6018 addwf DPX+B2 ;add lsb
01A6 0F1E movfp AARG+B1,WREG
01A7 6019 addwfc DPX+B3 ;add msb
01A8 111F

noadd14 if i < 8

rrcf AARG+B1,W ; rotate sign into carry bit
rrcf DPX+B3 ; for i < 8, no meaningful bits
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

01A9 1A19 rrcf AARG+B1,W ; rotate sign into
01AA 191F rrcf DPX+B3
01AB 191E rrcf DPX+B2
01AC 191D rrcf DPX+B1
01AD 191C rrcf DPX+B0
```

```

                                fi
000F                                i = i+1

                                if SIGNED
01AE 1A19                        rlcw   AARG+B1,W ; since BARG is alw
01AF 191F                        rrcw   DPX+B3 ; the last bit is k
01B0 191E                        rrcw   DPX+B2
01B1 191D                        rrcw   DPX+B1
01B2 191C                        rrcw   DPX+B0

                                endif

01B3 0002                        return
;
;*****
;*****

                                END
```

```
Errors : 0
Warnings : 0
```



Tone Generation

NOTES:

Servo Control of a DC-Brush Motor

INTRODUCTION

The PIC17C42 microcontroller is an excellent choice for cost-effective servo control in embedded applications. Due to its Harvard architecture and RISC-like features, the PIC17C42 offers excellent computation speed needed for real time closed loop servo control. This application note examines the use of the PIC17C42 as a DC brush motor servo controller. It is shown that a PID (Proportional, Integral, Differential) control calculation can be performed in less than 200 μ S (@16 MHz) allowing control loop sample times in the 2 KHz range. Encoder rates up to 3 MHz are easily handled by the PIC17C42's high speed peripherals. Further, the on-chip peripherals of the PIC17C42 allow an absolute minimum cost system to be constructed.

Closed-loop servo motor control is usually handled by 16-bit, high-end microcontrollers and external logic. In an attempt to increase performance many applications are upgrading to DSPs. However, the very high performance of the PIC17C42 makes it possible to implement these servo control applications at a significant reduction in overall system cost.

The servo system uses a PIC17C42 microcontroller, a programmable logic device (PLD), and a single-chip H-bridge driver. Such a system might be used as a positioning controller in a printer, plotter, or scanner. The low cost of implementing a servo control system using the PIC17C42 allows this system to compete favorably with stepper motor systems offering a number of advantages:

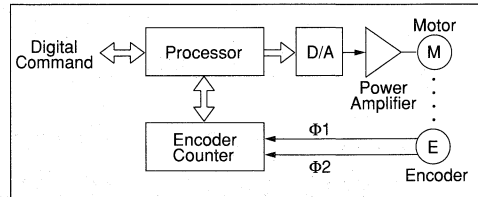
- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

SYSTEM OVERVIEW

DC Servo Control

Modern digital servo systems are formed as shown in Figure 1. These systems control a motor with an incremental feedback device known as a sequential encoder. They consist of an encoder counter, a processor, some form of digital-to-analog converter, and a power amplifier, which delivers current or voltage to the motor.

FIGURE 1 - A TYPICAL SERVO SYSTEM



The PIC17C42 implements both the servo compensator algorithm and the trajectory profile (trapezoidal) generation. A trajectory generation algorithm is necessary for optimum motion and its implementation is as important as the servo compensator itself. The servo compensator can be implemented as a traditional digital filter, a fuzzy logic algorithm, or the simple PID algorithm (implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The digital-to-analog conversion can be handled by a conventional DAC or by using pulse-width modulation (PWM). In either case the output signal is fed to a power stage which translates the analog signal(s) into usable voltages and currents to drive the motor.

PWM output can be a duty-cycle signal in combination with a direction signal or a single signal which carries both pieces of information. In the latter case a 50% duty cycle commands a null output, a 0% duty cycle commands maximum negative output, and 100% maximum positive output.

The amplifier can be configured to supply a controlled voltage or current to the motor. Most embedded systems use voltage output because of its simplicity and reduced cost.

Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of $\Phi 1$ and $\Phi 2$ represents an increment of position. The phase of the signals is used to determine direction of rotation.

Servo Control of a DC-Brush Motor

FIGURE 2 - THE PIC17C42 SERVO SYSTEM

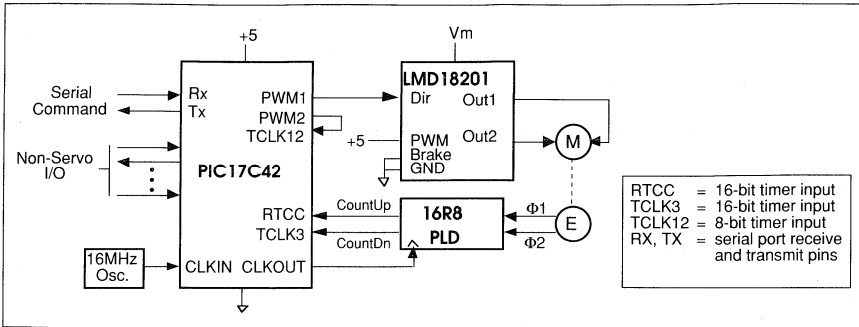


FIGURE 3 - THE PIC17C42 BASED SERVO CONTROL BOARD

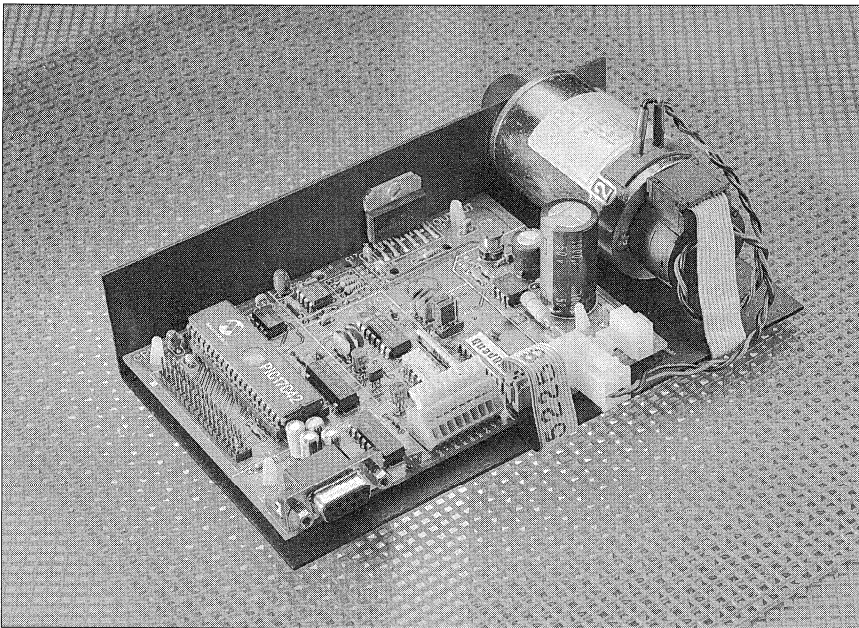
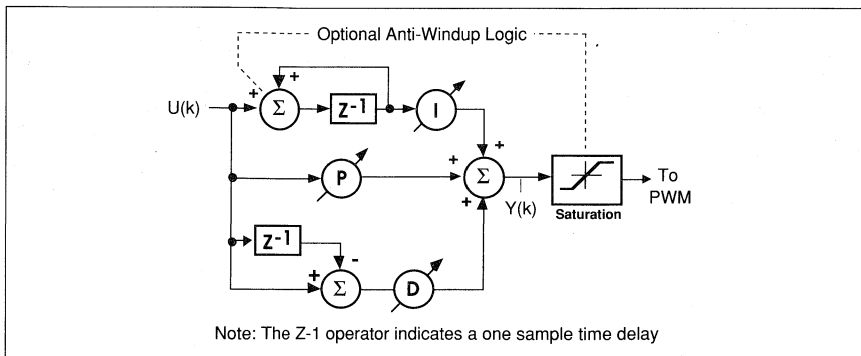


FIGURE 4 - DIGITAL PID IMPLEMENTATION



These encoder signals are usually decoded using a small state machine into Count Up and Count Down pulses. These pulses are then routed to an N-bit, up-down counter whose value corresponds to the position of the motor shaft. The decoder/counter may be implemented in hardware, software, or a combination of the two.

The PIC17C42 Based Motor Control Board

The PIC17C42 based servo system described here has a full RS-232 ASCII interface, on-board switching power supply, H-bridge motor drive, over-current protection, limit switch inputs and digital I/O. The entire system measures 5" x 3.5" and is shown in Figure 3. The system can be used to evaluate the PIC17C42 in servo applications. All unused PIC17C42 pins are available at an I/O connector for prototyping.

A PID algorithm is used as a servo compensator and position trajectories are derived from linear velocity ramp segments. This system uses 50%-null PWM as the digital-to-analog conversion technique. The power stage is a high current output switching stage which steps-up the level of the PWM signal. Encoder signal decoding is accomplished using an external PLD. The up/down counter is implemented internally in the PIC17C42 as combination of hardware and software (Figures 5 and 6).

THE COMPENSATOR

PID is the most widely used algorithm for servo motor control. Although it may not be the most optimum controller for all applications, however it is easy to understand and tune.

The standard digital PID algorithm's form is shown in Figure 4. $U(k)$ is the position or velocity error and $Y(k)$ is the output.

This algorithm has been implemented using the PIC17C42 math library. Only 800 instruction cycles are required resulting in a 0.2mS PID execution time at 16 MHz.

Integrator wind-up is a condition which occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator continually builds up during this following error condition even though the output is saturated. The integrator then "unwinds" when the servo system reaches its final destination causing excessive oscillation. The PID implementation shown above avoids this problem by stopping the action of the integrator during output saturation.

MOTOR ACTUATION

The PIC17C42 contains a high-resolution pulse width modulation (PWM) subsystem. This forms a very efficient power D/A converter when coupled to a simple switching power stage. The resolution of the PIC17C42 PWM subsystem is 62.5nS (at 16 MHz). This translates into 10-bit resolution at a 15.6KHz rate or 1 part in 800 (9 1/2-bit) resolution at 20KHz. This allows effective voltage control while still maintaining the modulation frequency at or above the limit of human hearing. This is especially relevant in office automation equipment where minimizing noise is a design goal.

The motor responds to a PWM output stage by time averaging the duty cycle of the output. Most motors react slowly, having an electrical time constant of 0.5mS or more and a mechanical time constant of 20.0mS or more. A 15KHz PWM output is effectively equivalent to that of a linear amplifier.

In the system shown in Figure 2, the H-bridge's direction input is wired directly to the PIC17C42's PWM output. The H-bridge is powered by a DC supply voltage, V_m . In this configuration 0 volts is presented to the motor when the PWM signal is at a 50% duty cycle, $-V_m$ volts at 0% duty cycle and $+V_m$ volts at 100% duty cycle.

ENCODER FEEDBACK

Position feedback for the example system is derived from a quadrature encoder mounted on the motor shaft. Both incremental position and direction can be derived from this inexpensive device. The quadrature encoder signals are processed by a 16R8-type PLD device as shown in Figure 2. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 5). These signals are then fed to two 16-bit timers of the PIC17C42 (TMR3 and RTCC). A logic description for the PLD decoder is shown in Appendix A.

The PIC17C42 keeps track of the motor shaft's incremental position by differencing these two 16-bit timers. This operation is performed each servo sample time and the current position is calculated by adding the incremental position to the previous position. Since both timers are 16-bits deep, keeping track of the overflow is unnecessary, unless the encoder signals frequency is greater than 32767 times the sample frequency. **For example, at a servo sample time of 1mS, the maximum encoder rate would be 3.2767 MHz.**

Counter wrap-around is not a concern because only the *difference* between the two counters is used. Two's-complement subtraction takes care of this automatically. Position is maintained as a three-byte, 24-bit quantity in the example program shown in Appendix F. However, there is no limit to the size of the internal position register. By adding the 16-bit incremental position each sample time to an N-byte software register, an N-byte position may be maintained.

Servo Control of a DC-Brush Motor

FIGURE 5 - SEQUENTIAL ENCODER SIGNALS

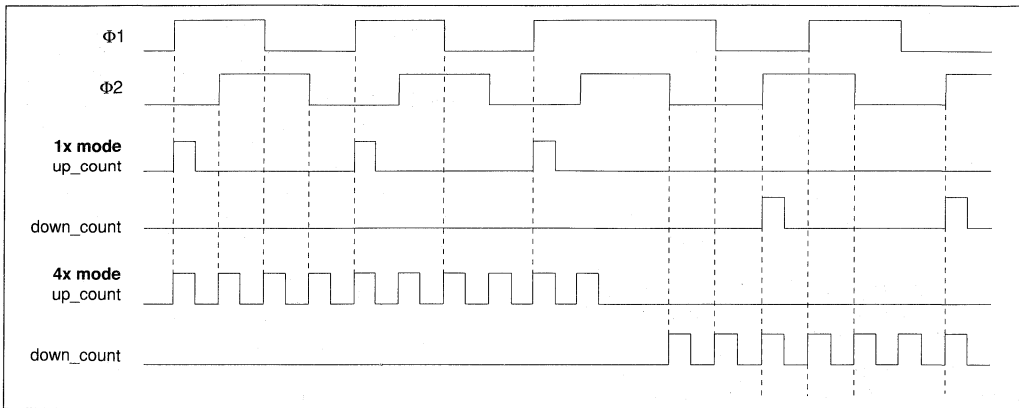
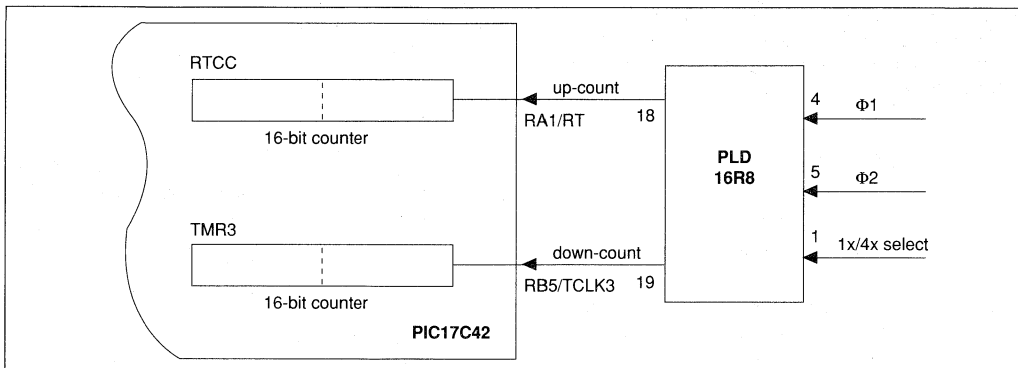


FIGURE 6 - ENCODER INTERFACE SCHEME



TRAJECTORY GENERATION

A trajectory generation algorithm is essential for optimum motion control. A linear piecewise velocity trajectory is implemented in this application. For a position move, the velocity is incremented by a constant acceleration value until a specified maximum velocity is reached. The maximum velocity is maintained for a required amount of time and then decremented by the same acceleration (deceleration) value until zero velocity is attained. The velocity trajectory is therefore trapezoidal for a long move and triangular short move where maximum velocity was not reached (Figure 8).

The doPreMove subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The doMove routine is then invoked at every sample time to calculate new "desired" velocity and position values as follows:

$$V_k = V_{k-1} + A \quad (A = \text{Acceleration})$$

$$P_k = P_{k-1} + V_{k-1} + A/2$$

For more details on trajectory generation, see Appendix E.

IMPLEMENTATION DETAILS

The program structure is straightforward: An interrupt service routine (ISR) processes the servo control and trajectory generation calculations, and a foreground loop is used to implement the user interface, serial communication and any exception processing (i.e. limit switches, watchdog timer, etc.).

The ISR has a simple structure. In order to effect servo control we need to read the encoder, calculate the new trajectory point and PID values, and set the output of the PWM, all at a constant, predefined rate. The ISR is initiated by a hardware timer (TMR2) on the PIC17C42. To make sure that the servo calculation always occurs synchronously with the PWM subsystem, the PWM2 output is wired to the input pin of TMR12 (TMR1 in internally-clocked, 8-bit timer mode; TMR2 in externally-clocked, 8-bit counter mode). N is loaded into the PR2 register. The sample rate then becomes the PWM rate divided by N. In this implementation N=16 (Figure 8).

Servo Control of a DC-Brush Motor

FIGURE 7 - SAMPLING SCHEME

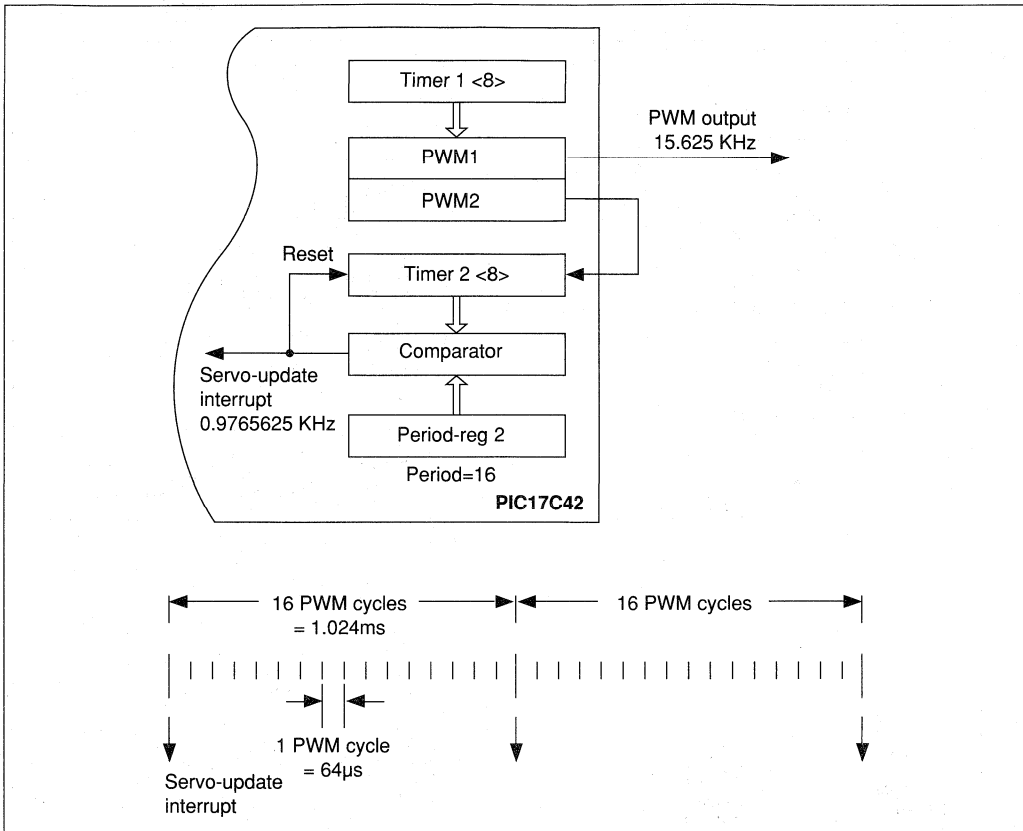
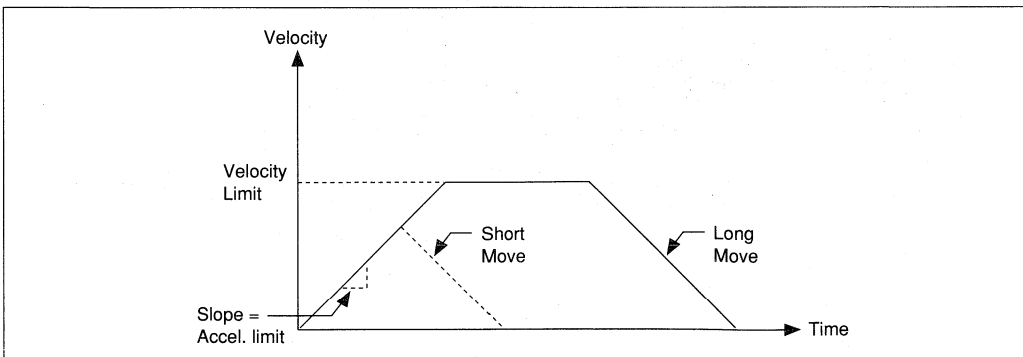


FIGURE 8 - VELOCITY RAMP SEGMENTS FOR POSITION MOVES



Servo Control of a DC-Brush Motor

FIGURE 9 - FLOWCHART FOR FOREGROUND PROCESSING

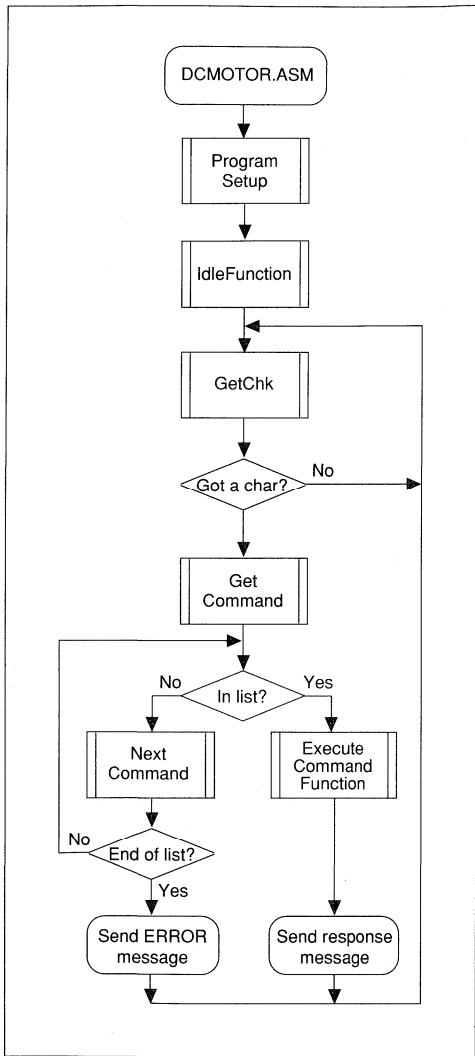
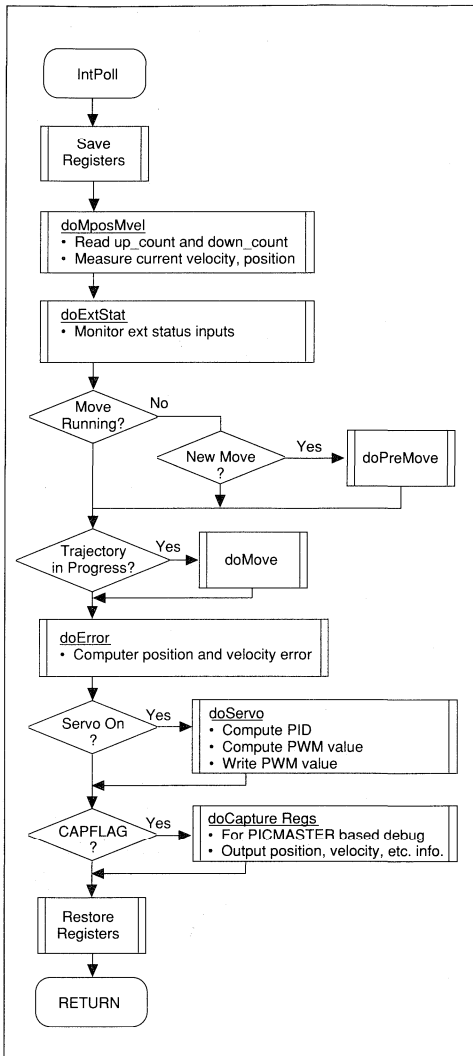


FIGURE 10 - FLOWCHART FOR INTERRUPT SERVICE ROUTINE



Servo Control of a DC-Brush Motor

The following events must occur in the interrupt service routine:

- Read Timers (RTCC & TMR3)
- Calculate the new Reference Position using the Trajectory Generation Routine.
- Calculate Error: $U(k) = \text{Reference Position} - \text{Current Position}$
- Calculate $Y(k)$ using PID
- Set PWM output
- Manage other housekeeping tasks (i.e. service serial characters)

The entire ISR requires only 0.250mS to execute with 16MHz processor clock frequency.

COMMAND INTERFACE

The following commands are implemented and recognized by the user interface in the foreground loop.

Move (Value): $M, [-8,388,608_{10} \text{ to } 8,388,607_{10}]$

Commands the axis to move to a new position or velocity. Position data is relative, velocity data is absolute. Position data is in encoder counts. Velocity data is given in encoder counts per sample time multiplied by 256. All moves are performed by the controller such that velocity and acceleration limits set into parameter memory will not be violated.

All move commands are kept in a one deep FIFO buffer. The command in the buffer is executed as soon as the executing command is complete. If no move is currently executing the commanded move will start immediately.

Mode: $Q, (\text{Type}), [P, V, T]$

An argument of "P" will cause all subsequent move commands to be incremental position moves. A "V" argument will cause all subsequent moves to be absolute velocity moves. A "T" argument sets a "Torque mode" where all subsequent M commands directly write to the PWM. This is useful for debug purposes.

Set Parameter: $S, (\#, \text{Value}) [00_n \text{ to } FF_n, -8,388,608_{10} \text{ to } 8,388,607_{10}]$

Sets controller parameters to the value given. Parameters are shown in Table 1.

TABLE 1 - PARAMETERS

Parameter	# _n	Range
Velocity Limit	00	0 to 8,388,607 ₁₀ *
Acceleration Limit	01	0 to 8,388,607 ₁₀ **
Kp: Proportional Gain	02	-32768 ₁₀ to 32767 ₁₀
Kd: Differential Gain	03	-32768 ₁₀ to 32767 ₁₀
Ki: Integral Gain	04	-32768 ₁₀ to 32767 ₁₀

* (counts per sample time multiplied by 256)

** (counts per sample time per sample time multiplied by 256)

Read Parameter: $B, (\#) [00_n \text{ to } FF_n]$

Returns the present value of a parameter.

Shutter: C

Returns the time (in sample time counts 0 to 65,536₁₀) since the start of the present move and captures the commanded and actual values of position and velocity at the time of the command.

Read commanded position: P

Returns the commanded position count which was captured during the last Shutter command.

Range: -8,388,608₁₀ to 8,388,607₁₀.

Read commanded velocity: V

Returns the commanded velocity multiplied by 256 which was captured during the last Shutter command.

Range: -8,388,608₁₀ to 8,388,607₁₀.

Read actual position: p

Returns the actual position count which was captured during the last Shutter command.

Range: -8,388,608₁₀ to 8,388,607₁₀.

Read actual velocity: v

Returns the actual velocity multiplied by 256 which was captured during the last Shutter command.

Range: -8,388,608₁₀ to 8,388,607₁₀.

External Status: X

Returns a two digit hex number which defines the state of the bits in the external status register. Issuing this command will clear all the bits in the external status register unless the event which set the bit is still true. The bits are defined in Table 2.

TABLE 2 - EXTERNAL STATUS REGISTER BITS

Bit 7	index marker detected
Bit 6	+limit reached
Bit 5	-limit reached
Bit 4	input true
Bit 7	n/a

Move Status: Y

Returns a two-digit hex number which defines the state of the bits in the move status register. Issuing this command will clear all the bits in the move status register unless the event which set the bit is still true. The bits are defined in Table 3.

TABLE 3 - MOVE STATUS REGISTER BITS

Bit 7	move buffer empty
Bit 6	move complete
Bit 5-0	n/a

Servo Control of a DC-Brush Motor

Read Index position: I

Returns the last index position captured in position counts.

Set Position (Value): H, [-8,388,608₁₀ to 8,388,607₁₀]

Sets the actual and commanded positions to the value given. Should not be sent unless the move FIFO buffer is empty.

Reset: Z

Performs a software reset.

Capture Servo-Response: c (#Count)

The c command will set a flag inside indicating that starting with the next M (servo move) command, velocity and position information will be sent out (by invoking the doCaptureRegs procedure) during every servo-loop for #count times. At the end of the #count, the processor will halt (see doCaptureRegs procedure). This is useful for debug purposes.

Disable Servo: s

This command disables servo actuation. The servo will activate again with the execution of the next M (move) command. This is useful for debug purposes.

Examples:

Z ;Reset software (No <CR> required)

OV ;Set velocity servo mode (No <CR> ;required)

M 1000<CR> ;Set velocity to 1000

M-1000<CR> ;Set velocity to 1000 in reverse ;direction

OPTIMIZING THE SYSTEM

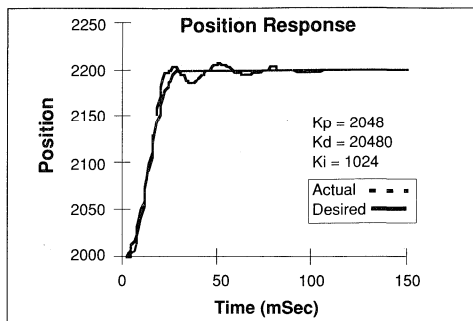
Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER In-Circuit Emulator for the PIC17C42.

The PICMASTER is a highly sophisticated real-time in-circuit emulator with unlimited break-point capability, 8K deep trace buffer and external logic probes. It's user interface software runs under Windows™ 3.1 with pull-down menus and on-line help. The PICMASTER software also support dynamic data exchange (DDE) through which it is possible to send its trace buffer information to a spreadsheet, such as EXCEL, also running under windows.

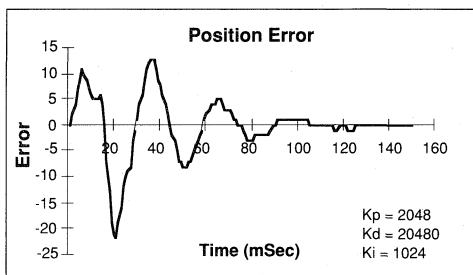
To tune the PID, first a small amount of diagnostics code is added in the servo routine (doCaptureRegs). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using TABLWT instruction. These are captured in the trace buffer of the emulator. The 'trace' condition is set up to only trace the data cycles of the 2-cycle TABLWT instructions. Next, the trace buffer is transferred to EXCEL and the various

FIGURE 11 - TYPICAL SERVO RESPONSE:

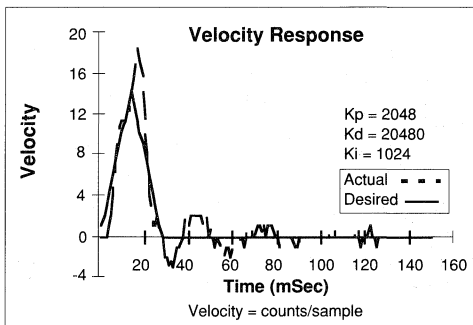
DESIRED/ACTUAL POSITION



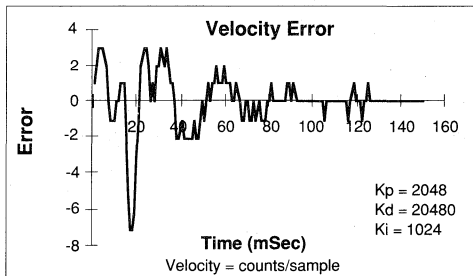
POSITION ERROR



DESIRED/ACTUAL VELOCITY



VELOCITY ERROR



Servo Control of a DC-Brush Motor

parameters are plotted. The plots graphically show the amounts of overshoot, ripple and response time. By altering K_p , K_i and K_d , and plotting the results, the system can be fine tuned.

Under windows multitasking environment, using PICMASTER emulator this can be done in real time as described below.

Three sessions are set up under windows:

1. A terminal emulator session to send commands to the motor control board. The "terminal" program provided with windows is used, although any communications software such as PROCOMM will work.
2. Second, a PICMASTER emulation session is invoked. The actual PIC17C42 is replaced in-circuit by the emulator probe. Within the emulator, trace points are setup to capture the actual and desired position and velocity values on appropriate bus cycles.
3. Third, a session of EXCEL is started and dynamically linked to the PICMASTER sessions such that whenever the trace buffer is full, the data is sent over to EXCEL. A few simple filtering commands in EXCEL are used to separate the various data types, i.e. actual position data from desired position from actual velocity etc. Next, various plot windows are set up within EXCEL to plot these information.

Once these setup have been done, for every servo move, the responses are automatically plotted. It is then a simple matter of varying the PID coefficients and observing the responses to achieve the desired system response. At any point, the responses can be stored in files and/or printed out.

Except for very long "move" commands, most position and velocity commands are executed (i.e. system settled) in less than 500 samples, making it possible to capture all variables (actual and desired position and velocity, and position errors and servo output) in PICMASTER's 8K trace buffer.

CONCLUSIONS

Using a high-performance 8-bit microcontroller as the heart of a servo control system is a cost-effective solution which requires very few external components. A comparison with a popular dedicated servo-control chip, is presented in Table 4.

TABLE 4 - SERVO CONTROL CHIP COMPARISON

	LM629 @8MHz	PIC17C42 @16MHz	PIC17C42 @25MHz
Max Encoder Rate	1MHz	3.3 MHz	4.5 MHz
Servo Update Time	-	0.25 ms	0.16 ms
Max Sampling Frequency	4 KHz	2-3 KHz	4-5 KHz

Also apparent in the comparison table is the additional processing power available when using the microcontroller. This processing can be used to provide a user interface, handle other I/O, etc. Alternatively, the additional processing time might be used to improve the performance of compensator and trajectory generation algorithms. A further advantage is that for many embedded applications using motor control the microcontroller proves to be a complete, minimum cost solution.



Credit

This application note and a working demo board has been developed by Teknic Inc. Teknic (Rochester, N.Y.) specializes in Motor Control Systems.

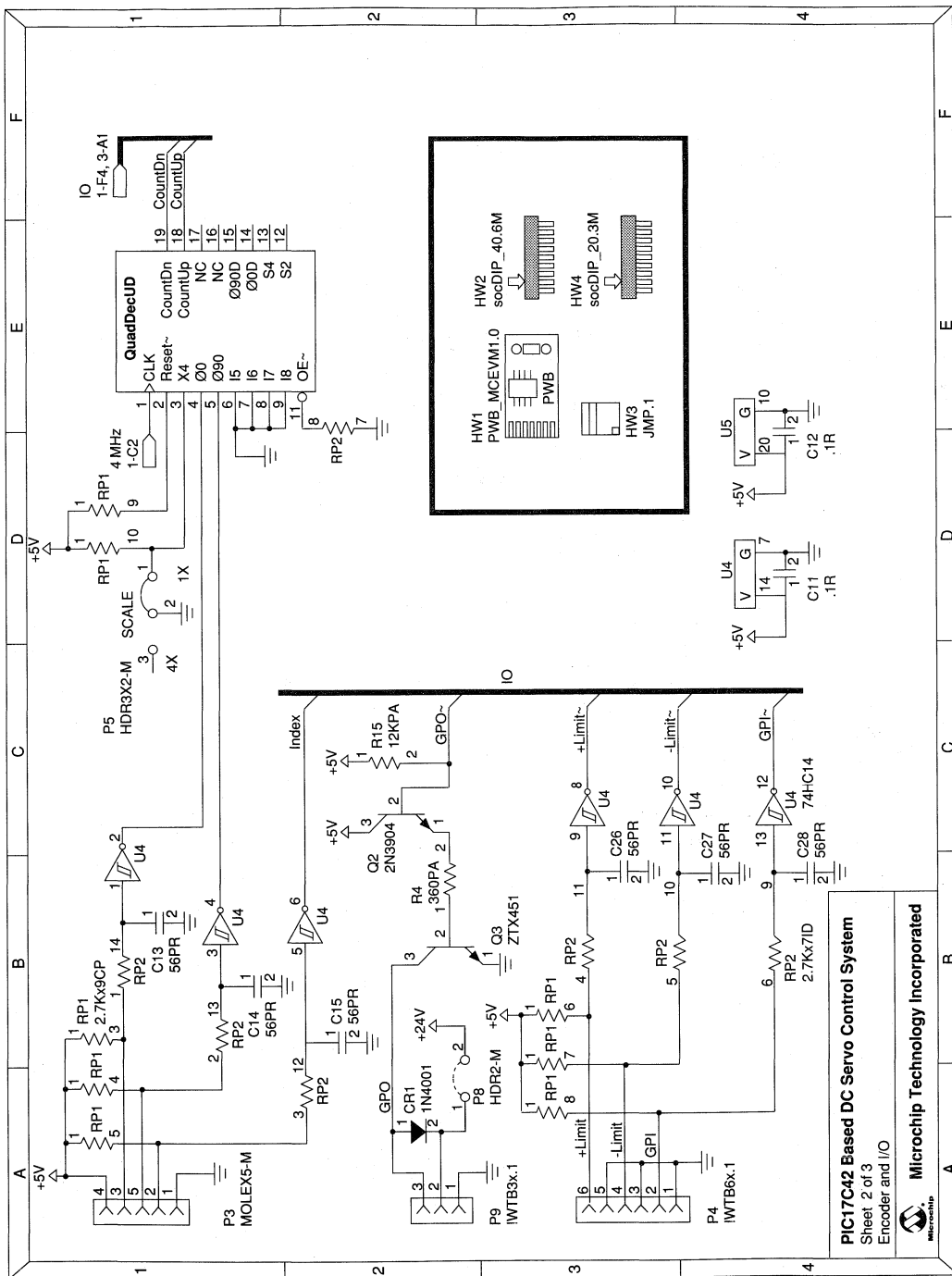
References

1. Thomas Bucella, "Comparing DSPs to Microprocessors in Motion Control Systems-Some Real World Data", PCIM conference proceedings © 1990 Intertec Communications, Inc.
2. David M. Auslander, Cheng H. Tham, "Real-Time Software for Control" © 1990 Prentice-Hall, Inc., Englewood Cliffs, NJ
3. "DC Motors, Speed Controls, Servo Systems" Fifth Edition © 1980 Electro-Craft Corporation, Hopkins, MN

Windows is a trademark of Microsoft Corporation.

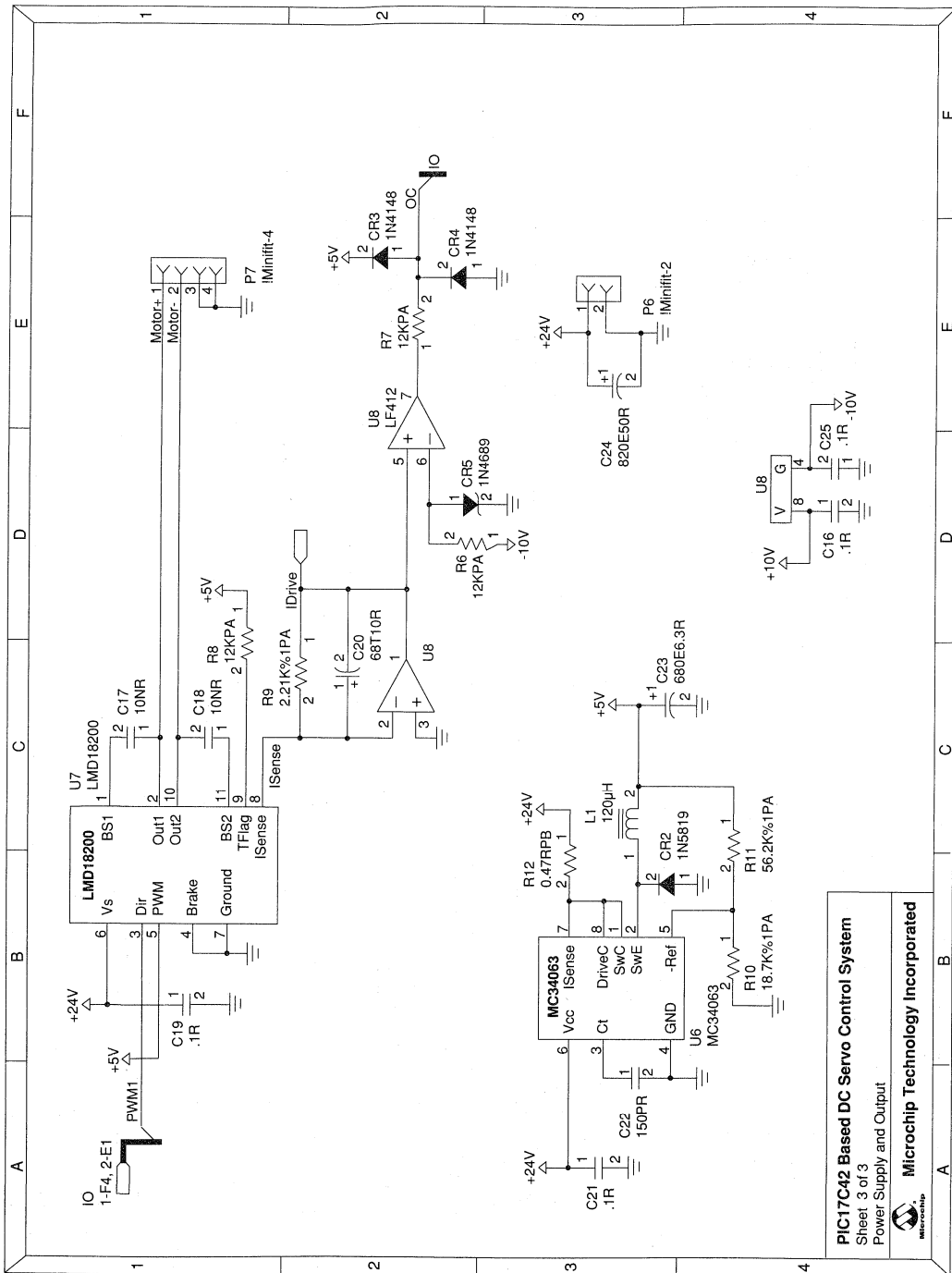
Servo Control of a DC-Brush Motor

APPENDIX A (CONT.): SCHEMATIC DIAGRAM



Servo Control of a DC-Brush Motor

APPENDIX A (CONT.): SCHEMATIC DIAGRAM



Servo Control of a DC-Brush Motor

APPENDIX B: ENCODER PLD EQUATIONS

Combination quadrature decoder and input synchronizer. This design allows 1x decoding or 4x decoding based on the X4 pin.

```
* Ver 1.0 - November 8, 1991
)
MODULE QuadDivider;
TITLE QuadDivider V1.0;
COMMENT Device: 16R8;

TYPE MMI 16R8;
INPUTS;
    RESET NODE[PIN2] INVERTED;
    X4 NODE[PIN3];
    P0 NODE[PIN4];
    P90 NODE[PIN5];
    INDX NODE[PIN6];
    { Feedback pins }
    S2 NODE[PIN12];
    S4 NODE[PIN13];
    P0D NODE[PIN14];
    P90D NODE[PIN15];
    CntUp NODE[PIN18];
    CntDn NODE[PIN19];
    UP NODE[PIN16];
    COUNT NODE[PIN17] INVERTED;
OUTPUTS;
    S2 NODE[PIN12];
    S4 NODE[PIN13];
    P0D NODE[PIN14];
    P90D NODE[PIN15];
    CntUp NODE[PIN18];
    CntDn NODE[PIN19];
    UP NODE[PIN16];
    COUNT NODE[PIN17] INVERTED;
TABLE;
    S2 := P0D & !RESET;
    S4 := P90D & !RESET;
    P0D := P0 & !RESET;
    P90D := P90 & !RESET;

    CntUp := COUNT & UP;
    CntDn := COUNT & !UP;

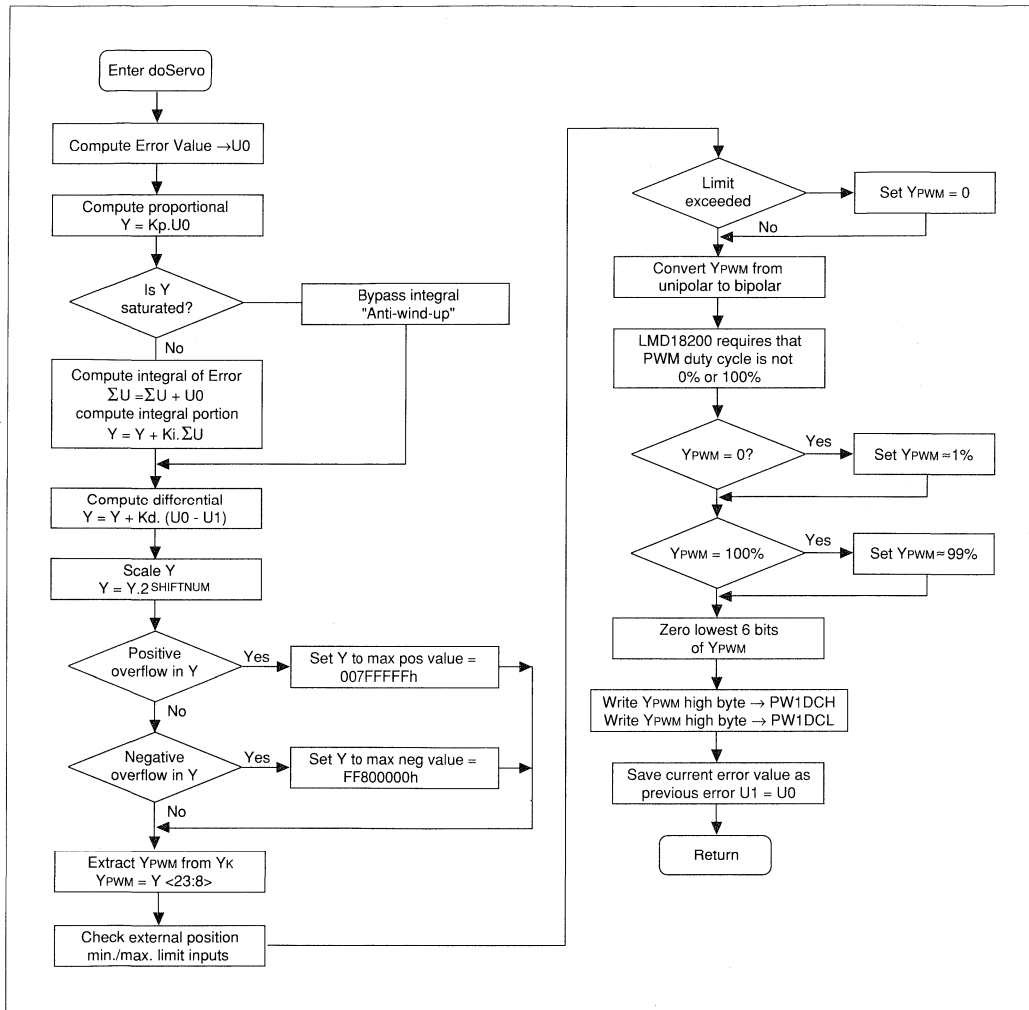
    COUNT :=
        ( P0D & S2 & !P90D & S4 & X4 { C1 }
        +!P0D & !S2 & P90D & !S4 { C2 }
        +!P0D & S2 & !P90D & !S4 & X4 { C3 }
        + P0D & !S2 & P90D & S4 & X4 { C4 }
        + P0D & S2 & P90D & !S4 & X4 { C5 }
        +!P0D & !S2 & !P90D & S4 { C6 }
        +!P0D & S2 & P90D & S4 & X4 { C7 }
        + P0D & !S2 & !P90D & !S4 & X4 { C8 }
        ) & !RESET;

    UP :=
        (
            !P0D & S2 & !P90D & S4
            +!P0D & S2 & P90D & S4
            +!P0D & S2 & P90D & !S4
            + P0D & S2 & P90D & !S4
            + P0D & !S2 & P90D & !S4
            + P0D & !S2 & !P90D & !S4
            +!P0D & !S2 & !P90D & S4
            ) & !RESET;
END;

END QuadDivider;
```

Servo Control of a DC-Brush Motor

APPENDIX C: (PART 1): PID ALGORITHM FLOWCHART



Servo Control of a DC-Brush Motor

APPENDIX C: (PART 2): PID ALGORITHM CODE LISTING

```

;*****
; NAME:          doServo
;
; DESCRIPTION:   Performs the servo loop calculations.
;
doServo
    MOV16  POSERROR,U0          ; save new position error in U0

    LOADAB  U0,KP               ; compute KP*U0
    CALL    Dmult
    MVPP32  DPX,Y              ; Y=KP*U0

    CLRF    WREG                ; if previous output saturated, do anti-
    CPFSGT  SATFLAG            ; not accumulate integrator          ] wind-up
    CALL    doIntegral

    LOADAB  INTEGRAL,KI        ; compute KI*INTEGRAL
    CALL    Dmult
    ADD32   DPX,Y              ; Y=KP*U0+KI*INTEGRAL

    MVFP16  U0,AARG            ; compute KV*(U0-U1)
    SUB16   U1,AARG
    MVFP16  KV,BARG
    CALL    Dmult
    ADD32   DPX,Y              ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)

    CLRF    WREG
    CPFSGT  SHIFTNUM           ; scale Y by SHIFTNUM
    GOTO    grabok             ; Y = Y * (2**SHIFTNUM)
    MOVFP   SHIFTNUM,TMP

grabloop
    RLC32   Y
    DECFSZ  TMP
    GOTO    grabloop

grabok
    CLRF    SATFLAG
    BTFSC   Y+B3,MSB
    GOTO    negs
    ; saturate to middle 16 bits,
    ; keeping top 10 bits for PW1DCH
    ; and PW1DCL
    MOVFP   Y+B2,WREG          ; check if Y >= 2**23
    ANDLW  0x80
    IORWF   Y+B3
    CLRF    WREG
    CPFSGT  Y+B3
    GOTO    zero6bits          ; if not, zero 6 bits

    INCF    SATFLAG           ; if so, set Y=0x007FFFFF
    CLRF    Y+B3              ; clear for debug purposes

    MOVFP   WREG,Y+B2
    SETF    Y+B1
    SETF    Y+B0
    GOTO    zero6bits

negs
    MOVFP   Y+B2,WREG          ; check if Y <= -2**23
    IORLW  0x7F
    ANDWF   Y+B3
    SETF    WREG
    CPFSLT  Y+B3
    GOTO    zero6bits          ; if not, zero 6 bits

    SETF    SATFLAG           ; if so, set Y = 0xFF800000
    SETF    Y+B3
    CLRF    Y+B2
    BSF    Y+B2,MSB
    CLRF    Y+B1
    CLRF    Y+B0

```

Basic PID calculation

anti-wind-up

Scale Y

If positive overflow, saturate y to maximum positive number

If negative overflow, saturate y to maximum negative value



Servo Control of a DC-Brush Motor

```

zero6bits
MOVZ4  Y+B1,YPWM+B0          ; move Y to YPWM and zero 6 bits
doTorque
    MOVLW  0xC0
    ANDWF  YPWM+B0

    BTFSC  YPWM+B1,MSB
    GOTO   tmlimit

tmlimit
    BTFSS  EXTSTAT,BIT6
    GOTO   mplimitok
    CLR32  YPWM
    GOTO   mplimitok

tmlimit
    BTFSS  EXTSTAT,BIT5
    GOTO   mplimitok
    CLR32  YPWM

mplimitok
    MOVLW  PW1DCH_INIT        ; adjustment from bipolar to unipolar
    MOVFP  WREG,TMP+B1        ; for 50% duty cycle
    MOVLW  PW1DCL_INIT
    MOVFP  WREG,TMP+B0
    ADD16  TMP,YPWM

    CLRF   TMP+B1            ; correct by 1 LSB
    MOVLW  0x40              ; add one to bit5 of PW1DCL
    MOVFP  WREG,TMP+B0
    ADD16  TMP,YPWM

testmax
    CLRF   TMP+B2            ; check pwm maximum limit
    CLRF   YPWM+B2          ; LMD18200 must have a minimum pulse
    CLRF   YPWM+B3          ; so duty cycle must not be 0 or 100%
    MVFP16 YPWM_MAX,TMP
    SUB24  YPWM,TMP
    BTFSS  TMP+B2,MSB
    GOTO   testmin
    MOV16  YPWM_MAX,YPWM    ; saturate to max
    GOTO   limitok

testmin
    CLRF   TMP+B2            ; check pwm minimum limit
    CLRF   YPWM+B2
    CLRF   YPWM+B3
    MVFP16 YPWM_MIN,TMP
    SUB24  YPWM,TMP
    BTFSC  TMP+B2,MSB
    GOTO   limitok
    MOV16  YPWM_MIN,YPWM    ; saturate to min

limitok
    MOVLB  BANK3            ; set new duty cycle
    MOVFP  YPWM+B0,PW1DCL
    MOVFP  YPWM+B1,PW1DCH

    MOV16  U0,U1            ; push errors into U(k-1)

RETURN

```

If external position limits have been reached then zero PWM output.

Convert PWM from unipolar to bipolar

PWM cycle must not be 0% of 100%

Write PWM values to PWM registers

Servo Control of a DC-Brush Motor

APPENDIX D: ENCODER INTERFACE ROUTINE

```
*****
; NAME: doMPosMVel
;
; DESCRIPTION: Calculates current position from UpCount and DownCount
;
doMPosMVel

; Do UpCounter first

readUp
    MVFP16   UPCOUNT,TMP+B0      ; save old upcount
    MOVFP   RTCCH,WREG
    MOVFP   RTCC,UPCOUNT+B0
    CPFSEQ  RTCCH                ; Skip next if HI hasn't changed
    GOTO   readUp               ; HI changed, re-read LO
    MOVFP   WREG,UPCOUNT+B1    ; OK to store HI now

    CLRF    MVELOCITY+B0       ; clear bits below binary point

    MOV16   UPCOUNT,MVELOCITY+B1 ; compute upcount increment
    SUB16   TMP+B0,MVELOCITY+B1

; Now do DownCounter

readDown
    MVFP16   DOWNCOUNT,TMP+B0  ; save old downcount

    MOVLB   BANK2                ; timers in Bank 2
    MOVFP   TMR3H,WREG
    MOVFP   TMR3L,DOWNCOUNT+B0
    CPFSEQ  TMR3H                ; Skip next if HI hasn't changed
    GOTO   readDown             ; HI changed, re-read LO
    MOVFP   WREG,DOWNCOUNT+B1  ; OK to store HI now

    MVFP16   DOWNCOUNT+B0,TMP+B2 ; compute downcount increment
    SUB16   TMP+B0,TMP+B2

    SUB16   TMP+B2,MVELOCITY+B1  ; compute new measured velocity

    CLRF    MVELOCITY+B3       ; sign extend measured velocity for
    BTFSC   MVELOCITY+B2,MSB    ; 24 bit addition to measured posi-
tion
    SETF    MVELOCITY+B3

    ADD24   MVELOCITY+B1,MPOSITION; compute new measured position
                                           ; delta position = measured velocity

    RETURN

*****
```

Servo Control of a DC-Brush Motor

APPENDIX E: IMPLEMENTATION DETAILS OF TRAJECTORY GENERATION

doPreMove:

This routine is executed only once at the beginning of each move. First, various buffers and flags are initialized and a test for modetype is performed. In position mode, the minimum move is triangular and consists of two steps. Therefore, if $abs(MOVVAL) > 2$, an immediate move is performed. Otherwise, normal move generation is possible with the sign of the move in MOVSIGN and the appropriate signed velocity and acceleration limits in V and A, and $MOVVAL/2$ in HMOVVAL.

In velocity mode, the sign of the move is calculated in MOVSIGN and the appropriate signed velocity and acceleration limits are placed in V and A. Finally, at modeready, MOVVAL is sign extended for higher precision arithmetic and the servo is enabled.

In torque mode, MOVVAL is output directly to the PWM and the servo is disabled, and doMove is not executed.

doMove:

Move generation is based on a piecewise constant acceleration model. During constant acceleration, this results in the standard equations for position and velocity given by

$$x(t) = x_0 + v_0 t + \frac{a}{2} t^2, v(t) = v_0 + a t$$

With the units for t in sample times, the time increment between subsequent sample times is 1, yielding the iterative equations for updating position and velocity implemented in doPosVel and given by

$$P(k) = P(k-1) + V(k-1) + A/2, \quad V(k) = V(k-1) + A,$$

where A is the signed acceleration limit calculated in doPreMove. The inverse equations of this iteration, necessary for undoing an unwanted step, are contained in undoPosVel and given by

$$P(k-1) = P(k) - V(k-1) - A/2, \quad V(k-1) = V(k) - A.$$

In position mode, the actual shape of the velocity profile depends on the values of V, A and the size of the move. Either the velocity limit is reached before half the move is completed, resulting in a trapezoidal velocity profile, or half the move is completed before the velocity limit is realized, resulting in a triangular velocity profile.

In the algorithm employed here, the velocity limit is treated as a bound on the actual velocity limit, thereby permitting exactly the same number of steps during the speedup and speeddown sections of the move. Phase 1 is defined as the section of the move where the commanded position is less than half the move, and phase 2 is the remaining portion of the move. T1 is time when the actual velocity limit is reached and T2 is the time at the end of phase 1.

FIGURE A - SPEED PROFILE FOR TRAPEZOIDAL MOVES

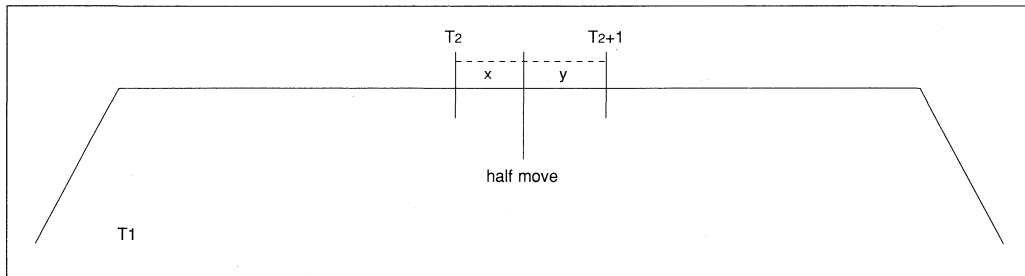


FIGURE B - SPEED PROFILE FOR TRIANGULAR MOVES

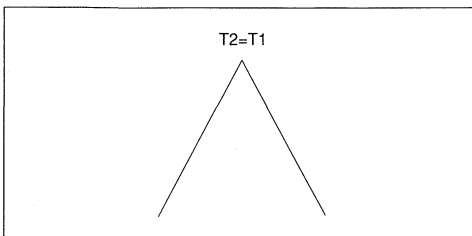
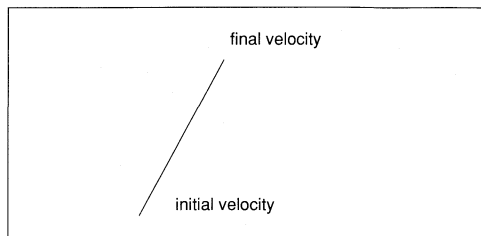


FIGURE C - SPEED PROFILE FOR VELOCITY MOVES



Servo Control of a DC-Brush Motor

Furthermore, let x be the amount of undershoot and y the amount of overshoot of half the move at $T2$. Discretization error is minimized by using the values of x and y whether one more step will reduce the size of the final immediate move during the last step of the move. For a triangular move, the discretization error is given by $\min.(2x, 2y)$, resulting in the condition that if $2x > 2y$, then take one more speedup step. In the case of a trapezoidal move, the discretization error is given by $\min.(2x, y-x)$, yielding the condition that if $3x > y$, take one more step during the flat section of phase2.

At the beginning of `doMove`, `MOVTIME` is incremented and `doPosVel` is called to evaluate the next proposed values of commanded position and velocity under the current value of A . In position mode, phase1, the original position plus half the move minus the new proposed commanded position is calculated and placed in `MOVDEL`, with the previous `MOVDEL` saved in `MOVTMP`. As half the move would be passed, `MOVTMP` = $-x$ and `MOVDEL` = y , with $y > 0$ for the first time indicating that phase1 is about to be completed. Therefore, if $y < 0$, we continue in phase1, where if maximum velocity has not been reached, the new proposed commanded position is executed. On the other hand, if the proposed move would exceed the maximum velocity, we undo the proposed move, set the current acceleration to zero, reevaluate the iterative equations with the new acceleration, set $T1 = \text{MOVTIME} - 1$, and execute the move.

Since $T1$ is cleared in `doPreMove`, it is used as a flag to indicate if this corner in the velocity profile has been reached. Once we find that $y > 0$, we drop into code that is executed only one time, with phase2 beginning on the next step. If $T1 = 0$, maximum velocity has not yet been

reached, so $T1 = T2$ and the velocity profile is triangular. In this case, A is negated for speeddown, and if $x > y$, one more step is needed to minimize the discretization error. So A is negated, the proposed step undone, A is again negated for speeddown and the step recalculated and executed, with $T2 = T1 = \text{MOVTIME} - 1$.

If $T1$ is not zero, indicating that we are in the flat section of phase1, then go to `t2net1`, where $T2 = \text{MOVTIME} - 1$, and if $3x > y$, then one more phase2 flat step is necessary to minimize the discretization error. `PH2FLAT` is defined as the number of steps in the flat section of phase2, and is used as a counter during its completion. If $3x > y$, then $\text{PH2FLAT} = T2 - T1$, otherwise $\text{PH2FLAT} = T2 - T1 - 1$ and phase1 is finally complete. All subsequent steps will proceed through phase2, first deciding if the flat section is finished by checking if `PH2FLAT` has reached zero. If not, go to flat where `PH2FLAT` is decremented, and tested if zero. If so, the speeddown section is begun by calculating the appropriate signed acceleration limit A , and executing the last of the flat section moves. For all following steps, `PH2FLAT` = 0, leaving only the final test for zero commanded velocity to indicate the end of the move. This will always occur since the actual maximum velocity, bounded above by the user supplied limit, is always an integer multiple of the user supplied acceleration limit, with exactly the same number of steps taken during speedup and speeddown.

The velocity mode is much more straightforward, with the velocity profile in the form of a ramp. If the final velocity has not been reached, the move continues at maximum acceleration. If the final velocity has been reached, the acceleration is set to zero and the move generation of commanded position and velocity continued unless the final velocity is zero.

Servo Control of a DC-Brush Motor

APPENDIX F: COMPLETE CODE LISTING (DCMOTOR.LST)

MPASM B0.54

PAGE 1

"Revision: 2.0, 27 June 93"

```
SubTitle      "Revision: 2.0, 27 June 93"

;*****
;
;   Revised: 8/5/92
;   CREDIT:  Developed by Teknic Inc. 1992
;
;   Assembled using MPASM. User's with ASM17 are suggested to get Microchip's
;   new universal assembler (MPASM). To assemble with ASM17, all "if", "else"
;   "endif" directives must be replaced by "#if", "#else" and "#endif"
;   respectively.
;*****

        PROCESSOR    PIC17C42
        LIST         COLUMNS=120, XREF=YES, NOWRAP, LINES=255, R=DEC

;*****
;
00F4 2400    MASTER_CLOCK    set    16000000        ; Input Clock Freq in Hz
03E8                _SAMPLE_RATE    set    1000        ; Sample rate in Hz
07D0                _ENCODER_RATE    set    2000        ; 2000 Pulses/rev
1770                _RATED_SPEED    set    6000        ; in RPM
2580                _BAUDRATE_    set    9600

;*****

        include "17c42.h"

        include "17c42.mac"    ; General Purpose Macros

0058                #define _PICMASTER_DEBUG    TRUE        ; Enable PIC-MASTER TRACE Capture.
0059                #define _SERVO_PID    TRUE        ; PID computation based on error
005A                #define DECIO    TRUE        ; true for decimal, false for hex
005B                #define _SERIAL_IO    TRUE

        include "dcmotor.h17"    ; Initialization, Global Defs,
;*****
;
; Header file for dcmotor.asm:
; Revised: 8/5/92
;*****
;
; hardware constants
;
;

005C                #define _SET_BAUD_RATE(bps)    ((10*MASTER_CLOCK/(64*bps))+5)/10 - 1
003D 0900    CLKOUT        set    MASTER_CLOCK >> 2    ; Clock Out = CLKIN/4

0006                TCON1_INIT    set    0x06
003F                TCON2_INIT    set    0x3F
00FF                PR1_INIT    set    0xFF        ; set pwm frequency to CLKOUT/256
```

Servo Control of a DC-Brush Motor

```

000F          PR2_INIT          set      ((10*MASTER_CLOCK/(4*(PR1_INIT+1)*_SAMPLE_RATE))+5)/
;pwldch_INIT  set      ((PR1_INIT+1) << 8) >> 9
;pwldcl_INIT  set      (((PR1_INIT+1) << 8) >> 1) & 0xff

007F          PWLDCH_INIT       set      0x7F
00C0          PWLDCL_INIT       set      0xC0

0080          RTCSTA_INIT       set      0x80
0090          RCSTA_INIT        set      0x90
0020          TXSTA_INIT        set      0x20
0019          SPBRG_INIT        set      _SET_BAUD_RATE(_BAUDRATE_)

00F3          DDRB_INIT         set      0xF3
0000          DDRD_INIT         set      0x00
;
;
;          max and min pwm values
;
0040          PWWMINL           set      0x40
0001          PWWMINH           set      0x01          ; 0x0000 + 0x0140 (min 10 bit
0080          PWWMAXL           set      0x80
00FE          PWWMAXH           set      0xFE          ; 0xFFC0 - 0x0140 ( max 10 bit
;
;
;*****
;          global variables
;
          CBLOCK 0x18
0018 0004    DPX,DPX1,DPX2,DPX3          ; arithmetic accumula
001C 0004    AARG,AARG1,BARG,BARG1      ; multiply arguments
          ENDC

          CBLOCK 0x18
0018 0004    TMP,TMP1,TMP2,TMP3         ; temporary variables
001C 0004    MOVTMP,MOVTMP1,MOVTMP2,MOVTMP3 ; move temporary
          ENDC

          CBLOCK 0x20
0020 0003    VL,VL1,VL2                 ; velocity limit
0023 0003    AL,AL1,AL2                 ; acceleration limit
0026 0000
0026 0002    KP,KP1                     ; proportional gain
0028 0002    KV,KV1                     ; velocity gain
002A 0002    KI,KI1                     ; integral gain
002C 0000
002C 0001    IM                         ; integrator mode
002D 0002    FV,FV1                     ; velocity feedforward
002F 0002    FA,FA1                     ; acceleration
0031 0000
0031 0003    VALBUF,VALBUF1,VALBUF2     ; iovalue buffer
0034 0000
0034 0003    DVALBUF,DVALBUF1,DVALBUF2  ; iovalue buffer
0037 0002    ISRBSR,ISRWREG             ; isr save storage
0039 0004    CMDCHAR,CMDTEMP,CMDPTRH,CMDPTRL ; command interface
003D 0003    PARTEMP,PARLEN,PARPTR      ; parameter variables
0040 0000
0040 0003    CPOSITION,CPOSITION1,CPOSITION2 ; shutter commanded
0043 0003    CVELOCITY,CVELOCITY1,CVELOCITY2 ; shutter commanded
0046 0003    CMPOSITION,CMPOSITION1,CMPOSITION2 ; shutter measured
0049 0003    CMVELOCITY,CMVELOCITY1,CMVELOCITY2 ; shutter measured
004C 0000
004C 0002    STRVALH,STRVALL            ; string io variables
004E 0003    HEXVAL,HEXTMP,HEXSTAT      ; hex io variables
0051 0000
0051 0002    OPOSITION,OPOSITION1
0053 0002    OPOSITION2,OPOSITION3      ; original commanded
0055 0003    POSITION,POSITION1,POSITION2 ; commanded position
0058 0003    VELOCITY,VELOCITY1,VELOCITY2 ; commanded velocity
005B 0000

```

Servo Control of a DC-Brush Motor

```

005B 0004      NMOVVAL,NMOVVAL1,NMOVVAL2,NMOVVAL3      ; move value
005F 0004      MOVVAL,MOVVAL1,MOVVAL2,MOVVAL3      ; move value
0063 0004      HMOVVAL,HMOVVAL1,HMOVVAL2,HMOVVAL3      ; half move value
0067 0002      MOVTIME,MOVTIME1                  ; move time in sample
0069 0000
0069 0001      MOVSIGN                            ; 0x00 for positive,
0x80 for
006A 0002      T1,T11                            ; time to maximum
006C 0002      T2,T21                            ; time for half the
006E 0002      TAU,TAU1                          ; total move time
0070 0001      NMODE                              ; next move modetype
0071 0001      MODE                              ; move modetype
0072 0000
0072 0003      MPOSITION,MPOSITION1,MPOSITION2      ; measured position
0075 0002      MVELOCITY,MVELOCITY1
0077 0002      MVELOCITY2,MVELOCITY3              ; measured velocity
0079 0003      POSERROR,POSERROR1,POSERROR2        ; position error
007C 0003      VELERROR,VELERROR1,VELERROR2        ; velocity error
007F 0000
007F 0001      SIGN                              ; multiply sign
0080 0000
0080 0004      Y,Y1,Y2,Y3                        ; Y(k) before pwm
0084 0004      U0,U01,U1,U11                     ; saturated error at
0088 0000
0088 0004      YPWM,YPWM1,YPWM2,YPWM3            ; pwm input
008C 0004      YPWMIN,YPWMIN1,YPWMAX,YPWMAX1      ; pwm input limits
0090 0000
0090 0001      SERVFLAG                          ; servoflag = 0 => no
0091 0001      MODETYPE                          ; mode
0092 0001      EXTSTAT                            ; external status
0093 0001      MOVSTAT                            ; move status register
0094 0001      MOVFLAG                            ; move flag
0095 0001      SATFLAG                            ; saturation flag
0096 0002      INTEGRAL,INTEGRAL1                ; integrator
0098 0000
0098 0004      DECVAL,DECSTAT,DECTMP,DECSIGN      ; decimal io variables
009C 0000
009C 0004      A,A1,A2,A3                        ; commanded accelera
V,V1,V2,V3                                        ; commanded velocity =
00A0 0004      MOVVBUF,MOVVBUF1,MOVVBUF2,MOVVBUF3 ; commanded position
00A8 0004      MOVVBUF,MOVVBUF1,MOVVBUF2,MOVVBUF3 ; commanded velocity
00AC 0000
00AC 0002      UPCOUNT,UPCOUNT1                ; running up counter
00AE 0002      DOWNCOUNT,DOWNCOUNT1             ; running down counter
00B0 0000
00B0 0004      MOVDEL,MOVDEL1,MOVDEL2,MOVDEL3    ; move discretization
00B4 0002      PH2FLAT,PH2FLAT1                  ; phase 2 flat itera
00B6 0003      INDEXPOS,INDEXPOS1,INDEXPOS2      ; position at last
00B9 0000
00B9 0001      SHIFTRNUM                          ; # of bit shifts from
00BA 0000
00BA 0001      if      _PICMASTER_DEBUG
00BB 0000      ;*****
00BB 0000      ;      For PICMASTER Debug/servo tuning Purposes Only
00BB 0000
00BB 0001      CAPFLAG                            ; trace capture flag
00BC 0002      CAPCOUNT,CAPCOUNT1              ; PICMASTER trace
00BE 0002      CAPTMP,CAPTMP1                    ; trace capture
00C0 0000
00C0 0000      ;*****

```

Servo Control of a DC-Brush Motor

```
00C0 0001          endif
00C1 0000
00C1 0002          ZERO,ONE          ; constants
00C3 0002          tblptrlTemp,tblptrhTemp ; temp TABLE Pointers for
00C5 0002          TblLatLo, TblLatHi    ; Table Latch for ISR save
00C7 0001          alustaTemp          ; temp alusta
00C8 0000
```

ENDC

```
;*****
; NAME:          AUTONO
;
; DESCRIPTION:   Sets no auto increment or decrement
;
; TIMING (cycles): 4
```

AUTONO MACRO

```
BSF    _fs0
BSF    _fs1
BSF    _fs2
BSF    _fs3
```

ENDM

```
;*****
; NAME:          AUTOINC
;
; DESCRIPTION:   Set auto increment
;
; TIMING (cycles): 4
;
```

AUTOINC MACRO

```
BSF    _fs0
BCF    _fs1
BSF    _fs2
BCF    _fs3
```

ENDM

```
;*****
; NAME:          AUTODEC
;
; DESCRIPTION:   Sets auto decrement
;
; TIMING (cycles): 4
;
```

AUTODEC MACRO

```
BCF    _fs0
BCF    _fs1
BCF    _fs2
BCF    _fs3
```

ENDM

```
;*****
;*****
```

Servo Control of a DC-Brush Motor

```

; NAME:          LOADAB
;
; DESCRIPTION:   Loads extended math library AARG and BARG
;
; ARGUMENTS:    A => AARG
;               B => BARG
;
; TIMING (cycles): 4

LOADAB MACRO    A,B

MOVFP  A+B0,AARG+B0 ; load lo byte of A to AARG
MOVFP  A+B1,AARG+B1 ; load hi byte of A to AARG
MOVFP  B+B0,BARG+B0 ; load lo byte of B to BARG
MOVFP  B+B1,BARG+B1 ; load hi byte of B to BARG

ENDM

;*****
;*****
;      ascii constants
;
000D      CR      set    0x0D
0018      CAN      set    0x18
0008      BS      set    0x08
0020      SP      set    0x20
000A      LF      set    0x0A
002D      MN      set    '-'
;
;*****
;      cmds constants and macros
;
;
0001      CHARREADY      set    0x01
;
0008      NUMPAR      set    0x08
;
; Response characters
;
0021      CMD_OK      set    '!'
003F      CMD_BAD      set    '?'
;
; Exit values
;
;
0000      HEX_SP      set    0x00
0001      HEX_MN      set    0x01
0002      HEX_CR      set    0x02
0003      HEX_CAN      set    0x03
;
0000      DEC_SP      set    0x00
0001      DEC_MN      set    0x01
0002      DEC_CR      set    0x02
0003      DEC_CAN      set    0x03
;
;
; Command characters
;
;
000D      DO_NULL      set    CR
004D      DO_MOVE      set    'M' ; M
004F      DO_MODE      set    'O' ; O
0053      DO_SETPARAMETER      set    'S' ; S
0052      DO_READPARAMETER      set    'R' ; R
0043      DO_SHUTTER      set    'C' ; C
0050      DO_READCOMPOSITION      set    'P' ; P
0056      DO_READCOMVELOCITY      set    'V' ; V

```

Servo Control of a DC-Brush Motor

```
0070 DO_READACTPOSITION set 'p' ; p
0076 DO_READACTVELOCITY set 'v' ; v
0058 DO_EXTERNALSTATUS set 'X' ; X
0059 DO_MOVESTATUS set 'Y' ; Y
0049 DO_READINDPOSITION set 'I' ; I
0048 DO_SETPOSITION set 'H' ; H
005A DO_RESET set 'Z' ; Z
0073 DO_STOP set 's' ; s
0063 DO_CAPTURE set 'c' ; c

;*****
; NAME: CMD_DEF
;
; DESCRIPTION: Creates all the definitions for a command table data structure. The first word is at the command character used, and the second word is a pointer to the function that handles this command function.
;
; ENTRY CONDITIONS: Must be contiguous with the other entries for the function to work.
;
; ARGUMENTS: FUNC command execution function
; ROOT NAME ROOT
;

CMD_DEF MACRO FUNC,ROOT
;
; DATA ROOT
; DATA FUNC
; ENDM

0002 CMD_ENTRY_LENGTH set 2

;*****
; NAME: CMD_START
;
; DESCRIPTION: Labels the start of the command table.
;

CMD_START MACRO LABEL
LABEL
; ENDM

;*****
; NAME: CMD_END
;
; DESCRIPTION: Marks the end of the command table with an entry of 0x00
;

CMD_END MACRO
;
; DATA 0x00
; ENDM

;*****
;
;
; PID Constantnts
; define PIV parameters, computation based on errors only. Does not involve
```



Servo Control of a DC-Brush Motor

```

;
; No Load @ 2Khz :           Kp=3600, Ki=112, Kd= 28800, Shiftcount = 3
; With Indicator Load @ 2Khz, Kp=2300 Ki= 41, Kd= 32200, Shiftcount = 4
; " " @ 4Khz                Kp=1024, Ki=8, Kd=31405 , ShiftCount = 5
; " " @ 0.5 Khz             Kp=5400, Ki=310, Kd=23400 , ShiftCount = 2
;
; No Load @ 1Khz           Kp=3600, Ki=192, Kd=16800, ShiftNum = 3
; No Load @ 1Khz           Kp=1800, Ki=52, Kd=15600, ShiftNum = 4
;
; Adjust Shiftcount by maximizing Kd (for a 16 bit signed num)
;
;*****
005D      #define _KP      1800                ; 16 bit Kp
005E      #define _KI      52                ; 16 bit Ki
005F      #define _KV      15600            ; 16 bit Kv

3200      _VEL_LIMIT      set      (( _RATED_SPEED*_ENCODER_RATE)/      ; 1/4 Rated sp
2000      _ACCL_LIMIT      set      0x2000                ; use smaller

          if _SERVO_PID == TRUE
0004      _SHIFTNUM      equ      4
          endif

;*****

          ORG      0x0                ; reset vector
0000 C021      goto      Startup                ; startup

          ORG      0x20
0020 C07D      goto      InterruptPoll        ; interrupt

;*****
; NAME:      Startup
;
; DESCRIPTION: This routine is called on the hardware reset or when the
;              program wishes to restore initial conditions. Initiali-
;              zation of run-time constants takes place here.
;
; RETURNS:   restart to safe and initial state
;
; STACK UTILIZATION: none
; TIMING (in cycles): X

Startup

0021 8406      bsf      _glintd                ; disable all
                                AUTONO
                                ; no auto

0022 8404      BSF      _fs0
0023 8504      BSF      _fs1
0024 8604      BSF      _fs2
0025 8704      BSF      _fs3

0026 B018      movlw   0x18                ; clear all
0027 4A01      movpf   wreg,fsr0

          memloop
0028 2900      clrf   indf0
0029 1F01      incfsz  fsr0
002A C028      goto   memloop

002B 15C2      incf   ONE

```


Servo Control of a DC-Brush Motor

```

002C B803          movlb   bank3          ; bank3 ini
002D B03F          movlw   TCON2_INIT
002E 770A          movfp   wreg,tcon2

002F B07F          movlw   PW1DCH_INIT        ; set duty
0030 720A          movfp   wreg,pw1dch
0031 730A          movfp   wreg,pw2dch

0032 B0C0          movlw   PW1DCL_INIT
0033 700A          movfp   wreg,pw1dcl
0034 710A          movfp   wreg,pw2dcl

0035 B006          movlw   TCON1_INIT        ; set organization of timers
0036 760A          movfp   wreg,tcon1

0037 B802          movlb   bank2          ; bank2 initialization

0038 B0FF          movlw   PR1_INIT
0039 740A          movfp   wreg,pr1        ; initialize timer1 period

003A B00F          movlw   PR2_INIT
003B 750A          movfp   wreg,pr2        ; initialize timer2 period

003C B800          movlb   bank0          ; bank0 initialization

003D B080          movlw   RTCSTA_INIT
003E 650A          movfp   wreg,rtcsta      ; sets RTC for external input

003F B0F8          movlw   0xF8          ; RA2 connected to BREAK Input of LMD18200
0040 0110          movwf   porta        ; On Reset, thus pulled high breaking the
motor

                if _SERIAL_IO

0041 B090          movlw   RCSTA_INIT
0042 730A          movfp   wreg,rcsta      ; set receive status

0043 B020          movlw   TXSTA_INIT        ; set transmit status
0044 750A          movfp   wreg,txsta

0045 B019          movlw   SPBRG_INIT      ; set baud rate
0046 770A          movfp   wreg,spbry

                endif

0047 B0F3          movlw   DDRB_INIT
0048 710A          movfp   wreg,ddrb      ; set port B for whatever

0049 B801          movlb   bank1          ; bank1 initialization

                if (_SERVO_PID == TRUE)
                MOVK16   _KP,KP

004A B008          MOVLW   (1800) & 0xff
004B 0126          MOVWF   KP+B0
004C B007          MOVLW   ((1800) >> 8)
004D 0127          MOVWF   KP+B1

                                MOVK16   _KI,KI

004E B034          MOVLW   (52) & 0xff
004F 012A          MOVWF   KI+B0
0050 B000          MOVLW   ((52) >> 8)
0051 012B          MOVWF   KI+B1

                                MOVK16   _KV,KV

```

Servo Control of a DC-Brush Motor

```

0052 B0F0          MOVLW   (15600) & 0xff
0053 0128          MOVWF   KV+B0
0054 B03C          MOVLW   ((15600) >> 8)
0055 0129          MOVWF   KV+B1

                                endif

                                MOVK16  _ACCL_LIMIT,AL

0056 B000          MOVLW   (_ACCL_LIMIT) & 0xff
0057 0123          MOVWF   AL+B0
0058 B020          MOVLW   ((_ACCL_LIMIT) >> 8)
0059 0124          MOVWF   AL+B1

                                MOVK16  _VEL_LIMIT,VL

005A B000          MOVLW   (_VEL_LIMIT) & 0xff
005B 0120          MOVWF   VL+B0
005C B032          MOVLW   ((_VEL_LIMIT) >> 8)
005D 0121          MOVWF   VL+B1

005E B004          movlw   _SHIFTNUM
005F 01B9          movwf   SHIFTNUM

0060 5289          movpf   pwldch,YPWM+B1

0061 B080          movlw   PWMAXL           ; initialize pwm limits
0062 4A8E          movpf   wreg,YPWMAX+B0
0063 B0FE          movlw   PWMAXH
0064 4A8F          movpf   wreg,YPWMAX+B1
0065 B040          movlw   PWMINL
0066 4A8C          movpf   wreg,YPWMIN+B0
0067 B001          movlw   PWMINH
0068 4A8D          movpf   wreg,YPWMIN+B1

0069 2916          clrfs   pir           ; clear flags, set individual interrupts
006A 2907          clrfs   intsta
006B 8517          bsfs   _tm2ie
006C 8307          bsfs   _peie

006D 8C06          bcf     _glintd         ; enable interrupts

006E B802          movlb   bank2

                                zeroctrs
006F 290B          clrfs   rtccl           ; clear up counter
0070 290C          clrfs   rtcch

0071 2912          clrfs   tmr3l           ; clear down counter
0072 2913          clrfs   tmr3h

0073 B0FF          movlw   0xFF
0074 170A          delay   decfsz  wreg
0075 C074          goto    delay

0076 6A0B          movfp   rtccl,wreg
0077 080C          iorwf   rtcch,W
0078 0812          iorwf   tmr3l,W
0079 0813          iorwf   tmr3h,W
007A 330A          tstfsz  wreg
007B C06F          goto    zeroctrs         ; motor still moving

007C C124          goto    PollingLoop

;*****

```

Servo Control of a DC-Brush Motor

```

;*****
; NAME:      InterruptPoll

InterruptPoll

007D 0138      movwf  ISRWREG          ; save W Reg
007E 6A04      movfp  alusta,wreg
007F 01C7      movwf  alustaTemp      ; save alusta
0080 4F37      movfp  bsr,ISRBSR      ; save BSR,wreg
0081 4DC3      movfp  tblptrl,tblptrlTemp ; save Table Pointers
0082 4EC4      movfp  tblptrh,tblptrhTemp
0083 A0C5      tlrld  0,TblLatLo
0084 A2C6      tlrld  1,TblLatHi          ; save Table Latch

0085 B801      movlb  bank1

0086 E0E5      call   doMPosMVel      ; calculate measured position and
                                ; velocity

0087 E111      call   doExtstat      ; evaluate external status

0088 2293      rlncf  MOVSTAT,W          ; if MOVFLAG=0 and MOVSTAT,bit7-1
0089 B501      andlw  0x01          ; then do premove. This is only
008A 0494      subwf  MOVFLAG,W      ; executed once at the beginning of
008B 9F0A      btfsz  wreg,MSB      ; each move
008C E23D      call   doPreMove

008D 9E93      btfsz  MOVSTAT,bit6   ; is motion continuing?
008E E30F      call   doMove        ; if so, do move

008F E09E      call   doError        ; calculate position and velocity
                                ; error

0090 3390      tstfsz  SERVOFLAG     ; test servoflag, if 0 then no servo
0091 E4AC      call   doServo       ; do servo

      if      _PICMASTER_DEBUG
0092 33BB      tstfsz  CAPFLAG
0093 E79A      call   doCaptureRegs ; for PIC-MASTER Trace Capture, demo
      endif

0094 B801      movlb  bank1

0095 2916      clrf  pir            ; clear all interrupt request flags

0096 A4C5      tlwt   0,TblLatLo
0097 A6C6      tlwt   1,TblLatHi          ; restored table latch
0098 6DC3      movfp  tblptrlTemp,tblptrl
0099 6EC4      movfp  tblptrhTemp,tblptrh ; restored table pointers
009A 6F37      movfp  ISRBSR,bsr      ; restore BSR,wreg,alusta & Table
009B 64C7      movfp  alustaTemp,alusta
009C 6A38      movfp  ISRWREG,wreg

009D 0005      retfie

;*****
;*****
; NAME:      doError
;
; DESCRIPTION: Calculates the position and velocity error.
;
doError

MOV24 POSITION,POSERROR          ; calculate position error

```

Servo Control of a DC-Brush Motor

```

009E 6A55          MOVFP  POSITION+B0,wreg      ; get byte of POSITION into w
009F 4A79          MOVFP  wreg,POSERROR+B0    ; move to POSERROR(B0)
00A0 6A56          MOVFP  POSITION+B1,wreg      ; get byte of POSITION into w
00A1 4A7A          MOVFP  wreg,POSERROR+B1    ; move to POSERROR(B1)
00A2 6A57          MOVFP  POSITION+B2,wreg      ; get byte of POSITION into w
00A3 4A7B          MOVFP  wreg,POSERROR+B2    ; move to POSERROR(B2)

SUB24  MPOSITION, POSERROR

00A4 6A72          MOVFP  MPOSITION+B0,wreg    ; get lowest byte of MPOSITION into
00A5 0579          SUBWF  POSERROR+B0          ; sub lowest byte of POSERROR,
00A6 6A73          MOVFP  MPOSITION+B1,wreg    ; get 2nd byte of MPOSITION into
00A7 037A          SUBWFB POSERROR+B1          ; sub 2nd byte of POSERROR, save
00A8 6A74          MOVFP  MPOSITION+B2,wreg    ; get 3rd byte of MPOSITION into
00A9 037B          SUBWFB POSERROR+B2          ; sub 3rd byte of POSERROR, save

00AA 9F7B          btfscc POSERROR+B2,MSB      ; saturate error to lowest 16 bits
00AB C0B7          goto   pneg

ppos
00AC 6A7A          movfp  POSERROR+B1,wreg
00AD B580          andlw  0x80
00AE 097B          iorwf  POSERROR+B2
00AF 290A          clrf   wreg
00B0 327B          cpfsgt POSERROR+B2
00B1 C0C1          goto   psatok
00B2 297B          clrf   POSERROR+B2          ; clear high byte for debug purposes
00B3 B07F          movlw  0x7F
00B4 4A7A          movpf  wreg,POSERROR+B1
00B5 2B79          setf   POSERROR
00B6 C0C1          goto   psatok

pneg
00B7 6A7A          movfp  POSERROR+B1,wreg
00B8 B37F          iorlw  0x7F
00B9 0B7B          andwf  POSERROR+B2
00BA 2B0A          setf   wreg
00BB 307B          cpfslt POSERROR+B2
00BC C0C1          goto   psatok
00BD 2B7B          setf   POSERROR+B2          ; set high byte to 0xFF for debug
00BE 297A          clrf   POSERROR+B1
00BF 877A          bsf    POSERROR+B1,MSB
00C0 2979          clrf   POSERROR

psatok
MOV24  VELOCITY,VELERROR      ; calculate velocity error

00C1 6A58          MOVFP  VELOCITY+B0,wreg     ; get byte of VELOCITY into w
00C2 4A7C          MOVFP  wreg,VELERROR+B0     ; move to VELERROR(B0)
00C3 6A59          MOVFP  VELOCITY+B1,wreg     ; get byte of VELOCITY into w
00C4 4A7D          MOVFP  wreg,VELERROR+B1     ; move to VELERROR(B1)
00C5 6A5A          MOVFP  VELOCITY+B2,wreg     ; get byte of VELOCITY into w
00C6 4A7E          MOVFP  wreg,VELERROR+B2     ; move to VELERROR(B2)

SUB24  MVELOCITY,VELERROR

00C7 6A75          MOVFP  MVELOCITY+B0,wreg    ; get lowest byte of MVELOCITY into w
00C8 057C          SUBWF  VELERROR+B0          ; sub lowest byte of VELERROR, save
00C9 6A76          MOVFP  MVELOCITY+B1,wreg    ; get 2nd byte of MVELOCITY into w
00CA 037D          SUBWFB VELERROR+B1          ; sub 2nd byte of VELERROR, save in
00CB 6A77          MOVFP  MVELOCITY+B2,wreg    ; get 3rd byte of MVELOCITY into w
00CC 037E          SUBWFB VELERROR+B2          ; sub 3rd byte of VELERROR, save in

00CD 9F7E          btfscc VELERROR+B2,MSB     ; saturate error to lowest 16 bits
00CE C0DA          goto   vneg

vpos

```

Servo Control of a DC-Brush Motor

```

00CF 6A7D          movfp  VEERROR+B1,wreg
00D0 B580          andlw  0x80
00D1 097E          iorwf  VEERROR+B2
00D2 290A          clrf   wreg
00D3 327E          cpfsgt VEERROR+B2
00D4 C0E4          goto   vsatok
00D5 297E          clrf   VEERROR+B2
00D6 B07F          movlw  0x7F
00D7 4A7D          movfp  wreg,VEERROR+B1
00D8 2B7C          setf   VEERROR
00D9 C0E4          goto   vsatok

vneg
00DA 6A7D          movfp  VEERROR+B1,wreg
00DB B37F          iorlw  0x7F
00DC 0B7E          andwf  VEERROR+B2
00DD 2B0A          setf   wreg
00DE 307E          cpfslt VEERROR+B2
00DF C0E4          goto   vsatok
00E0 2B7E          setf   VEERROR+B2
00E1 297D          clrf   VEERROR+B1
00E2 877D          bsf   VEERROR+B1,MSB
00E3 297C          clrf   VEERROR

vsatok
00E4 0002          return

;*****
;*****
; NAME:          doMPosMVel
;
; DESCRIPTION:   Calculates current position from UpCount and DownCount
;
doMPosMVel

; Do UpCounter first

MOVFP16  UPCOUNT,TMP+B0 ; save old upcount

00E5 78AC          MOVFP  UPCOUNT+B0,TMP+B0+B0 ; move UPCOUNT(B0) to TMP+B0(B0)
00E6 79AD          MOVFP  UPCOUNT+B1,TMP+B0+B1 ; move UPCOUNT(B1) to TMP+B0(B1)

readUp
00E7 4C0A          movfp  rtcch,wreg
00E8 4BAC          movfp  rtccl,UPCOUNT+B0
00E9 310C          cpfseq rtcch ; Skip next if HI hasn't changed
00EA C0E7          goto   readUp ; HI changed, re-read LO
00EB 4AAD          movfp  wreg,UPCOUNT+B1 ; OK to store HI now

00EC 2975          clrf   MVELOCITY+B0 ; clear bits below binary point

MOV16  UPCOUNT,MVELOCITY+B1 ; compute upcount increment

00ED 6AAC          MOVFP  UPCOUNT+B0,wreg ; get byte of UPCOUNT into w
00EE 0176          MOVWF  MVELOCITY+B1+B0 ; move to MVELOCITY+B1(B0)
00EF 6AAD          MOVFP  UPCOUNT+B1,wreg ; get byte of UPCOUNT into w
00F0 0177          MOVWF  MVELOCITY+B1+B1 ; move to MVELOCITY+B1(B1)

SUB16  TMP+B0,MVELOCITY+B1

00F1 6A18          MOVFP  TMP+B0+B0,wreg ; get lowest byte of TMP+B0 into
00F2 0576          SUBWF  MVELOCITY+B1+B0 ; sub lowest byte of
00F3 6A19          MOVFP  TMP+B0+B1,wreg ; get 2nd byte of TMP+B0 into w
00F4 0377          SUBWFB MVELOCITY+B1+B1 ; sub 2nd byte of MVELOCITY+B1,

```

Servo Control of a DC-Brush Motor

```

; Now do DownCounter

MOVFP16  DOWNCOUNT,TMP+B0                ; save old downcount

00F5 78AE          MOVFP  DOWNCOUNT+B0,TMP+B0+B0    ; move DOWNCOUNT(B0) to
00F6 79AF          MOVFP  DOWNCOUNT+B1,TMP+B0+B1    ; move DOWNCOUNT(B1) to

readDown
00F7 B802          movlb  bank2      ; timers in Bank 2
00F8 530A          movpf  tmr3h,wreg
00F9 52AE          movpf  tmr3l,DOWNCOUNT+B0
00FA 3113          cpfseq tmr3h                ; Skip next if HI hasn't changed
00FB C0F7          goto   readDown           ; HI changed, re-read LO
00FC 4AAF          movpf  wreg,DOWNCOUNT+B1      ; OK to store HI now

MOVFP16  DOWNCOUNT+B0,TMP+B2            ; compute downcount increment

00FD 7AAE          MOVFP  DOWNCOUNT+B0+B0,TMP+B2+B0  ; move DOWNCOUNT+B0(B0) to
00FE 7BAF          MOVFP  DOWNCOUNT+B0+B1,TMP+B2+B1  ; move DOWNCOUNT+B0(B1) to

SUB16    TMP+B0,TMP+B2

00FF 6A18          MOVFP  TMP+B0+B0,wreg          ; get lowest byte of TMP+B0 into
0100 051A          SUBWF  TMP+B2+B0                ; sub lowest byte of TMP+B2, save
0101 6A19          MOVFP  TMP+B0+B1,wreg          ; get 2nd byte of TMP+B0 into w
0102 031B          SUBWFB TMP+B2+B1                ; sub 2nd byte of TMP+B2, save in

SUB16    TMP+B2,MVELOCITY+B1            ; compute new measured velocity

0103 6A1A          MOVFP  TMP+B2+B0,wreg          ; get lowest byte of TMP+B2 into
0104 0576          SUBWF  MVELOCITY+B1+B0          ; sub lowest byte of
0105 6A1B          MOVFP  TMP+B2+B1,wreg          ; get 2nd byte of TMP+B2 into w
0106 0377          SUBWFB MVELOCITY+B1+B1          ; sub 2nd byte of MVELOCITY+B1,

0107 2978          clrf   MVELOCITY+B3            ; sign extend measured velocity
0108 9F77          btfs  MVELOCITY+B2,MSB          ; 24 bit addition to measured
0109 2B78          setf   MVELOCITY+B3

ADD24    MVELOCITY+B1,MPOSITION        ; compute new measured position

010A 6A76          MOVFP  MVELOCITY+B1+B0,wreg          ; get lowest byte of MVELOCITY+B1
010B 0F72          ADDWF  MPOSITION+B0                ; add lowest byte of MPOSITION,
010C 6A77          MOVFP  MVELOCITY+B1+B1,wreg          ; get 2nd byte of MVELOCITY+B1
010D 1173          ADDWFC MPOSITION+B1                ; add 2nd byte of MPOSITION, save
010E 6A78          MOVFP  MVELOCITY+B1+B2,wreg          ; get 3rd byte of MVELOCITY+B1
010F 1174          ADDWFC MPOSITION+B2                ; add 3rd byte of MPOSITION, save

; delta position = measured

0110 0002          return

;*****
; NAME:           doExtstat
;
; DESCRIPTION:    Get +limit,-limit,GPI from PORTB and set in EXTSTAT
;
doExtstat
0111 9407          btfs  _intir
0112 C11B          goto  otherbits

MOV24    MPOSITION,INDEXPOS

```

Servo Control of a DC-Brush Motor

```

0113 6A72          MOVFP  MPOSITION+B0,wreg          ; get byte of MPOSITION into w
0114 4AB6          MOVFP  wreg,INDEXPOS+B0          ; move to INDEXPOS (B0)
0115 6A73          MOVFP  MPOSITION+B1,wreg          ; get byte of MPOSITION into w
0116 4AB7          MOVFP  wreg,INDEXPOS+B1          ; move to INDEXPOS (B1)
0117 6A74          MOVFP  MPOSITION+B2,wreg          ; get byte of MPOSITION into w
0118 4AB8          MOVFP  wreg,INDEXPOS+B2          ; move to INDEXPOS (B2)

0119 8C07          bcf     _intir
011A 8792          bsf     EXTSTAT,MSB

otherbits
011B B800          movlb  bank0                ; get +limit,-limit and GPI
011C 6A12          movfp  portb,wreg
011D 190A          rrcf   wreg                ; arrange in correct bit posi
011E B561          andlw  0x61
011F 4A18          movpf  wreg,TMP
0120 1D18          swapf  TMP
0121 0818          iorwf  TMP,W
0122 0992          iorwf  EXTSTAT                ; set in EXTSTAT

0123 0002          return

;*****
;*****
; NAME:           PollingLoop
;
; DESCRIPTION:    The actual polling loop called after the board's
;                 initialization
;
; ENTRY CONDITIONS: System globals and hardware initialized and the
;                 interrupt processes started.
;
;
PollingLoop
    if _SERIAL_IO
0124 E557          call   IdleFunction
0125 E681          call   GetChk
0126 31C2          cpfseq ONE                ; GetChk, is receive buffer full?
0127 C124          goto   PollingLoop

0128 E676          call   GetChar                ; if so, get character
0129 4A39          movpf  wreg,CMDCHAR          ; put in CMDCHAR
012A C559          goto   DoCommand

    else
        clrwdt
        goto   PollingLoop                ; wait for Interrupt
    endif

;*****
;*****
include "mult.asm" ; Double Precision
Multiplication Routine
;*****
;
;                 Double Precision Multiplication Routine
;
; NAME:           Dmult
;
; DESCRIPTION:    Mult: AARG (16 bits) * BARG (16 bits) -> DPX (32 bits)
;
; (a) Load the 1st operand in locations AARG+B0 & AARG+B1 (16 bits)
; (b) Load the 2nd operand in locations BARG+B0 & BARG+B1 (16 bits)
; (c) call Dmult
; (d) The 32 bit result is in locations (DPX+B0,DPX+B1,DPX+B2,DPX+B3)
;
; In the signed case, a savings of 9 clks can be realized by choosing

```

Servo Control of a DC-Brush Motor

```

;          BARG as the positive factor in the product when possible.
;
; TIMING (worst case):  unsigned:          173 clks
;                       signed:   if BARG positive: 170 clks
;                               if BARG negative: 179 clks
;
;*****
0001 SIGNED equ      TRUE          ; Set This To 'TRUE' for signed multiply
;                               ; and 'FALSE' for unsigned.
;*****
;          Multiplication Macro
;*****
; TIMING:      unsigned:    11+7*10+8*11 = 169 clks
; (worst case) signed:     11+7*10+7*11+5 = 163 clks
;
MULTMAC MACRO
variable i

        i = 0
        if SIGNED
            while i < 15
        else
            while i < 16
        endif
            if i < 8
                btfsc    BARG+B0,i          ; test low byte
            else
                btfsc    BARG+B1,i-8        ; test high byte
            endif
            goto    add#v(i)
            if i < 8
                rrcf    DPX+B3,W           ; rotate sign into carry bit
                rrcf    DPX+B3            ; for i < 8, no meaningful
                rrcf    DPX+B2            ; are in DPX+B0
                rrcf    DPX+B1
            else
                rrcf    DPX+B3,W           ; rotate sign into carry bit
                rrcf    DPX+B3
                rrcf    DPX+B2
                rrcf    DPX+B1
                rrcf    DPX+B0
            endif
            i = i+1
        endw

        clrfs    DPX+B0                    ; if we get here, BARG = 0
        return

add0
        movfpc    AARG+B0,wreg
        addwf    DPX+B2                    ;add lsb
        movfpc    AARG+B1,wreg
        addwfc    DPX+B3                    ;add msb
        rrcf    AARG+B1,W                 ; rotate sign into carry bit
        rrcf    DPX+B3                    ; for i < 8, no meaningful
        rrcf    DPX+B2                    ; are in DPX+B0
        rrcf    DPX+B1

        i = 1

        if SIGNED

            while i < 15

```


Servo Control of a DC-Brush Motor

```

else
  while i < 16
endif
  if i < 8
    btfss    BARG+B0,i          ; test low byte
  else
    btfss    BARG+B1,i-8      ; test high byte
  endif
  goto      noadd#v(i)
  add#v(i)
  movfp     AARG+B0,wreg
  addwf     DPX+B2            ;add lsb
  movfp     AARG+B1,wreg
  addwfc    DPX+B3            ;add msb

noadd#v(i)
  if i < 8

    rlcfc   AARG+B1,W          ; rotate sign into carry bit
    rrcfc   DPX+B3            ; for i < 8, no meaningful
    rrcfc   DPX+B2            ; are in DPX+B0
    rrcfc   DPX+B1

  else

    rlcfc   AARG+B1,W          ; rotate sign into carry bit
    rrcfc   DPX+B3
    rrcfc   DPX+B2
    rrcfc   DPX+B1
    rrcfc   DPX+B0

  endif

  i = i+1
endw

if    SIGNED

  rlcfc   AARG+B1,W          ; since BARG is always made
  rrcfc   DPX+B3            ; the last bit is known to be
  rrcfc   DPX+B2
  rrcfc   DPX+B1
  rrcfc   DPX+B0

endif

ENDM

; Double Precision Multiply (
; ( AARG*BARG -> : 32 bit

;
; Dmult
if    SIGNED

012B 971F      btfss    BARG+B1,MSB          ; test sign of BARG
012C C137      goto      argsok          ; if positive, ok
                                NEG16   AARG+B0      ; if negative, then negate

012D 131C      COMF     AARG+B0+B0
012E 131D      COMF     AARG+B0+B1
012F 290A      CLRF    wreg
0130 151C      INCF    AARG+B0+B0
0131 111D      ADDWFC   AARG+B0+B1

                                NEG16   BARG+B0      ; AARG and BARG

0132 131E      COMF     BARG+B0+B0
0133 131F      COMF     BARG+B0+B1

```

Servo Control of a DC-Brush Motor

```

0134 290A          CLRF    wreg
0135 151E          INCF    BARG+B0+B0
0136 111F          ADDWFC  BARG+B0+B1

                                endif

                                argsok
0137 291B          clrf    DPX+B3          ; clear initial partial product
0138 291A          clrf    DPX+B2

MULTMAC                                ; use macro for multiplication
0000          variable i
0000          i = 0
                                if SIGNED
                                while i < 15
                                else
                                while i < 16
                                endif

                                if i < 8
                                btfscc    BARG+B0,i          ; test low byte
                                else
                                btfscc    BARG+B1,i-8          ; test high byte
                                endif
                                goto    add#v(i)
                                if i < 8
                                rlcfc    DPX+B3,W          ; rotate sign into carry bit
                                rrcfc    DPX+B3          ; for i < 8, no meaningful
bits
                                rrcfc    DPX+B2          ; are in DPX+B0
                                rrcfc    DPX+B1
                                else
                                rlcfc    DPX+B3,W          ; rotate sign into carry bit
                                rrcfc    DPX+B3
                                rrcfc    DPX+B2
                                rrcfc    DPX+B1
                                rrcfc    DPX+B0
                                endif
                                i = i+1
                                endw

                                if i < 8
0139 981E          btfscc    BARG+B0,i          ; test low byte
                                else
                                btfscc    BARG+B1,i-8          ; test high byte
                                endif

                                goto    add0
                                if i < 8
013B 1A1B          rlcfc    DPX+B3,W          ; rotate sign into carry bit
013C 191B          rrcfc    DPX+B3          ; for i < 8, no meaningful
013D 191A          rrcfc    DPX+B2          ; are in DPX+B0
013E 1919          rrcfc    DPX+B1
                                else
                                rlcfc    DPX+B3,W          ; rotate sign into carry bit

                                rrcfc    DPX+B3

                                rrcfc    DPX+B2

                                rrcfc    DPX+B1

                                rrcfc    DPX+B0

```

Servo Control of a DC-Brush Motor

```
endif
0001          i = i+1
if i < 8
013F 991E    btfsc    BARG+B0,i          ; test low byte
            else
            btfsc    BARG+B1,i-8        ; test high byte
endif
0140 C1A6    goto    add1
if i < 8
0141 1A1B    rlcfc   DPX+B3,W          ; rotate sign into carry bit
0142 191B    rrcfc   DPX+B3            ; for i < 8, no meaningful
0143 191A    rrcfc   DPX+B2            ; are in DPX+B0
0144 1919    rrcfc   DPX+B1
else
            rlcfc   DPX+B3,W          ; rotate sign into carry bit
            rrcfc   DPX+B3
            rrcfc   DPX+B2
            rrcfc   DPX+B1
            rrcfc   DPX+B0
endif
0002          i = i+1
if i < 8
0145 9A1E    btfsc    BARG+B0,i          ; test low byte
            else
            btfsc    BARG+B1,i-8        ; test high byte
endif
0146 C1B0    goto    add2
if i < 8
0147 1A1B    rlcfc   DPX+B3,W          ; rotate sign into carry bit
0148 191B    rrcfc   DPX+B3            ; for i < 8, no meaningful
0149 191A    rrcfc   DPX+B2            ; are in DPX+B0
014A 1919    rrcfc   DPX+B1
else
            rlcfc   DPX+B3,W          ; rotate sign into carry bit
            rrcfc   DPX+B3
            rrcfc   DPX+B2
            rrcfc   DPX+B1
            rrcfc   DPX+B0
endif
0003          i = i+1
if i < 8
014B 9B1E    btfsc    BARG+B0,i          ; test low byte
            else
            btfsc    BARG+B1,i-8        ; test high byte
endif
014C C1BA    goto    add3
if i < 8
```

Servo Control of a DC-Brush Motor

```
014D 1A1B          rrcf   DPX+B3,W           ; rotate sign into carry bit
014E 191B          rrcf   DPX+B3           ; for i < 8, no meaningful
014F 191A          rrcf   DPX+B2           ; are in DPX+B0
0150 1919          rrcf   DPX+B1
                    else
                    rrcf   DPX+B3,W         ; rotate sign into carry bit
                    rrcf   DPX+B3
                    rrcf   DPX+B2
                    rrcf   DPX+B1
                    rrcf   DPX+B0
                    endif
0004              i = i+1
                    if i < 8
0151 9C1E          btfsc  BARG+B0,i           ; test low byte
                    else
                    btfsc  BARG+B1,i-8       ; test high byte
                    endif
0152 C1C4          goto   add4
                    if i < 8
0153 1A1B          rrcf   DPX+B3,W           ; rotate sign into carry bit
0154 191B          rrcf   DPX+B3           ; for i < 8, no meaningful
0155 191A          rrcf   DPX+B2           ; are in DPX+B0
0156 1919          rrcf   DPX+B1
                    else
                    rrcf   DPX+B3,W         ; rotate sign into carry bit
                    rrcf   DPX+B3
                    rrcf   DPX+B2
                    rrcf   DPX+B1
                    rrcf   DPX+B0
                    endif
0005              i = i+1
                    if i < 8
0157 9D1E          btfsc  BARG+B0,i           ; test low byte
                    else
                    btfsc  BARG+B1,i-8       ; test high byte
                    endif
0158 C1CE          goto   add5
                    if i < 8
0159 1A1B          rrcf   DPX+B3,W           ; rotate sign into carry bit
015A 191B          rrcf   DPX+B3           ; for i < 8, no meaningful
015B 191A          rrcf   DPX+B2           ; are in DPX+B0
015C 1919          rrcf   DPX+B1
                    else
                    rrcf   DPX+B3,W         ; rotate sign into carry bit
                    rrcf   DPX+B3
                    rrcf   DPX+B2
                    rrcf   DPX+B1
```

Servo Control of a DC-Brush Motor

```

                                rrcf    DFX+B0
endif
0006                            i = i+1
                                if i < 8
015D 9E1E                        btfscc BARG+B0,i          ; test low byte
                                else
                                btfscc BARG+B1,i-8        ; test high byte
                                endif
015E C1D8                        goto    add6
                                if i < 8
015F 1A1B                        rlcfc  DFX+B3,W          ; rotate sign into carry bit
0160 191B                        rrcfc  DFX+B3          ; for i < 8, no meaningful
0161 191A                        rrcfc  DFX+B2          ; are in DFX+B0
0162 1919                        rrcfc  DFX+B1
                                else
                                rlcfc  DFX+B3,W          ; rotate sign into carry bit
                                rrcfc  DFX+B3
                                rrcfc  DFX+B2
                                rrcfc  DFX+B1
                                rrcfc  DFX+B0
                                endif
0007                            i = i+1
                                if i < 8
0163 9F1E                        btfscc BARG+B0,i          ; test low byte
                                else
                                btfscc BARG+B1,i-8        ; test high byte
                                endif
0164 C1E2                        goto    add7
                                if i < 8
0165 1A1B                        rlcfc  DFX+B3,W          ; rotate sign into carry bit
0166 191B                        rrcfc  DFX+B3          ; for i < 8, no meaningful
0167 191A                        rrcfc  DFX+B2          ; are in DFX+B0
0168 1919                        rrcfc  DFX+B1
                                else
                                rlcfc  DFX+B3,W          ; rotate sign into carry bit
                                rrcfc  DFX+B3
                                rrcfc  DFX+B2
                                rrcfc  DFX+B1
                                rrcfc  DFX+B0
                                endif
0008                            i = i+1
                                if i < 8
                                btfscc BARG+B0,i          ; test low byte
                                else
0169 981F                        btfscc BARG+B1,i-8        ; test high byte
                                endif
```



Servo Control of a DC-Brush Motor

```
016A C1EC          goto    add8
                   if i < 8
                   rlcfc  DPX+B3,W          ; rotate sign into carry bit

                   rrcfc  DPX+B3          ; for i < 8, no meaningful

                   rrcfc  DPX+B2          ; are in DPX+B0

                   rrcfc  DPX+B1

                   else

016B 1A1B          rlcfc  DPX+B3,W          ; rotate sign into carry bit
016C 191B          rrcfc  DPX+B3
016D 191A          rrcfc  DPX+B2
016E 1919          rrcfc  DPX+B1
016F 1918          rrcfc  DPX+B0
                   endif
0009              i = i+1

                   if i < 8
                   btfscc BARG+B0,i        ; test low byte

                   else

0170 991F          btfscc BARG+B1,i-8      ; test high byte
                   endif
0171 C1F7          goto    add9
                   if i < 8
                   rlcfc  DPX+B3,W          ; rotate sign into carry bit

                   rrcfc  DPX+B3          ; for i < 8, no meaningful

                   rrcfc  DPX+B2          ; are in DPX+B0

                   rrcfc  DPX+B1

                   else

0172 1A1B          rlcfc  DPX+B3,W          ; rotate sign into carry bit
0173 191B          rrcfc  DPX+B3
0174 191A          rrcfc  DPX+B2
0175 1919          rrcfc  DPX+B1
0176 1918          rrcfc  DPX+B0
                   endif
000A              i = i+1

                   if i < 8
                   btfscc BARG+B0,i        ; test low byte

                   else

0177 9A1F          btfscc BARG+B1,i-8      ; test high byte
                   endif
0178 C202          goto    add10
                   if i < 8
                   rlcfc  DPX+B3,W          ; rotate sign into carry bit

                   rrcfc  DPX+B3          ; for i < 8, no meaningful

                   rrcfc  DPX+B2          ; are in DPX+B0

                   rrcfc  DPX+B1

                   else

0179 1A1B          rlcfc  DPX+B3,W          ; rotate sign into carry bit
017A 191B          rrcfc  DPX+B3
017B 191A          rrcfc  DPX+B2
```

Servo Control of a DC-Brush Motor

```
017C 1919          rrcf    DPX+B1
017D 1918          rrcf    DPX+B0
                    endif
000B              i = i+1

                    if i < 8
                        btfscc BARG+B0,i          ; test low byte
                    else
017E 9B1F          btfscc BARG+B1,i-8          ; test high byte
                    endif
017F C20D          goto    add11
                    if i < 8
                        rlcfc DPX+B3,W            ; rotate sign into carry bit

                        rrcf    DPX+B3            ; for i < 8, no meaningful
                        rrcf    DPX+B2            ; are in DPX+B0

                        rrcf    DPX+B1

                    else

0180 1A1B          rlcfc DPX+B3,W            ; rotate sign into carry bit
0181 191B          rrcf    DPX+B3
0182 191A          rrcf    DPX+B2
0183 1919          rrcf    DPX+B1
0184 1918          rrcf    DPX+B0
                    endif
000C              i = i+1

                    if i < 8
                        btfscc BARG+B0,i          ; test low byte
                    else

0185 9C1F          btfscc BARG+B1,i-8          ; test high byte
                    endif
0186 C218          goto    add12
                    if i < 8
                        rlcfc DPX+B3,W            ; rotate sign into carry bit

                        rrcf    DPX+B3            ; for i < 8, no meaningful
                        rrcf    DPX+B2            ; are in DPX+B0

                        rrcf    DPX+B1

                    else

0187 1A1B          rlcfc DPX+B3,W            ; rotate sign into carry bit
0188 191B          rrcf    DPX+B3
0189 191A          rrcf    DPX+B2
018A 1919          rrcf    DPX+B1
018B 1918          rrcf    DPX+B0
                    endif
000D              i = i+1

                    if i < 8
                        btfscc BARG+B0,i          ; test low byte
                    else

018C 9D1F          btfscc BARG+B1,i-8          ; test high byte
                    endif
018D C223          goto    add13
                    if i < 8
                        rlcfc DPX+B3,W            ; rotate sign into carry bit
```



Servo Control of a DC-Brush Motor

```

                                rrcf    DPX+B3                ; for i < 8, no meaningful
                                rrcf    DPX+B2                ; are in DPX+B0
                                rrcf    DPX+B1
                                else
018E 1A1B                        rlcfc   DPX+B3,W           ; rotate sign into carry bit
018F 191B                        rrcf   DPX+B3
0190 191A                        rrcf   DPX+B2
0191 1919                        rrcf   DPX+B1
0192 1918                        rrcf   DPX+B0
                                endif
000E                               i = i+1
                                if i < 8
                                btfscc  BARG+B0,i          ; test low byte
                                else
0193 9E1F                        btfscc  BARG+B1,i-8      ; test high byte
                                endif
0194 C22E                        goto    add14
                                if i < 8
                                rlcfc   DPX+B3,W           ; rotate sign into carry bit
                                rrcf    DPX+B3                ; for i < 8, no meaningful
                                rrcf    DPX+B2                ; are in DPX+B0
                                rrcf    DPX+B1
                                else
0195 1A1B                        rlcfc   DPX+B3,W           ; rotate sign into carry bit
0196 191B                        rrcf   DPX+B3
0197 191A                        rrcf   DPX+B2
0198 1919                        rrcf   DPX+B1
0199 1918                        rrcf   DPX+B0
                                endif
000F                               i = i+1
                                clrf    DPX+B0                ; if we get here, BARG = 0
019A 2918                        return
019B 0002
                                add0
019C 6A1C                        movfpc  AARG+B0,wreg
019D 0F1A                        addwfc DPX+B2                ;add lsb
019E 6A1D                        movfpc  AARG+B1,wreg
019F 111B                        addwfc  DPX+B3 ;add msb
01A0 1A1D                        rlcfc   AARG+B1,W         ; rotate sign into carry bit
01A1 191B                        rrcf   DPX+B3                ; for i < 8, no meaningful
01A2 191A                        rrcf   DPX+B2                ; are in DPX+B0
01A3 1919                        rrcf   DPX+B1
0001                               i = 1
                                if    SIGNED
                                while i < 15
                                else
                                while i < 16
                                endif
                                endif
```


Servo Control of a DC-Brush Motor

```

                                if i < 8
                                btfss      BARG+B0,i   ; test low byte
                                else
                                btfss      BARG+B1,i-8 ; test high byte
                                endif
                                goto       noadd#v(i)

add#v(i)
                                movfp     AARG+B0,wreg
                                addwf     DPX+B2 ;add lsb
                                movfp     AARG+B1,wreg
                                addwfc    DPX+B3 ;add msb

noadd#v(i)
                                if i < 8

                                rlcfc     AARG+B1,W     ; rotate sign into carry bit
                                rrcfc     DPX+B3         ; for i < 8, no meaningful bits
                                rrcfc     DPX+B2         ; are in DPX+B0
                                rrcfc     DPX+B1

                                else

                                rlcfc     AARG+B1,W     ; rotate sign into carry bit
                                rrcfc     DPX+B3
                                rrcfc     DPX+B2
                                rrcfc     DPX+B1
                                rrcfc     DPX+B0

                                endif

                                i = i+1
                                endw

                                if i < 8
01A4 911E      btfss      BARG+B0,i   ; test low byte
                                else
                                btfss      BARG+B1,i-8 ; test high byte
                                endif

                                goto       noadd1

01A5 C1AA      addl
                                movfp     AARG+B0,wreg
01A6 6A1C      addwf     DPX+B2 ;add lsb
01A7 0F1A      movfp     AARG+B1,wreg
01A8 6A1D      addwfc    DPX+B3 ;add msb
01A9 111B

noadd1
                                if i < 8

01AA 1A1D      rlcfc     AARG+B1,W     ; rotate sign into carry bit
01AB 191B      rrcfc     DPX+B3         ; for i < 8, no meaningful bits
01AC 191A      rrcfc     DPX+B2         ; are in DPX+B0
01AD 1919      rrcfc     DPX+B1

                                else

                                rlcfc     AARG+B1,W     ; rotate sign into carry bit

                                rrcfc     DPX+B3

                                rrcfc     DPX+B2

                                rrcfc     DPX+B1

                                rrcfc     DPX+B0

```

Servo Control of a DC-Brush Motor

```

                                endif

0002                                i = i+1

                                if i < 8
01AE 921E                btfss    BARG+B0,i        ; test low byte
                                else
                                btfss    BARG+B1,i-8    ; test high byte
                                endif

01AF C1B4                                goto    noadd2

                                add2
01B0 6A1C                movfp    AARG+B0,wreg
01B1 0F1A                addwf   DPX+B2                ;add lsb
01B2 6A1D                movfp    AARG+B1,wreg
01B3 111B                addwfc  DPX+B3                ;add msb

                                noadd2
                                if i < 8
01B4 1A1D                rlcfc   AARG+B1,W        ; rotate sign into carry bit
01B5 191B                rrcfc   DPX+B3                ; for i < 8, no meaningful
01B6 191A                rrcfc   DPX+B2                ; are in DPX+B0
01B7 1919                rrcfc   DPX+B1

                                else

                                rlcfc   AARG+B1,W        ; rotate sign into carry bit

                                rrcfc   DPX+B3

                                rrcfc   DPX+B2

                                rrcfc   DPX+B1

                                rrcfc   DPX+B0

                                endif

0003                                i = i+1

                                if i < 8
01B8 931E                btfss    BARG+B0,i        ; test low byte
                                else
                                btfss    BARG+B1,i-8    ; test high byte
                                endif

01B9 C1BE                                goto    noadd3

                                add3
01BA 6A1C                movfp    AARG+B0,wreg
01BB 0F1A                addwf   DPX+B2                ;add lsb
01BC 6A1D                movfp    AARG+B1,wreg
01BD 111B                addwfc  DPX+B3                ;add msb

                                noadd3
                                if i < 8
01BE 1A1D                rlcfc   AARG+B1,W        ; rotate sign into carry bit
01BF 191B                rrcfc   DPX+B3                ; for i < 8, no meaningful
01C0 191A                rrcfc   DPX+B2                ; are in DPX+B0
```

Servo Control of a DC-Brush Motor

```
01C1 1919          rrcf    DPX+B1
                   else
                   rlcfc   AARG+B1,W          ; rotate sign into carry bit
                   rrcf    DPX+B3
                   rrcf    DPX+B2
                   rrcf    DPX+B1
                   rrcf    DPX+B0
                   endif

0004              i = i+1
01C2 941E          if i < 8
                   btfs    BARG+B0,i          ; test low byte
                   else
                   btfs    BARG+B1,i-8        ; test high byte
                   endif

01C3 C1C8          goto    noadd4
                   add4
01C4 6A1C          movfp   AARG+B0,wreg
01C5 0F1A          addwf   DPX+B2              ;add lsb
01C6 6A1D          movfp   AARG+B1,wreg
01C7 111B          addwfc  DPX+B3              ;add msb
                   noadd4
                   if i < 8
01C8 1A1D          rlcfc   AARG+B1,W          ; rotate sign into carry bit
01C9 191B          rrcf    DPX+B3              ; for i < 8, no meaningful
01CA 191A          rrcf    DPX+B2              ; are in DPX+B0
01CB 1919          rrcf    DPX+B1
                   else
                   rlcfc   AARG+B1,W          ; rotate sign into carry bit
                   rrcf    DPX+B3
                   rrcf    DPX+B2
                   rrcf    DPX+B1
                   rrcf    DPX+B0
                   endif

0005              i = i+1
01CC 951E          if i < 8
                   btfs    BARG+B0,i          ; test low byte
                   else
                   btfs    BARG+B1,i-8        ; test high byte
                   endif
```



Servo Control of a DC-Brush Motor

```
01CD C1D2                goto    noadd5
                        add5
01CE 6A1C                movfp   AARG+B0,wreg
01CF 0F1A                addwf  DPX+B2 ;add lsb
01D0 6A1D                movfp   AARG+B1,wreg
01D1 111B                addwfc DPX+B3 ;add msb

                        noadd5
                        if i < 8

01D2 1A1D                rlcfc  AARG+B1,W          ; rotate sign into carry bit
01D3 191B                rrcfc  DPX+B3            ; for i < 8, no meaningful
01D4 191A                rrcfc  DPX+B2            ; are in DPX+B0
01D5 1919                rrcfc  DPX+B1

                        else

                        rlcfc  AARG+B1,W          ; rotate sign into carry bit

                        rrcfc  DPX+B3

                        rrcfc  DPX+B2

                        rrcfc  DPX+B1

                        rrcfc  DPX+B0

                        endif

0006                    i = i+1

                        if i < 8
01D6 961E                btfss  BARG+B0,i          ; test low byte
                        else
                        btfss  BARG+B1,i-8        ; test high byte
                        endif

01D7 C1DC                goto    noadd6
                        add6
01D8 6A1C                movfp   AARG+B0,wreg
01D9 0F1A                addwf  DPX+B2 ;add lsb
01DA 6A1D                movfp   AARG+B1,wreg
01DB 111B                addwfc DPX+B3 ;add msb

                        noadd6
                        if i < 8

01DC 1A1D                rlcfc  AARG+B1,W          ; rotate sign into carry bit
01DD 191B                rrcfc  DPX+B3            ; for i < 8, no meaningful
01DE 191A                rrcfc  DPX+B2            ; are in DPX+B0
01DF 1919                rrcfc  DPX+B1

                        else

                        rlcfc  AARG+B1,W          ; rotate sign into carry bit

                        rrcfc  DPX+B3

                        rrcfc  DPX+B2

                        rrcfc  DPX+B1
```

Servo Control of a DC-Brush Motor

```

                                rrcf    DPX+B0

                                endif

0007                            i = i+1

                                if i < 8
01E0 971E                        btfss   BARG+B0,i          ; test low byte
                                else
                                btfss   BARG+B1,i-8        ; test high byte
                                endif

01E1 C1E6                        goto    noadd7
                                add7
01E2 6A1C                        movfp  AARG+B0,wreg
01E3 0F1A                        addwf  DPX+B2 ;add lsb
01E4 6A1D                        movfp  AARG+B1,wreg
01E5 111B                        addwfc DPX+B3 ;add msb

                                noadd7

                                if i < 8
01E6 1A1D                        rlcfc  AARG+B1,W          ; rotate sign into carry bit
01E7 191B                        rrcf   DPX+B3            ; for i < 8, no meaningful
01E8 191A                        rrcf   DPX+B2            ; are in DPX+B0
01E9 1919                        rrcf   DPX+B1

                                else

                                rlcfc  AARG+B1,W          ; rotate sign into carry bit

                                rrcf   DPX+B3
                                rrcf   DPX+B2
                                rrcf   DPX+B1
                                rrcf   DPX+B0

                                endif

0008                            i = i+1

                                if i < 8
01EA 901F                        btfss   BARG+B0,i          ; test low byte
                                else
                                btfss   BARG+B1,i-8        ; test high byte
                                endif
01EB C1F0                        goto    noadd8
                                add8
01EC 6A1C                        movfp  AARG+B0,wreg
01ED 0F1A                        addwf  DPX+B2 ;add lsb
01EE 6A1D                        movfp  AARG+B1,wreg
01EF 111B                        addwfc DPX+B3 ;add msb

                                noadd8

                                if i < 8
```

Servo Control of a DC-Brush Motor

```

        rlcfc      AARG+B1,W          ; rotate sign into carry bit
        rrcfc      DPX+B3            ; for i < 8, no meaningful
        rrcfc      DPX+B2            ; are in DPX+B0
        rrcfc      DPX+B1

    else

01F0 1A1D          rlcfc      AARG+B1,W          ; rotate sign into carry bit
01F1 191B          rrcfc      DPX+B3
01F2 191A          rrcfc      DPX+B2
01F3 1919          rrcfc      DPX+B1
01F4 1918          rrcfc      DPX+B0

    endif

0009              i = i+1

        if i < 8
            btffss     BARG+B0,i          ; test low byte
        else
01F5 911F          btffss     BARG+B1,i-8      ; test high byte
        endif
01F6 C1FB          goto      noadd9
    add9
01F7 6A1C          movfp      AARG+B0,wreg
01F8 0F1A          addwfc     DPX+B2 ;add lsb
01F9 6A1D          movfp      AARG+B1,wreg
01FA 111B          addwfc     DPX+B3 ;add msb

    noadd9
        if i < 8

            rlcfc      AARG+B1,W          ; rotate sign into carry bit
            rrcfc      DPX+B3            ; for i < 8, no meaningful
            rrcfc      DPX+B2            ; are in DPX+B0
            rrcfc      DPX+B1

        else

01FB 1A1D          rlcfc      AARG+B1,W          ; rotate sign into carry bit
01FC 191B          rrcfc      DPX+B3
01FD 191A          rrcfc      DPX+B2
01FE 1919          rrcfc      DPX+B1
01FF 1918          rrcfc      DPX+B0

        endif

000A              i = i+1

        if i < 8
            btffss     BARG+B0,i          ; test low byte
        else
```

Servo Control of a DC-Brush Motor

```
0200 921F          btfs    BARG+B1,i-8      ; test high byte
                    endif
0201 C206          goto    noadd10
                    add10
0202 6A1C          movfp   AARG+B0,wreg
0203 0F1A          addwf   DPX+B2 ;add lsb
0204 6A1D          movfp   AARG+B1,wreg
0205 111B          addwfc  DPX+B3 ;add msb

                    noadd10
                    if i < 8

                                rlcfc   AARG+B1,W          ; rotate sign into carry bit
                                rrcfc   DPX+B3              ; for i < 8, no meaningful
                                rrcfc   DPX+B2              ; are in DPX+B0
                                rrcfc   DPX+B1

                    else

0206 1A1D          rlcfc   AARG+B1,W          ; rotate sign into carry bit
0207 191B          rrcfc   DPX+B3
0208 191A          rrcfc   DPX+B2
0209 1919          rrcfc   DPX+B1
020A 1918          rrcfc   DPX+B0

                    endif

000B              i = i+1

                    if i < 8
                                btfs    BARG+B0,i          ; test low byte
                    else

020B 931F          btfs    BARG+B1,i-8      ; test high byte
                    endif
020C C211          goto    noadd11
                    add11
020D 6A1C          movfp   AARG+B0,wreg
020E 0F1A          addwf   DPX+B2 ;add lsb
020F 6A1D          movfp   AARG+B1,wreg
0210 111B          addwfc  DPX+B3 ;add msb

                    noadd11
                    if i < 8

                                rlcfc   AARG+B1,W          ; rotate sign into carry bit
                                rrcfc   DPX+B3              ; for i < 8, no meaningful
                                rrcfc   DPX+B2              ; are in DPX+B0
                                rrcfc   DPX+B1

                    else

0211 1A1D          rlcfc   AARG+B1,W          ; rotate sign into carry bit
0212 191B          rrcfc   DPX+B3
```

Servo Control of a DC-Brush Motor

```
0213 191A                rrcf    DPX+B2
0214 1919                rrcf    DPX+B1
0215 1918                rrcf    DPX+B0

                                endif

000C                    i = i+1

                                if i < 8
                                btfs    BARG+B0,i          ; test low byte

                                else

0216 941F                btfs    BARG+B1,i-8          ; test high byte
                                endif
0217 C21C                goto    noadd12
                                add12
0218 6A1C                movfp   AARG+B0,wreg
0219 0F1A                addwf   DPX+B2 ;add lsb
021A 6A1D                movfp   AARG+B1,wreg
021B 111B                addwfc  DPX+B3 ;add msb

                                noadd12
                                if i < 8

                                rlcfc   AARG+B1,W          ; rotate sign into carry bit

                                rrcf    DPX+B3          ; for i < 8, no meaningful
                                rrcf    DPX+B2          ; are in DPX+B0
                                rrcf    DPX+B1

                                else

021C 1A1D                rlcfc   AARG+B1,W          ; rotate sign into carry bit
021D 191B                rrcf    DPX+B3
021E 191A                rrcf    DPX+B2
021F 1919                rrcf    DPX+B1
0220 1918                rrcf    DPX+B0

                                endif

000D                    i = i+1

                                if i < 8
                                btfs    BARG+B0,i          ; test low byte

                                else

0221 951F                btfs    BARG+B1,i-8          ; test high byte
                                endif
0222 C227                goto    noadd13
                                add13
0223 6A1C                movfp   AARG+B0,wreg
0224 0F1A                addwf   DPX+B2 ;add lsb
0225 6A1D                movfp   AARG+B1,wreg
0226 111B                addwfc  DPX+B3 ;add msb

                                noadd13
                                if i < 8

                                rlcfc   AARG+B1,W          ; rotate sign into carry bit
```


Servo Control of a DC-Brush Motor

```
rrcf DPX+B3 ; for i < 8, no meaningful
rrcf DPX+B2 ; are in DPX+B0
rrcf DPX+B1

else

0227 1A1D rlcfc AARG+B1,W ; rotate sign into carry bit
0228 191B rrcfc DPX+B3
0229 191A rrcfc DPX+B2
022A 1919 rrcfc DPX+B1
022B 1918 rrcfc DPX+B0

endif

000E i = i+1

if i < 8
    btfss BARG+B0,i ; test low byte
else
    btfss BARG+B1,i-8 ; test high byte
endif
goto noadd14
add14
022E 6A1C movfp AARG+B0,wreg
022F 0F1A addwf DPX+B2 ;add lsb
0230 6A1D movfp AARG+B1,wreg
0231 111B addwfc DPX+B3 ;add msb

noadd14
if i < 8

    rlcfc AARG+B1,W ; rotate sign into carry bit
    rrcfc DPX+B3 ; for i < 8, no meaningful
    rrcfc DPX+B2 ; are in DPX+B0
    rrcfc DPX+B1

else

0232 1A1D rlcfc AARG+B1,W ; rotate sign into carry bit
0233 191B rrcfc DPX+B3
0234 191A rrcfc DPX+B2
0235 1919 rrcfc DPX+B1
0236 1918 rrcfc DPX+B0

endif

000F i = i+1

if SIGNED

0237 1A1D rlcfc AARG+B1,W ; since BARG is always made
0238 191B rrcfc DPX+B3 ; the last bit is known to be
0239 191A rrcfc DPX+B2
023A 1919 rrcfc DPX+B1
023B 1918 rrcfc DPX+B0
```

Servo Control of a DC-Brush Motor

```

endif

023C 0002          return

;*****

                                include "traject.asm" ;
Trajectory Generation
;*****
;
;                               Trajectory Generation Routines
;
;*****

;*****
; NAME:          doPreMove
;
; DESCRIPTION:

doPreMove:

CLR16  INTEGRAL

023D 2996          CLRF    INTEGRAL+B0
023E 2997          CLRF    INTEGRAL+B1

MOV24  NMOVVAL,MOVVAL          ; move buffer to MOVVAL

023F 6A5B          MOVFP   NMOVVAL+B0,wreg          ; get byte of NMOVVAL into w
0240 4A5F          MOVFP   wreg,MOVVAL+B0          ; move to MOVVAL (B0)
0241 6A5C          MOVFP   NMOVVAL+B1,wreg          ; get byte of NMOVVAL into w
0242 4A60          MOVFP   wreg,MOVVAL+B1          ; move to MOVVAL (B1)
0243 6A5D          MOVFP   NMOVVAL+B2,wreg          ; get byte of NMOVVAL into w
0244 4A61          MOVFP   wreg,MOVVAL+B2          ; move to MOVVAL (B2)

0245 8F93          bcf     MOVSTAT,bit7          ; clear buffer flag
0246 8693          bsf     MOVSTAT,bit6          ; set motion status flag
0247 8593          bsf     MOVSTAT,bit5          ; set move in progress flag
0248 6AC2          movfp  ONE,wreg
0249 4A94          movpf  wreg,MOVFLAG          ; initialize MOVEFLAG to 1

024A 2951          clrf   OPOSITION+B0          ; initialize buffers
MOV24  POSITION,OPOSITION+B1

024B 6A55          MOVFP   POSITION+B0,wreg          ; get byte of POSITION into w
024C 4A52          MOVFP   wreg,OPOSITION+B1+B0          ; move to OPOSITION+B1 (B0)
024D 6A56          MOVFP   POSITION+B1,wreg          ; get byte of POSITION into w
024E 4A53          MOVFP   wreg,OPOSITION+B1+B1          ; move to OPOSITION+B1 (B1)
024F 6A57          MOVFP   POSITION+B2,wreg          ; get byte of POSITION into w
0250 4A54          MOVFP   wreg,OPOSITION+B1+B2          ; move to OPOSITION+B1 (B2)

MOV32  OPOSITION,MOVPCBUF

0251 6A51          MOVFP   OPOSITION+B0,wreg          ; get byte of OPOSITION into w
0252 4AA4          MOVFP   wreg,MOVPCBUF+B0          ; move to MOVPCBUF (B0)
0253 6A52          MOVFP   OPOSITION+B1,wreg          ; get byte of OPOSITION into w

```

Servo Control of a DC-Brush Motor

```

0254 4AA5          MOVFP  wreg,MOVFBUF+B1          ; move to MOVFBUF (B1)
0255 6A53          MOVFP  OPOSITION+B2,wreg      ; get byte of OPOSITION into w
0256 4AA6          MOVFP  wreg,MOVFBUF+B2      ; move to MOVFBUF (B2)
0257 6A54          MOVFP  OPOSITION+B3,wreg      ; get byte of OPOSITION into w
0258 4AA7          MOVFP  wreg,MOVFBUF+B3      ; move to MOVFBUF (B3)

0259 2995          CLR16  clrf  SATFLAG
CLR16  MOVTIME          ; clear move times

025A 2967          CLR16  MOVTIME+B0
025B 2968          CLR16  MOVTIME+B1

CLR16  T1              ; 0 used as flag for no maximum

025C 296A          CLR16  T1+B0
025D 296B          CLR16  T1+B1

CLR16  T2

025E 296C          CLR16  T2+B0
025F 296D          CLR16  T2+B1

CLR16  TAU

0260 296E          CLR16  TAU+B0
0261 296F          CLR16  TAU+B1

CLR32  MOVDEL          ; clear move discretization error

0262 29B0          CLR16  MOVDEL+B0
0263 29B1          CLR16  MOVDEL+B1
0264 29B2          CLR16  MOVDEL+B2
0265 29B3          CLR16  MOVDEL+B3

CLR16  PH2FLAT          ; clear phase 2 flat counter

0266 29B4          CLR16  PH2FLAT+B0
0267 29B5          CLR16  PH2FLAT+B1

0268 3391          tstfsz  MODETYPE
0269 C2C5          goto   vmode
pmode
MOVFP24 MOVVAL,TMP

026A 785F          MOVFP  MOVVAL+B0,TMP+B0          ; move MOVVAL (B0) to TMP (B0)
026B 7960          MOVFP  MOVVAL+B1,TMP+B1          ; move MOVVAL (B1) to TMP (B1)
026C 7A61          MOVFP  MOVVAL+B2,TMP+B2          ; move MOVVAL (B2) to TMP (B2)

026D 971A          btfss  TMP+B2,MSB
026E C276          goto   mvpos
NEG24  TMP

026F 1318          COMF   TMP+B0
0270 1319          COMF   TMP+B1
0271 131A          COMF   TMP+B2
0272 290A          CLR16  wreg
0273 1518          INCF   TMP+B0
0274 1119          ADDWFC TMP+B1
0275 111A          ADDWFC TMP+B2

```

Servo Control of a DC-Brush Motor

```

                                mvpos
0276 291C                clrf    MOVTMP+B0                ; calculate abs(MOVVAL) - 3
0277 291D                clrf    MOVTMP+B1                ; do immediate move if nega-
tive
0278 291E                clrf    MOVTMP+B2
0279 801C                bsf     MOVTMP+B0,bit0
027A 811C                bsf     MOVTMP+B0,bit1
                                SUB24  MOVTMP, TMP

027B 6A1C                MOVFP   MOVTMP+B0,wreg                ; get lowest byte of MOVTMP
027C 0518                SUBWF  TMP+B0                        ; sub lowest byte of TMP, save
027D 6A1D                MOVFP   MOVTMP+B1,wreg                ; get 2nd byte of MOVTMP into
027E 0319                SUBWF  TMP+B1                        ; sub 2nd byte of TMP, save in
027F 6A1E                MOVFP   MOVTMP+B2,wreg                ; get 3rd byte of MOVTMP into
0280 031A                SUBWF  TMP+B2                        ; sub 3rd byte of TMP, save in

0281 971A                btfsz  TMP+B2,MSB                    ; check for zero move
0282 C28E                goto   nonzero
0283 2B90                setf   SERVOFLAG                    ; set servoflag to restore
0284 2994                clrf   MOVFLAG
0285 8D93                bcf    MOVSTAT,bit5
0286 8E93                bcf    MOVSTAT,bit6
                                ADD24  MOVVAL, POSITION

0287 6A5F                MOVFP   MOVVAL+B0,wreg                ; get lowest byte of MOVVAL
0288 0F55                ADDWF  POSITION+B0                    ; add lowest byte of POSITION,
0289 6A60                MOVFP   MOVVAL+B1,wreg                ; get 2nd byte of MOVVAL into
028A 1156                ADDWFC POSITION+B1                    ; add 2nd byte of POSITION,
028B 6A61                MOVFP   MOVVAL+B2,wreg                ; get 3rd byte of MOVVAL into
028C 1157                ADDWFC POSITION+B2                    ; add 3rd byte of POSITION,

028D 0002                return
                                nonzero
                                CLR32  MOVVBUF

028E 29A8                CLRF   MOVVBUF+B0
028F 29A9                CLRF   MOVVBUF+B1
0290 29AA                CLRF   MOVVBUF+B2
0291 29AB                CLRF   MOVVBUF+B3

0292 6A61                movfp  MOVVAL+B2,wreg                ; move sign
0293 B580                andlw  0x80
0294 4A69                movpf  wreg,MOVSIGN

0295 29A3                clrf   V+B3                          ; create appropriate velocity
                                MOV24  VL,V                          ; acceleration limits from

0296 6A20                MOVFP   VL+B0,wreg                    ; get byte of VL into w
0297 4AA0                MOVFP   wreg,V+B0                    ; move to V(B0)
0298 6A21                MOVFP   VL+B1,wreg                    ; get byte of VL into w
0299 4AA1                MOVFP   wreg,V+B1                    ; move to V(B1)
029A 6A22                MOVFP   VL+B2,wreg                    ; get byte of VL into w
029B 4AA2                MOVFP   wreg,V+B2                    ; move to V(B2)

029C 299F                clrf   A+B3
                                MOV24  AL,A

029D 6A23                MOVFP   AL+B0,wreg                    ; get byte of AL into w
029E 4A9C                MOVFP   wreg,A+B0                    ; move to A(B0)
029F 6A24                MOVFP   AL+B1,wreg                    ; get byte of AL into w
02A0 4A9D                MOVFP   wreg,A+B1                    ; move to A(B1)
02A1 6A25                MOVFP   AL+B2,wreg                    ; get byte of AL into w

```

Servo Control of a DC-Brush Motor

```

02A2 4A9E          MOVFP  wreg,A+B2          ; move to A(B2)

02A3 290A          clrfl  wreg
02A4 3269          cpfsgt MOVSIGN
02A5 C2B8          goto   minc
                    NEG32  V

02A6 13A0          COMF   V+B0
02A7 13A1          COMF   V+B1
02A8 13A2          COMF   V+B2
02A9 13A3          COMF   V+B3
02AA 290A          CLRFL  wreg
02AB 15A0          INCF   V+B0
02AC 11A1          ADDWFC V+B1
02AD 11A2          ADDWFC V+B2
02AE 11A3          ADDWFC V+B3

                    NEG32  A

02AF 139C          COMF   A+B0
02B0 139D          COMF   A+B1
02B1 139E          COMF   A+B2
02B2 139F          COMF   A+B3
02B3 290A          CLRFL  wreg
02B4 159C          INCF   A+B0
02B5 119D          ADDWFC A+B1
02B6 119E          ADDWFC A+B2
02B7 119F          ADDWFC A+B3

                    minc

02B8 2963          clrfl  HMOVVAL+B0          ; evaluate MOVVAL/2
                    MOV24  MOVVAL,HMOVVAL+B1

02B9 6A5F          MOVFP  MOVVAL+B0,wreg          ; get byte of MOVVAL into w
02BA 4A64          MOVFP  wreg,HMOVVAL+B1+B0      ; move to HMOVVAL+B1(B0)
02BB 6A60          MOVFP  MOVVAL+B1,wreg          ; get byte of MOVVAL into w
02BC 4A65          MOVFP  wreg,HMOVVAL+B1+B1      ; move to HMOVVAL+B1(B1)
02BD 6A61          MOVFP  MOVVAL+B2,wreg          ; get byte of MOVVAL into w
02BE 4A66          MOVFP  wreg,HMOVVAL+B1+B2      ; move to HMOVVAL+B1(B2)

                    RRC32  HMOVVAL          ; half move in Q8

02BF 1A66          RLCLF  HMOVVAL+B3,W          ; move sign into carry bit
02C0 1966          RRCFL  HMOVVAL+B3
02C1 1965          RRCFL  HMOVVAL+B2
02C2 1964          RRCFL  HMOVVAL+B1
02C3 1963          RRCFL  HMOVVAL+B0

02C4 C2FE          goto   modeready

                    vmode

02C5 9F91          btfscl  MODETYPE,MSB          ; is it torque move?
02C6 C306          goto   tmode

02C7 2966          clrfl  HMOVVAL+B3          ; compute final minus initial
                    MOV24  MOVVAL,HMOVVAL

02C8 6A5F          MOVFP  MOVVAL+B0,wreg          ; get byte of MOVVAL into w
02C9 4A63          MOVFP  wreg,HMOVVAL+B0      ; move to HMOVVAL(B0)
02CA 6A60          MOVFP  MOVVAL+B1,wreg          ; get byte of MOVVAL into w
02CB 4A64          MOVFP  wreg,HMOVVAL+B1      ; move to HMOVVAL(B1)
02CC 6A61          MOVFP  MOVVAL+B2,wreg          ; get byte of MOVVAL into w
02CD 4A65          MOVFP  wreg,HMOVVAL+B2      ; move to HMOVVAL(B2)

```

Servo Control of a DC-Brush Motor

```

02CE 9F61          btfscc MOVVAL+B2,MSB
02CF 2B66          setf  HMOVVAL+B3
                SUB32  MOVVBUFF,HMOVVAL

02D0 6AA8          MOVFP  MOVVBUFF+B0,wreg      ; get lowest byte of MOVVBUFF into w
02D1 0563          SUBWF  HMOVVAL+B0          ; sub lowest byte of HMOVVAL, save in
02D2 6AA9          MOVFP  MOVVBUFF+B1,wreg      ; get 2nd byte of MOVVBUFF into w
02D3 0364          SUBWFB HMOVVAL+B1          ; sub 2nd byte of HMOVVAL, save in
02D4 6AAA          MOVFP  MOVVBUFF+B2,wreg      ; get 3rd byte of MOVVBUFF into w
02D5 0365          SUBWFB HMOVVAL+B2          ; sub 3rd byte of HMOVVAL, save in
02D6 6AAB          MOVFP  MOVVBUFF+B3,wreg      ; get 4th byte of MOVVBUFF into w
02D7 0366          SUBWFB HMOVVAL+B3          ; sub 4th byte of HMOVVAL, save in

02D8 6A66          movfp  HMOVVAL+B3,wreg
02D9 B580          andlw  0x80
02DA 4A69          movpf  wreg,MOVSIGN

02DB 29A3          clrfs  V+B3                ; create appropriate velocity and
                MOV24  VL,V                ; acceleration limits from move sign

02DC 6A20          MOVFP  VL+B0,wreg          ; get byte of VL into w
02DD 4AA0          MOVFP  wreg,V+B0          ; move to V(B0)
02DE 6A21          MOVFP  VL+B1,wreg          ; get byte of VL into w
02DF 4AA1          MOVFP  wreg,V+B1          ; move to V(B1)
02E0 6A22          MOVFP  VL+B2,wreg          ; get byte of VL into w
02E1 4AA2          MOVFP  wreg,V+B2          ; move to V(B2)

02E2 299F          clrfs  A+B3
                MOV24  AL,A

02E3 6A23          MOVFP  AL+B0,wreg          ; get byte of AL into w
02E4 4A9C          MOVFP  wreg,A+B0          ; move to A(B0)
02E5 6A24          MOVFP  AL+B1,wreg          ; get byte of AL into w
02E6 4A9D          MOVFP  wreg,A+B1          ; move to A(B1)
02E7 6A25          MOVFP  AL+B2,wreg          ; get byte of AL into w
02E8 4A9E          MOVFP  wreg,A+B2          ; move to A(B2)

02E9 290A          clrfs  wreg
02EA 3269          cplfsgt MOVSIGN
02EB C2FE          goto  modeready
                NEG32  V

02EC 13A0          COMF   V+B0
02ED 13A1          COMF   V+B1
02EE 13A2          COMF   V+B2
02EF 13A3          COMF   V+B3
02F0 290A          CLRFS  wreg
02F1 15A0          INCF  V+B0
02F2 11A1          ADDWFC V+B1
02F3 11A2          ADDWFC V+B2
02F4 11A3          ADDWFC V+B3

                NEG32  A

02F5 139C          COMF   A+B0
02F6 139D          COMF   A+B1
02F7 139E          COMF   A+B2
02F8 139F          COMF   A+B3
02F9 290A          CLRFS  wreg
02FA 159C          INCF  A+B0
02FB 119D          ADDWFC A+B1
02FC 119E          ADDWFC A+B2
02FD 119F          ADDWFC A+B3

```

Servo Control of a DC-Brush Motor

```
modeready
02FE 2962      clr  MOVVAL+B3
02FF 9F61      btfs  MOVVAL+B2,MSB
0300 2B62      setf  MOVVAL+B3

0301 2B90      setf  SERVOFLAG          ; set servoflag to restore servo
                                           ; if stopped

      if      _PICMASTER_DEBUG

;*****
;      For PICMASTER Debug/servo tuning puporses only Purposes Only
;
testCapCount
0302 6ABC      movfp  CAPCOUNT+B0,wreg
0303 08BD      lorwf  CAPCOUNT+B1,W
0304 4ABB      movpf  wreg,CAPFLAG
;*****
endif

0305 0002      return

tmode          ; torque/voltage mode
MOV16 MOVVAL+B1,YPWM          ; set new commanded value

0306 6A60      MOVFP  MOVVAL+B1+B0,wreg          ; get byte of MOVVAL+B1 into w
0307 0188      MOVWF  YPWM+B0                    ; move to YPWM(B0)
0308 6A61      MOVFP  MOVVAL+B1+B1,wreg          ; get byte of MOVVAL+B1 into w
0309 0189      MOVWF  YPWM+B1                    ; move to YPWM(B1)

030A 2990      clr  SERVOFLAG          ; disable servo
030B E50B      call doTorque          ; set pwm duty cycle
030C 2994      clr  MOVFLAG
030D 8D93      bcf  MOVSTAT,bit5

      if      _PICMASTER_DEBUG
030E C302      goto  testCapCount
      else
      return
endif

;*****
;*****
; NAME:          doMove
;
; DESCRIPTION:   In position mode, trapezoidal moves are performed. Phasel
;                and phase2 respectively, are the periods for the first and
;                second halves of the move. The move time is defined as zero
;                at the beginning of the move,T2 is the time at half the
;                move, T1 is the time when c
;                begins,(the region of constant velocity reduces to a point
;                in the case where maximum speed is not realized, and the
;                trapezoidal move degenerates into a trianglular move,
;                together with T1=T2), and TAU is the total time of the move.
;                The accelerations are +-AL or 0.
;
;
```



Servo Control of a DC-Brush Motor

```

0318 7CB0      MOVFP  MOVDEL+B0,MOVTMP+B0      ; move MOVDEL (B0) to MOVTMP (B0)
0319 7DB1      MOVFP  MOVDEL+B1,MOVTMP+B1      ; move MOVDEL (B1) to MOVTMP (B1)
031A 7EB2      MOVFP  MOVDEL+B2,MOVTMP+B2      ; move MOVDEL (B2) to MOVTMP (B2)
031B 7FB3      MOVFP  MOVDEL+B3,MOVTMP+B3      ; move MOVDEL (B3) to MOVTMP (B3)

MOV32  OPOSITION,MOVDEL      ; test if half move

031C 6A51      MOVFP  OPOSITION+B0,wreg      ; get byte of OPOSITION into w
031D 4AB0      MOVFP  wreg,MOVDEL+B0      ; move to MOVDEL (B0)
031E 6A52      MOVFP  OPOSITION+B1,wreg      ; get byte of OPOSITION into w
031F 4AB1      MOVFP  wreg,MOVDEL+B1      ; move to MOVDEL (B1)
0320 6A53      MOVFP  OPOSITION+B2,wreg      ; get byte of OPOSITION into w
0321 4AB2      MOVFP  wreg,MOVDEL+B2      ; move to MOVDEL (B2)
0322 6A54      MOVFP  OPOSITION+B3,wreg      ; get byte of OPOSITION into w
0323 4AB3      MOVFP  wreg,MOVDEL+B3      ; move to MOVDEL (B3)

ADD32  HMOVVAL,MOVDEL

0324 6A63      MOVFP  HMOVVAL+B0,wreg      ; get lowest byte of HMOVVAL into w
0325 0FB0      ADDWF  MOVDEL+B0      ; add lowest byte of MOVDEL, save in
0326 6A64      MOVFP  HMOVVAL+B1,wreg      ; get 2nd byte of HMOVVAL into w
0327 11B1      ADDWFC MOVDEL+B1      ; add 2nd byte of MOVDEL, save in
0328 6A65      MOVFP  HMOVVAL+B2,wreg      ; get 3rd byte of HMOVVAL into w
0329 11B2      ADDWFC MOVDEL+B2      ; add 3rd byte of MOVDEL, save in
032A 6A66      MOVFP  HMOVVAL+B3,wreg      ; get 4th byte of HMOVVAL into w
032B 11B3      ADDWFC MOVDEL+B3      ; add 4th byte of MOVDEL, save in

SUB32  MOVPCBUF,MOVDEL

032C 6AA4      MOVFP  MOVPCBUF+B0,wreg      ; get lowest byte of MOVPCBUF into
032D 05B0      SUBWF  MOVDEL+B0      ; sub lowest byte of MOVDEL, save
032E 6AA5      MOVFP  MOVPCBUF+B1,wreg      ; get 2nd byte of MOVPCBUF into w
032F 03B1      SUBWFB MOVDEL+B1      ; sub 2nd byte of MOVDEL, save in
0330 6AA6      MOVFP  MOVPCBUF+B2,wreg      ; get 3rd byte of MOVPCBUF into w
0331 03B2      SUBWFB MOVDEL+B2      ; sub 3rd byte of MOVDEL, save in
0332 6AA7      MOVFP  MOVPCBUF+B3,wreg      ; get 4th byte of MOVPCBUF into w
0333 03B3      SUBWFB MOVDEL+B3      ; sub 4th byte of MOVDEL, save in

0334 9769      btfss  MOVSIGN,MSB
0335 C33F      goto   mpos1

NEG32  MOVDEL

0336 13B0      COMF  MOVDEL+B0
0337 13B1      COMF  MOVDEL+B1
0338 13B2      COMF  MOVDEL+B2
0339 13B3      COMF  MOVDEL+B3
033A 290A      CLRF  wreg
033B 15B0      INCF  MOVDEL+B0
033C 11B1      ADDWFC MOVDEL+B1
033D 11B2      ADDWFC MOVDEL+B2
033E 11B3      ADDWFC MOVDEL+B3

mpos1

033F 97B3      btfss  MOVDEL+B3,MSB
0340 C3A5      goto   speedup      ; continue to speed up if in

TFSZ16 T1      ; if T1=0, maximum velocity not

0341 6A6A      MOVFP  T1+B0,wreg
0342 086B      IORWF  T1+B1,W
0343 330A      TSTFSZ wreg

; reached,

```

Servo Control of a DC-Brush Motor

```

; has been set in speedup
0344 C378          goto    t2net1

                NEG32   A          ; negate A for speeddown

0345 139C          COMF    A+B0
0346 139D          COMF    A+B1
0347 139E          COMF    A+B2
0348 139F          COMF    A+B3
0349 290A          CLRF   wreg
034A 159C          INCF   A+B0
034B 119D          ADDWFC A+B1
034C 119E          ADDWFC A+B2
034D 119F          ADDWFC A+B3

                ADD32   MOVDEL,MOVTMP ; test x-y < 0

034E 6AB0          MOVFP   MOVDEL+B0,wreg ; get lowest byte of MOVDEL into
034F 0F1C          ADDWF  MOVTMP+B0          ; add lowest byte of MOVTMP, save
0350 6AB1          MOVFP   MOVDEL+B1,wreg ; get 2nd byte of MOVDEL into w
0351 111D          ADDWFC MOVTMP+B1          ; add 2nd byte of MOVTMP, save in
0352 6AB2          MOVFP   MOVDEL+B2,wreg ; get 3rd byte of MOVDEL into w
0353 111E          ADDWFC MOVTMP+B2          ; add 3rd byte of MOVTMP, save in
0354 6AB3          MOVFP   MOVDEL+B3,wreg ; get 4th byte of MOVDEL into w
0355 111F          ADDWFC MOVTMP+B3          ; add 4th byte of MOVTMP, save in

0356 971F          btfs   MOVTMP+B3,MSB ; if new discretization error larger,
0357 C36E          goto    triok          ; backup to define T2, otherwise ok

0358 2B6C          setf   T2+B0          ; set T2=-1 for backup
0359 2B6D          setf   T2+B1

                NEG32   A          ; negate A to undo

035A 139C          COMF    A+B0
035B 139D          COMF    A+B1
035C 139E          COMF    A+B2
035D 139F          COMF    A+B3
035E 290A          CLRF   wreg
035F 159C          INCF   A+B0
0360 119D          ADDWFC A+B1
0361 119E          ADDWFC A+B2
0362 119F          ADDWFC A+B3

0363 E48A          call   undoPosVel ; negate A again for speeddown

                NEG32   A

0364 139C          COMF    A+B0
0365 139D          COMF    A+B1
0366 139E          COMF    A+B2
0367 139F          COMF    A+B3
0368 290A          CLRF   wreg
0369 159C          INCF   A+B0
036A 119D          ADDWFC A+B1
036B 119E          ADDWFC A+B2
036C 119F          ADDWFC A+B3

036D E468          call   doPosVel ; and reevaluate iterative equations
                triok
                ADD16   MOVTIME,T2 ; add time to T2

036E 6A67          MOVFP   MOVTIME+B0,wreg ; get lowest byte of MOVTIME into w
036F 0F6C          ADDWF  T2+B0          ; add lowest byte of T2, save in
0370 6A68          MOVFP   MOVTIME+B1,wreg ; get 2nd byte of MOVTIME into w
0371 116D          ADDWFC T2+B1          ; add 2nd byte of T2, save in T2(B1)

```

Servo Control of a DC-Brush Motor

```

MOV16  T2,T1

0372 6A6C          MOVFP  T2+B0,wreg          ; get byte of T2 into w
0373 016A          MOVWF  T1+B0                ; move to T1(B0)
0374 6A6D          MOVFP  T2+B1,wreg          ; get byte of T2 into w
0375 016B          MOVWF  T1+B1                ; move to T1(B1)

0376 1594          incf   MOVFLAG              ; increment move flag for
0377 C3CE          goto   mvok                 ; execute last phasel move

t2net1
0378 2B6C          setf   T2+B0                ; set T2=-1 for backup
0379 2B6D          setf   T2+B1

ADD16  MOVTIME,T2          ; add time to T2

037A 6A67          MOVFP  MOVTIME+B0,wreg     ; get lowest byte of MOVTIME
037B 0F6C          ADDWF  T2+B0                ; add lowest byte of T2, save
037C 6A68          MOVFP  MOVTIME+B1,wreg     ; get 2nd byte of MOVTIME into
037D 116D          ADDWFC T2+B1                ; add 2nd byte of T2, save in
T2 (B1)

MOVFP32 MOVTMP,TMP          ; test if 3x-y < 0

037E 781C          MOVFP  MOVTMP+B0,TMP+B0    ; move MOVTMP (B0) to TMP (B0)
037F 791D          MOVFP  MOVTMP+B1,TMP+B1    ; move MOVTMP (B1) to TMP (B1)
0380 7A1E          MOVFP  MOVTMP+B2,TMP+B2    ; move MOVTMP (B2) to TMP (B2)
0381 7B1F          MOVFP  MOVTMP+B3,TMP+B3    ; move MOVTMP (B3) to TMP (B3)

RLC32  MOVTMP

0382 8804          BCF    _carry
0383 1B1C          RLCF  MOVTMP+B0
0384 1B1D          RLCF  MOVTMP+B1
0385 1B1E          RLCF  MOVTMP+B2
0386 1B1F          RLCF  MOVTMP+B3

ADD32  TMP,MOVTMP

0387 6A18          MOVFP  TMP+B0,wreg          ; get lowest byte of TMP into
0388 0F1C          ADDWF  MOVTMP+B0            ; add lowest byte of MOVTMP,
0389 6A19          MOVFP  TMP+B1,wreg          ; get 2nd byte of TMP into w
038A 111D          ADDWFC MOVTMP+B1            ; add 2nd byte of MOVTMP, save
038B 6A1A          MOVFP  TMP+B2,wreg          ; get 3rd byte of TMP into w
038C 111E          ADDWFC MOVTMP+B2            ; add 3rd byte of MOVTMP, save
038D 6A1B          MOVFP  TMP+B3,wreg          ; get 4th byte of TMP into w
038E 111F          ADDWFC MOVTMP+B3            ; add 4th byte of MOVTMP, save

ADD32  MOVDEL,MOVTMP

038F 6AB0          MOVFP  MOVDEL+B0,wreg       ; get lowest byte of MOVDEL
0390 0F1C          ADDWF  MOVTMP+B0            ; add lowest byte of MOVTMP,
0391 6AB1          MOVFP  MOVDEL+B1,wreg       ; get 2nd byte of MOVDEL into
0392 111D          ADDWFC MOVTMP+B1            ; add 2nd byte of MOVTMP, save
0393 6AB2          MOVFP  MOVDEL+B2,wreg       ; get 3rd byte of MOVDEL into
0394 111E          ADDWFC MOVTMP+B2            ; add 3rd byte of MOVTMP, save
0395 6AB3          MOVFP  MOVDEL+B3,wreg       ; get 4th byte of MOVDEL into
0396 111F          ADDWFC MOVTMP+B3            ; add 4th byte of MOVTMP, save

```

Servo Control of a DC-Brush Motor

```

0397 971F          btfss  MOVTMP+B3,MSB          ; if new discretization error
0398 C39B          goto   trapok                  ; take one more flat step
0399 2BB4          setf   PH2FLAT+B0
039A 2BB5          setf   PH2FLAT+B1

trapok
ADD16  T2,PH2FLAT

039B 6A6C          MOVFP  T2+B0,wreg          ; get lowest byte of T2 into w
039C 0FB4          ADDWF  PH2FLAT+B0        ; add lowest byte of PH2FLAT,
039D 6A6D          MOVFP  T2+B1,wreg          ; get 2nd byte of T2 into w
039E 11B5          ADDWFC PH2FLAT+B1        ; add 2nd byte of PH2FLAT,

SUB16  T1,PH2FLAT

039F 6A6A          MOVFP  T1+B0,wreg          ; get lowest byte of T1 into w
03A0 05B4          SUBWF  PH2FLAT+B0        ; sub lowest byte of PH2FLAT,
03A1 6A6B          MOVFP  T1+B1,wreg          ; get 2nd byte of T1 into w
03A2 03B5          SUBWFB PH2FLAT+B1        ; sub 2nd byte of PH2FLAT,

03A3 1594          incf   MOVFLAG          ; increment move flag for
03A4 C3CE          goto   mvok            ; execute last phasel move

speedup
MOVFP32 V,MOVTMP          ; test if maximum velocity

03A5 7CA0          MOVFP  V+B0,MOVTMP+B0    ; move V(B0) to MOVTMP (B0)
03A6 7DA1          MOVFP  V+B1,MOVTMP+B1    ; move V(B1) to MOVTMP (B1)
03A7 7EA2          MOVFP  V+B2,MOVTMP+B2    ; move V(B2) to MOVTMP (B2)
03A8 7FA3          MOVFP  V+B3,MOVTMP+B3    ; move V(B3) to MOVTMP (B3)

SUB32  MOVVBUF,MOVTMP

03A9 6AA8          MOVFP  MOVVBUF+B0,wreg    ; get lowest byte of MOVVBUF
03AA 051C          SUBWF  MOVTMP+B0        ; sub lowest byte of MOVTMP,
03AB 6AA9          MOVFP  MOVVBUF+B1,wreg    ; get 2nd byte of MOVVBUF into
03AC 031D          SUBWFB MOVTMP+B1        ; sub 2nd byte of MOVTMP, save
03AD 6AAA          MOVFP  MOVVBUF+B2,wreg    ; get 3rd byte of MOVVBUF into
03AE 031E          SUBWFB MOVTMP+B2        ; sub 3rd byte of MOVTMP, save
03AF 6AAB          MOVFP  MOVVBUF+B3,wreg    ; get 4th byte of MOVVBUF into
03B0 031F          SUBWFB MOVTMP+B3        ; sub 4th byte of MOVTMP, save
in

03B1 9769          btfss  MOVSIGN,MSB      ;
03B2 C3BC          goto   mpos            ;

NEG32  MOVTMP

03B3 131C          COMF  MOVTMP+B0
03B4 131D          COMF  MOVTMP+B1
03B5 131E          COMF  MOVTMP+B2
03B6 131F          COMF  MOVTMP+B3
03B7 290A          CLRF  wreg
03B8 151C          INCF  MOVTMP+B0
03B9 111D          ADDWFC MOVTMP+B1
03BA 111E          ADDWFC MOVTMP+B2
03BB 111F          ADDWFC MOVTMP+B3

mpos

03BC 971F          btfss  MOVTMP+B3,MSB    ;
03BD C3CE          goto   mvok            ; if not, execute move

TFSZ16  T1          ; if so, check to see if T1

03BE 6A6A          MOVFP  T1+B0,wreg
03BF 086B          IORWF T1+B1,W

```

Servo Control of a DC-Brush Motor

```

03C0 330A          TSTFSZ  wreg
                                ; already been set
03C1 C3CE          goto  mvok
03C2 E48A          call   undoPosVel          ; if not, backup and redo
                                ; equations, resulting in an
CLR32  A
03C3 299C          CLRF  A+B0
03C4 299D          CLRF  A+B1
03C5 299E          CLRF  A+B2
03C6 299F          CLRF  A+B3

03C7 E468          call   doPosVel          ; maximum speed <= VL
03C8 2B6A          setf  T1+B0          ; evaluate T1
03C9 2B6B          setf  T1+B1
ADD16  MOVTIME, T1

03CA 6A67          MOVFP  MOVTIME+B0, wreg      ; get lowest byte of MOVTIME
03CB 0F6A          ADDWF  T1+B0          ; add lowest byte of T1, save
03CC 6A68          MOVFP  MOVTIME+B1, wreg      ; get 2nd byte of MOVTIME into
03CD 116B          ADDWFC T1+B1          ; add 2nd byte of T1, save in

mvok
MOV24  MOVVPBUF+B1, POSITION      ; move Q8 calculated position

03CE 6AA5          MOVFP  MOVVPBUF+B1+B0, wreg      ; get byte of MOVVPBUF+B1 into
03CF 4A55          MOVFP  wreg, POSITION+B0      ; move to POSITION (B0)
03D0 6AA6          MOVFP  MOVVPBUF+B1+B1, wreg      ; get byte of MOVVPBUF+B1 into
03D1 4A56          MOVFP  wreg, POSITION+B1      ; move to POSITION (B1)
03D2 6AA7          MOVFP  MOVVPBUF+B1+B2, wreg      ; get byte of MOVVPBUF+B1 into
03D3 4A57          MOVFP  wreg, POSITION+B2      ; move to POSITION (B2)

MOV24  MOVVBUF+B0, VELOCITY      ; move Q0 calculated velocity

03D4 6AA8          MOVFP  MOVVBUF+B0+B0, wreg      ; get byte of MOVVBUF+B0 into
03D5 4A58          MOVFP  wreg, VELOCITY+B0      ; move to VELOCITY (B0)
03D6 6AA9          MOVFP  MOVVBUF+B0+B1, wreg      ; get byte of MOVVBUF+B0 into
03D7 4A59          MOVFP  wreg, VELOCITY+B1      ; move to VELOCITY (B1)
03D8 6AAA          MOVFP  MOVVBUF+B0+B2, wreg      ; get byte of MOVVBUF+B0 into
03D9 4A5A          MOVFP  wreg, VELOCITY+B2      ; move to VELOCITY (B2)

03DA 0002          return

                                phase2
TFSZ16 PH2FLAT          ; is flat section finished?

03DB 6AB4          MOVFP  PH2FLAT+B0, wreg
03DC 08B5          IORWF  PH2FLAT+B1, W
03DD 330A          TSTFSZ  wreg

03DE C3FF          goto  flat

TFSZ32 MOVVBUF          ; is velocity zero?

03DF 6AA8          MOVFP  MOVVBUF+B0, wreg
03E0 08A9          IORWF  MOVVBUF+B1, W
03E1 08AA          IORWF  MOVVBUF+B2, W
03E2 08AB          IORWF  MOVVBUF+B3, W
03E3 330A          TSTFSZ  wreg

03E4 C41C          goto  mready          ; if not, execute move

```



Servo Control of a DC-Brush Motor

```

03E5 2994          clrf    MOVFLAG          ; if so, clear MOVFLAG
03E6 8E93          bcf    MOVSTAT,bit6        ; clear motion status flag
03E7 8D93          bcf    MOVSTAT,bit5        ; clear move in progress flag
                    CLR32   A              ; set zero velocity and acceleration,

03E8 299C          CLRF   A+B0
03E9 299D          CLRF   A+B1
03EA 299E          CLRF   A+B2
03EB 299F          CLRF   A+B3

                    MOV16  MOVTIME,TAU

03EC 6A67          MOVFP  MOVTIME+B0,wreg     ; get byte of MOVTIME into w
03ED 016E          MOVWF  TAU+B0             ; move to TAU(B0)
03EE 6A68          MOVFP  MOVTIME+B1,wreg     ; get byte of MOVTIME into w
03EF 016F          MOVWF  TAU+B1             ; move to TAU(B1)

                    MOV32  OPOSITION,MOVBUF      ; execute last move to P(0)+MOVVAL

03F0 6A51          MOVFP  OPOSITION+B0,wreg   ; get byte of OPOSITION into w
03F1 4AA4          MOVFP  wreg,MOVBUF+B0      ; move to MOVBUF(B0)
03F2 6A52          MOVFP  OPOSITION+B1,wreg   ; get byte of OPOSITION into w
03F3 4AA5          MOVFP  wreg,MOVBUF+B1      ; move to MOVBUF(B1)
03F4 6A53          MOVFP  OPOSITION+B2,wreg   ; get byte of OPOSITION into w
03F5 4AA6          MOVFP  wreg,MOVBUF+B2      ; move to MOVBUF(B2)
03F6 6A54          MOVFP  OPOSITION+B3,wreg   ; get byte of OPOSITION into w
03F7 4AA7          MOVFP  wreg,MOVBUF+B3      ; move to MOVBUF(B3)

                    ADD24  MOVVAL,MOVBUF+B1

03F8 6A5F          MOVFP  MOVVAL+B0,wreg      ; get lowest byte of MOVVAL into w
03F9 0FA5          ADDWF  MOVBUF+B1+B0        ; add lowest byte of MOVBUF+B1, save
03FA 6A60          MOVFP  MOVVAL+B1,wreg     ; get 2nd byte of MOVVAL into w
03FB 11A6          ADDWFC MOVBUF+B1+B1        ; add 2nd byte of MOVBUF+B1, save in
03FC 6A61          MOVFP  MOVVAL+B2,wreg     ; get 3rd byte of MOVVAL into w
03FD 11A7          ADDWFC MOVBUF+B1+B2        ; add 3rd byte of MOVBUF+B1, save in

03FE C41C          goto   mready

                    flat

03FF 2B1C          setf  MOVTMP+B0
0400 2B1D          setf  MOVTMP+B1
                    ADD16  MOVTMP,PH2FLAT        ; decrement by one use DEC16

0401 6A1C          MOVFP  MOVTMP+B0,wreg     ; get lowest byte of MOVTMP into w
0402 0FB4          ADDWF  PH2FLAT+B0         ; add lowest byte of PH2FLAT, save in
0403 6A1D          MOVFP  MOVTMP+B1,wreg     ; get 2nd byte of MOVTMP into w
0404 11B5          ADDWFC PH2FLAT+B1        ; add 2nd byte of PH2FLAT, save in

                    TFSZ16 PH2FLAT

0405 6AB4          MOVFP  PH2FLAT+B0,wreg     ; get lowest byte of PH2FLAT into w
0406 08B5          TORWF PH2FLAT+B1,W       ; test for zero
0407 330A          TSTFSZ wreg

0408 C41C          goto   mready

0409 299F          clrf   A+B3              ; begin speed down section
                    MOV24  AL,A

040A 6A23          MOVFP  AL+B0,wreg         ; get byte of AL into w

```

Servo Control of a DC-Brush Motor

```

040B 4A9C          MOVFP  wreg,A+B0          ; move to A(B0)
040C 6A24          MOVFP  AL+B1,wreg        ; get byte of AL into w
040D 4A9D          MOVFP  wreg,A+B1          ; move to A(B1)
040E 6A25          MOVFP  AL+B2,wreg        ; get byte of AL into w
040F 4A9E          MOVFP  wreg,A+B2          ; move to A(B2)

0410 290A          clrfl  wreg
0411 3169          cpfseq MOVSIGN
0412 C41C          goto   mready

                NEG32  A

0413 139C          COMF   A+B0
0414 139D          COMF   A+B1
0415 139E          COMF   A+B2
0416 139F          COMF   A+B3
0417 290A          CLRF  wreg
0418 159C          INCF  A+B0
0419 119D          ADDWFC A+B1
041A 119E          ADDWFC A+B2
041B 119F          ADDWFC A+B3

mready
MOV24          MOVVBUF+B1,POSITION

041C 6AA5          MOVFP  MOVVBUF+B1+B0,wreg ; get byte of MOVVBUF+B1 into w
041D 4A55          MOVFP  wreg,POSITION+B0  ; move to POSITION(B0)
041E 6AA6          MOVFP  MOVVBUF+B1+B1,wreg ; get byte of MOVVBUF+B1 into w
041F 4A56          MOVFP  wreg,POSITION+B1  ; move to POSITION(B1)
0420 6AA7          MOVFP  MOVVBUF+B1+B2,wreg ; get byte of MOVVBUF+B1 into w
0421 4A57          MOVFP  wreg,POSITION+B2  ; move to POSITION(B2)

                MOV24  MOVVBUF+B0,VELOCITY

0422 6AA8          MOVFP  MOVVBUF+B0+B0,wreg ; get byte of MOVVBUF+B0 into w
0423 4A58          MOVFP  wreg,VELOCITY+B0  ; move to VELOCITY(B0)
0424 6AA9          MOVFP  MOVVBUF+B0+B1,wreg ; get byte of MOVVBUF+B0 into w
0425 4A59          MOVFP  wreg,VELOCITY+B1  ; move to VELOCITY(B1)
0426 6AAA          MOVFP  MOVVBUF+B0+B2,wreg ; get byte of MOVVBUF+B0 into w
0427 4A5A          MOVFP  wreg,VELOCITY+B2  ; move to VELOCITY(B2)

0428 0002          return

vmove
MOVFP32        MOVVAL,MOVTMP ; test if final velocity reached

0429 7C5F          MOVFP  MOVVAL+B0,MOVTMP+B0 ; move MOVVAL(B0) to MOVTMP(B0)
042A 7D60          MOVFP  MOVVAL+B1,MOVTMP+B1 ; move MOVVAL(B1) to MOVTMP(B1)
042B 7E61          MOVFP  MOVVAL+B2,MOVTMP+B2 ; move MOVVAL(B2) to MOVTMP(B2)
042C 7F62          MOVFP  MOVVAL+B3,MOVTMP+B3 ; move MOVVAL(B3) to MOVTMP(B3)

                SUB32  MOVVBUF,MOVTMP

042D 6AA8          MOVFP  MOVVBUF+B0,wreg    ; get lowest byte of MOVVBUF into w
042E 051C          SUBWF  MOVTMP+B0        ; sub lowest byte of MOVTMP, save in
042F 6AA9          MOVFP  MOVVBUF+B1,wreg    ; get 2nd byte of MOVVBUF into w
0430 031D          SUBWFB MOVTMP+B1        ; sub 2nd byte of MOVTMP, save in
0431 6AAA          MOVFP  MOVVBUF+B2,wreg    ; get 3rd byte of MOVVBUF into w
0432 031E          SUBWFB MOVTMP+B2        ; sub 3rd byte of MOVTMP, save in
0433 6AAB          MOVFP  MOVVBUF+B3,wreg    ; get 4th byte of MOVVBUF into w
0434 031F          SUBWFB MOVTMP+B3        ; sub 4th byte of MOVTMP, save in

0435 9769          btfss MOVSIGN,MSB
0436 C440          goto   vmpos

```

Servo Control of a DC-Brush Motor

```

NEG32  MOVTMP

0437 131C          COMF  MOVTMP+B0
0438 131D          COMF  MOVTMP+B1
0439 131E          COMF  MOVTMP+B2
043A 131F          COMF  MOVTMP+B3
043B 290A          CLRf  wreg
043C 151C          INCF  MOVTMP+B0
043D 111D          ADDWFC MOVTMP+B1
043E 111E          ADDWFC MOVTMP+B2
043F 111F          ADDWFC MOVTMP+B3

vmpos
0440 971F          btfs  MOVTMP+B3,MSB
0441 C45B          goto  vmoveok          ; if not, continue

CLR32  A          ; if so, set A=0 and continue with

0442 299C          CLRf  A+B0
0443 299D          CLRf  A+B1
0444 299E          CLRf  A+B2
0445 299F          CLRf  A+B3

MOV32  MOVVAL,MOVVBUF          ; move unless the final velocity

0446 6A5F          MOVFP  MOVVAL+B0,wreg          ; get byte of MOVVAL into w
0447 4AA8          MOVFP  wreg,MOVVBUF+B0          ; move to MOVVBUF (B0)
0448 6A60          MOVFP  MOVVAL+B1,wreg          ; get byte of MOVVAL into w
0449 4AA9          MOVFP  wreg,MOVVBUF+B1          ; move to MOVVBUF (B1)
044A 6A61          MOVFP  MOVVAL+B2,wreg          ; get byte of MOVVAL into w
044B 4AAA          MOVFP  wreg,MOVVBUF+B2          ; move to MOVVBUF (B2)
044C 6A62          MOVFP  MOVVAL+B3,wreg          ; get byte of MOVVAL into w
044D 4AAB          MOVFP  wreg,MOVVBUF+B3          ; move to MOVVBUF (B3)

; is zero.
044E 2994          clrf  MOVFLAG          ; clear MOVFLAG
044F 8D93          bcf   MOVSTAT,bit5          ; clear move in progress flag

MOV16  MOVTIME,TAU

0450 6A67          MOVFP  MOVTIME+B0,wreg          ; get byte of MOVTIME into w
0451 016E          MOVWF  TAU+B0          ; move to TAU (B0)
0452 6A68          MOVFP  MOVTIME+B1,wreg          ; get byte of MOVTIME into w
0453 016F          MOVWF  TAU+B1          ; move to TAU (B1)

TFSZ32 MOVVAL

0454 6A5F          MOVFP  MOVVAL+B0,wreg
0455 0860          IORWF  MOVVAL+B1,W
0456 0861          IORWF  MOVVAL+B2,W
0457 0862          IORWF  MOVVAL+B3,W
0458 330A          TSTFSZ wreg

0459 C45B          goto  vmoveok

045A 8E93          bcf   MOVSTAT,bit6          ; if final velocity is zero, clear
; motion status flag

vmoveok
MOV24  MOVVBUF+B1,POSITION

045B 6AA5          MOVFP  MOVVBUF+B1+B0,wreg          ; get byte of MOVVBUF+B1 into w
045C 4A55          MOVFP  wreg,POSITION+B0          ; move to POSITION (B0)
045D 6AA6          MOVFP  MOVVBUF+B1+B1,wreg          ; get byte of MOVVBUF+B1 into w
045E 4A56          MOVFP  wreg,POSITION+B1          ; move to POSITION (B1)
045F 6AA7          MOVFP  MOVVBUF+B1+B2,wreg          ; get byte of MOVVBUF+B1 into w

```


Servo Control of a DC-Brush Motor

```

0460 4A57          MOVFP   wreg,POSITION+B2      ; move to POSITION (B2)

MOV24  MOVVBUF+B0,VELOCITY

0461 6AA8          MOVFP   MOVVBUF+B0+B0,wreg      ; get byte of MOVVBUF+B0 into w
0462 4A58          MOVFP   wreg,VELOCITY+B0      ; move to VELOCITY (B0)
0463 6AA9          MOVFP   MOVVBUF+B0+B1,wreg      ; get byte of MOVVBUF+B0 into w
0464 4A59          MOVFP   wreg,VELOCITY+B1      ; move to VELOCITY (B1)
0465 6AAA          MOVFP   MOVVBUF+B0+B2,wreg      ; get byte of MOVVBUF+B0 into w
0466 4A5A          MOVFP   wreg,VELOCITY+B2      ; move to VELOCITY (B2)

0467 0002          return

;*****
;*****
; NAME:           doPosVel
;
; DESCRIPTION:    Evaluates the iterative equations for trapezoidal
;                generation
;
;                 $V(k)=V(k-1)+A,$             $P(k)=P(k-1)+V(k-1)+A/2,$ 
;
;                where abs(A)={AL,0} depending on the region of the
;                being executed.
;
doPosVel
ADD32  MOVVBUF,MOVVBUF          ; P(k-1)+V(k-1)

0468 6AA8          MOVFP   MOVVBUF+B0,wreg      ; get lowest byte of MOVVBUF into w
0469 0FA4          ADDWF   MOVVBUF+B0          ; add lowest byte of MOVVBUF, save in
046A 6AA9          MOVFP   MOVVBUF+B1,wreg      ; get 2nd byte of MOVVBUF into w
046B 11A5          ADDWFC  MOVVBUF+B1          ; add 2nd byte of MOVVBUF, save in
046C 6AAA          MOVFP   MOVVBUF+B2,wreg      ; get 3rd byte of MOVVBUF into w
046D 11A6          ADDWFC  MOVVBUF+B2          ; add 3rd byte of MOVVBUF, save in
046E 6AAB          MOVFP   MOVVBUF+B3,wreg      ; get 4th byte of MOVVBUF into w
046F 11A7          ADDWFC  MOVVBUF+B3          ; add 4th byte of MOVVBUF, save in

ADD32  A,MOVVBUF              ; V(k)=V(k-1)+A

0470 6A9C          MOVFP   A+B0,wreg          ; get lowest byte of A into w
0471 0FA8          ADDWF   MOVVBUF+B0          ; add lowest byte of MOVVBUF, save in
0472 6A9D          MOVFP   A+B1,wreg          ; get 2nd byte of A into w
0473 11A9          ADDWFC  MOVVBUF+B1          ; add 2nd byte of MOVVBUF, save in
0474 6A9E          MOVFP   A+B2,wreg          ; get 3rd byte of A into w
0475 11AA          ADDWFC  MOVVBUF+B2          ; add 3rd byte of MOVVBUF, save in
0476 6A9F          MOVFP   A+B3,wreg          ; get 4th byte of A into w
0477 11AB          ADDWFC  MOVVBUF+B3          ; add 4th byte of MOVVBUF, save in

MOVFP32 A,MOVTMP              ; compute A/2

0478 7C9C          MOVFP   A+B0,MOVTMP+B0      ; move A(B0) to MOVTMP (B0)
0479 7D9D          MOVFP   A+B1,MOVTMP+B1      ; move A(B1) to MOVTMP (B1)
047A 7E9E          MOVFP   A+B2,MOVTMP+B2      ; move A(B2) to MOVTMP (B2)
047B 7F9F          MOVFP   A+B3,MOVTMP+B3      ; move A(B3) to MOVTMP (B3)

RRC32  MOVTMP

047C 1A1F          RLCF   MOVTMP+B3,W          ; move sign into carry bit

```



Servo Control of a DC-Brush Motor

```

047D 191F          RRCF      MOVTMP+B3
047E 191E          RRCF      MOVTMP+B2
047F 191D          RRCF      MOVTMP+B1
0480 191C          RRCF      MOVTMP+B0

                ADD32     MOVTMP,MOVVPBUF          ; P(k)=P(k-1)+V(k-1)+A/2,

0481 6A1C          MOVFP     MOVTMP+B0,wreg ; get lowest byte of MOVTMP into w
0482 0FA4          ADDWF    MOVVPBUF+B0    ; add lowest byte of MOVVPBUF, save in
0483 6A1D          MOVFP     MOVTMP+B1,wreg ; get 2nd byte of MOVTMP into w
0484 11A5          ADDWFC   MOVVPBUF+B1    ; add 2nd byte of MOVVPBUF, save in
0485 6A1E          MOVFP     MOVTMP+B2,wreg ; get 3rd byte of MOVTMP into w
0486 11A6          ADDWFC   MOVVPBUF+B2    ; add 3rd byte of MOVVPBUF, save in
0487 6A1F          MOVFP     MOVTMP+B3,wreg ; get 4th byte of MOVTMP into w
0488 11A7          ADDWFC   MOVVPBUF+B3    ; add 4th byte of MOVVPBUF, save in )

0489 0002          return

                ,*****
                ,*****
; NAME: undoPosVel
;
; ; DESCRIPTION: Backward iteration of the equations for trapezoidal
; ; generation
; ;
; ; V(k-1)=V(k)-A,          P(k-1)=P(k)-V(k-1)-A/2,
; ;
; ; where abs(A)={AL,0} depending on the region of the
; ; being executed. This routine is used to reverse a
; ; to be made beyond a decision point.
; ;
                undoPosVel

SUB32  A,MOVVPBUF          ; V(k-1)=V(k)-A

048A 6A9C          MOVFP     A+B0,wreg ; get lowest byte of A into w
048B 05A8          SUBWF    MOVVPBUF+B0    ; sub lowest byte of MOVVPBUF, save in
048C 6A9D          MOVFP     A+B1,wreg ; get 2nd byte of A into w
048D 03A9          SUBWFB   MOVVPBUF+B1    ; sub 2nd byte of MOVVPBUF, save in
048E 6A9E          MOVFP     A+B2,wreg ; get 3rd byte of A into w
048F 03AA          SUBWFB   MOVVPBUF+B2    ; sub 3rd byte of MOVVPBUF, save in
0490 6A9F          MOVFP     A+B3,wreg ; get 4th byte of A into w
0491 03AB          SUBWFB   MOVVPBUF+B3    ; sub 4th byte of MOVVPBUF, save in

SUB32  MOVVPBUF,MOVVPBUF          ; P(k)-V(k-1)

0492 6AA8          MOVFP     MOVVPBUF+B0,wreg ; get lowest byte of MOVVPBUF into w
0493 05A4          SUBWF    MOVVPBUF+B0    ; sub lowest byte of MOVVPBUF, save in
0494 6AA9          MOVFP     MOVVPBUF+B1,wreg ; get 2nd byte of MOVVPBUF into w
0495 03A5          SUBWFB   MOVVPBUF+B1    ; sub 2nd byte of MOVVPBUF, save in
0496 6AAA          MOVFP     MOVVPBUF+B2,wreg ; get 3rd byte of MOVVPBUF into w
0497 03A6          SUBWFB   MOVVPBUF+B2    ; sub 3rd byte of MOVVPBUF, save in
0498 6AAB          MOVFP     MOVVPBUF+B3,wreg ; get 4th byte of MOVVPBUF into w
0499 03A7          SUBWFB   MOVVPBUF+B3    ; sub 4th byte of MOVVPBUF, save in

                MOVFP32  A,MOVTMP          ; compute A/2

049A 7C9C          MOVFP     A+B0,MOVTMP+B0 ; move A(B0) to MOVTMP(B0)

```

Servo Control of a DC-Brush Motor

```

049B 7D9D          MOVFP  A+B1,MOVTMP+B1 ; move A(B1) to MOVTMP (B1)
049C 7E9E          MOVFP  A+B2,MOVTMP+B2 ; move A(B2) to MOVTMP (B2)
049D 7F9F          MOVFP  A+B3,MOVTMP+B3 ; move A(B3) to MOVTMP (B3)

RRC32  MOVTMP

049E 1A1F          RLCF   MOVTMP+B3,W      ; move sign into carry bit
049F 191F          RRCF   MOVTMP+B3
04A0 191E          RRCF   MOVTMP+B2
04A1 191D          RRCF   MOVTMP+B1
04A2 191C          RRCF   MOVTMP+B0

SUB32  MOVTMP,MOVBUF          ; P(k-1)=P(k)-V(k-1)-A/2,

04A3 6A1C          MOVFP  MOVTMP+B0,wreg ; get lowest byte of MOVTMP into w
04A4 05A4          SUBWF  MOVBUF+B0      ; sub lowest byte of MOVBUF, save in
04A5 6A1D          MOVFP  MOVTMP+B1,wreg ; get 2nd byte of MOVTMP into w
04A6 03A5          SUBWFB MOVBUF+B1     ; sub 2nd byte of MOVBUF, save in
04A7 6A1E          MOVFP  MOVTMP+B2,wreg ; get 3rd byte of MOVTMP into w
04A8 03A6          SUBWFB MOVBUF+B2     ; sub 3rd byte of MOVBUF, save in
04A9 6A1F          MOVFP  MOVTMP+B3,wreg ; get 4th byte of MOVTMP into w
04AA 03A7          SUBWFB MOVBUF+B3     ; sub 4th byte of MOVBUF, save in

04AB 0002          return

;*****
if _SERVO_PID
include "pid.asm"          ; PID Algorithm
;*****
; PID Servo Implementation
;
; Implement Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
;
;*****

;*****
; NAME:          doServo
;
; DESCRIPTION:   Performs the servo loop calculations.
;

doServo:

MOV16  POSERROR,U0          ; save new position error in

04AC 6A79          MOVFP  POSERROR+B0,wreg ; get byte of POSERROR into w
04AD 0184          MOVWF  U0+B0            ; move to U0 (B0)
04AE 6A7A          MOVFP  POSERROR+B1,wreg ; get byte of POSERROR into w
04AF 0185          MOVWF  U0+B1            ; move to U0 (B1)

LOADAB  U0,KP              ; compute KP*U0

04B0 7C84          MOVFP  U0+B0,AARG+B0    ; load lo byte of U0 to AARG
04B1 7D85          MOVFP  U0+B1,AARG+B1    ; load hi byte of U0 to AARG
04B2 7E26          MOVFP  KP+B0,BARG+B0 ; load lo byte of KP to BARG
04B3 7F27          MOVFP  KP+B1,BARG+B1 ; load hi byte of KP to BARG

04B4 E12B          call   Dmult

```



Servo Control of a DC-Brush Motor

```

MOVFP32 DPX, Y ; Y=KP*U0

04B5 5880 MOVFP DPX+B0, Y+B0 ; move DPX (B0) to Y (B0)
04B6 5981 MOVFP DPX+B1, Y+B1 ; move DPX (B1) to Y (B1)
04B7 5A82 MOVFP DPX+B2, Y+B2 ; move DPX (B2) to Y (B2)
04B8 5B83 MOVFP DPX+B3, Y+B3 ; move DPX (B3) to Y (B3)

04B9 290A clrf wreg ; if previous output saturated, do
04BA 3295 cpfsgt SATFLAG ; not accumulate integrator
04BB E552 call doIntegral

LOADAB INTEGRAL, KI ; compute KI*INTEGRAL

04BC 7C96 MOVFP INTEGRAL+B0, AARG+B0 ; load lo byte of INTEGRAL to AARG
04BD 7D97 MOVFP INTEGRAL+B1, AARG+B1 ; load hi byte of INTEGRAL to AARG
04BE 7E2A MOVFP KI+B0, BARG+B0 ; load lo byte of KI to BARG
04BF 7F2B MOVFP KI+B1, BARG+B1 ; load hi byte of KI to BARG

04C0 E12B call Dmult
ADD32 DPX, Y ; Y=KP*U0+KI*INTEGRAL

04C1 6A18 MOVFP DPX+B0, wreg ; get lowest byte of DPX into w
04C2 0F80 ADDWF Y+B0 ; add lowest byte of Y, save in Y (B0)
04C3 6A19 MOVFP DPX+B1, wreg ; get 2nd byte of DPX into w
04C4 1181 ADDWFC Y+B1 ; add 2nd byte of Y, save in Y (B1)
04C5 6A1A MOVFP DPX+B2, wreg ; get 3rd byte of DPX into w
04C6 1182 ADDWFC Y+B2 ; add 3rd byte of Y, save in Y (B2)
04C7 6A1B MOVFP DPX+B3, wreg ; get 4th byte of DPX into w
04C8 1183 ADDWFC Y+B3 ; add 4th byte of Y, save in Y (B3)

MOVFP16 U0, AARG ; compute KV*(U0-U1)

04C9 7C84 MOVFP U0+B0, AARG+B0 ; move U0 (B0) to AARG (B0)
04CA 7D85 MOVFP U0+B1, AARG+B1 ; move U0 (B1) to AARG (B1)

SUB16 U1, AARG

04CB 6A86 MOVFP U1+B0, wreg ; get lowest byte of U1 into w
04CC 051C SUBWF AARG+B0 ; sub lowest byte of AARG, save in
04CD 6A87 MOVFP U1+B1, wreg ; get 2nd byte of U1 into w
04CE 031D SUBWFB AARG+B1 ; sub 2nd byte of AARG, save in

MOVFP16 KV, BARG

04CF 7E28 MOVFP KV+B0, BARG+B0 ; move KV (B0) to BARG (B0)
04D0 7F29 MOVFP KV+B1, BARG+B1 ; move KV (B1) to BARG (B1)

04D1 E12B call Dmult
ADD32 DPX, Y ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)

04D2 6A18 MOVFP DPX+B0, wreg ; get lowest byte of DPX into w
04D3 0F80 ADDWF Y+B0 ; add lowest byte of Y, save in Y (B0)
04D4 6A19 MOVFP DPX+B1, wreg ; get 2nd byte of DPX into w
04D5 1181 ADDWFC Y+B1 ; add 2nd byte of Y, save in Y (B1)
04D6 6A1A MOVFP DPX+B2, wreg ; get 3rd byte of DPX into w
04D7 1182 ADDWFC Y+B2 ; add 3rd byte of Y, save in Y (B2)
04D8 6A1B MOVFP DPX+B3, wreg ; get 4th byte of DPX into w
04D9 1183 ADDWFC Y+B3 ; add 4th byte of Y, save in Y (B3)

```

Servo Control of a DC-Brush Motor

```

MOV16    U0,U1                ; push errors into U(k-1)

04DA 6A84                MOVFP    U0+B0,wreg        ; get byte of U0 into w
04DB 0186                MOVWF   U1+B0            ; move to U1(B0)
04DC 6A85                MOVFP    U0+B1,wreg        ; get byte of U0 into w
04DD 0187                MOVWF   U1+B1            ; move to U1(B1)

04DE 290A                clr     wreg
04DF 32B9                cpfsgt  SHIFTNUM
04E0 C4E9                goto   grabok
04E1 78B9                movfp  SHIFTNUM,TMP

grabloop
RLC32    Y

04E2 8804                BCF     _carry
04E3 1B80                RLCF   Y+B0
04E4 1B81                RLCF   Y+B1
04E5 1B82                RLCF   Y+B2
04E6 1B83                RLCF   Y+B3

04E7 1718                decfsz  TMP
04E8 C4E2                goto   grabloop

grabok

04E9 2995                clr     SATFLAG
04EA 9F83                btfs   Y+B3,MSB          ; saturate to middle 16 bits,
04EB C4F9                goto   negs              ; keeping top 10 bits for pwldcL
                                ; and pwldcL
                                ; check if Y >= 2**23
04EC 6A82                movfp  Y+B2,wreg
04ED B580                andlw  0x80
04EE 0983                iorwf  Y+B3
04EF 290A                clr     wreg
04F0 3283                cpfsgt  Y+B3
04F1 C505                goto   zero6bits        ; if not, zero 6 bits

04F2 1595                incf   SATFLAG          ; if so, set Y=0x007FFFFFFF
04F3 2983                clr     Y+B3            ; clear for debug purposes
04F4 B07F                movlw  0x7F
04F5 4A82                movfp  wreg,Y+B2
04F6 2B81                setf   Y+B1
04F7 2B80                setf   Y+B0
04F8 C505                goto   zero6bits

negs

04F9 6A82                movfp  Y+B2,wreg        ; check if Y <= -2**23
04FA B37F                iorlw  0x7F
04FB 0B83                andwf  Y+B3
04FC 2B0A                setf   wreg
04FD 3083                cpfslt  Y+B3
04FE C505                goto   zero6bits        ; if not, zero 6 bits

04FF 2B95                setf   SATFLAG          ; if so, set Y = 0xFF800000
0500 2B83                setf   Y+B3
0501 2982                clr     Y+B2
0502 8782                bsf    Y+B2,MSB
0503 2981                clr     Y+B1
0504 2980                clr     Y+B0

zero6bits
MOV24    Y+B1,YPWM+B0        ; move Y to YPWM and zero 6 bits

0505 6A81                MOVFP    Y+B1+B0,wreg    ; get byte of Y+B1 into w
0506 4A88                MOVFP    wreg,YPWM+B0+B0 ; move to YPWM+B0 (B0)
0507 6A82                MOVFP    Y+B1+B1,wreg    ; get byte of Y+B1 into w
0508 4A89                MOVFP    wreg,YPWM+B0+B1 ; move to YPWM+B0 (B1)
0509 6A83                MOVFP    Y+B1+B2,wreg    ; get byte of Y+B1 into w
050A 4A8A                MOVFP    wreg,YPWM+B0+B2 ; move to YPWM+B0 (B2)

```

Servo Control of a DC-Brush Motor

```

doTorque                                     ; entry point for torque mode

050B B0C0          movlw  0xC0
050C 0B88          andwf  YPWM+B0

050D 9F89          btfsc  YPWM+B1,MSB
050E C516          goto   tmlimit

050F 9692          btfss  EXTSTAT,bit6
0510 C51C          goto   mplimitok

CLR32  YPWM

0511 2988          CLRF   YPWM+B0
0512 2989          CLRF   YPWM+B1
0513 298A          CLRF   YPWM+B2
0514 298B          CLRF   YPWM+B3

0515 C51C          goto   mplimitok

tmlimit
0516 9592          btfss  EXTSTAT,bit5
0517 C51C          goto   mplimitok

CLR32  YPWM

0518 2988          CLRF   YPWM+B0
0519 2989          CLRF   YPWM+B1
051A 298A          CLRF   YPWM+B2
051B 298B          CLRF   YPWM+B3

mplimitok

051C B07F          movlw  PWDCH_INIT      ; adjustment from bipolar to unipolar
051D 4A19          movpf  wreg,TMP+B1      ; for 50% duty cycle
051E B0C0          movlw  PWDCL_INIT
051F 4A18          movpf  wreg,TMP+B0
                      ADDL6   TMP,YPWM

0520 6A18          MOVFPP TMP+B0,wreg      ; get lowest byte of TMP into w
0521 0F88          ADDWF  YPWM+B0          ; add lowest byte of YPWM, save in YPWM(B0)
0522 6A19          MOVFPP TMP+B1,wreg      ; get 2nd byte of TMP into w
0523 1189          ADDWFC YPWM+B1          ; add 2nd byte of YPWM, save in YPWM(B1)

0524 2919          clrf   TMP+B1          ; correct by 1 LSB
0525 B040          movlw  0x40            ; add one to bit5 of pwldcL
0526 4A18          movpf  wreg,TMP+B0
                      ADDL6   TMP,YPWM

0527 6A18          MOVFPP TMP+B0,wreg      ; get lowest byte of TMP into w
0528 0F88          ADDWF  YPWM+B0          ; add lowest byte of YPWM, save in YPWM(B0)
0529 6A19          MOVFPP TMP+B1,wreg      ; get 2nd byte of TMP into w
052A 1189          ADDWFC YPWM+B1          ; add 2nd byte of YPWM, save in YPWM(B1)

testmax
052B 291A          clrf   TMP+B2          ; check pwm maximum limit
052C 298A          clrf   YPWM+B2          ; LMD18200 must have a minimum pulse
052D 298B          clrf   YPWM+B3          ; so duty cycle must not be 0 or 100%
                      MOVFPP16 YPWM_MAX,TMP

052E 788E          MOVFPP YPWM_MAX+B0,TMP+B0; move YPWM_MAX(B0) to TMP(B0)
052F 798F          MOVFPP YPWM_MAX+B1,TMP+B1; move YPWM_MAX(B1) to TMP(B1)

```

Servo Control of a DC-Brush Motor

```

SUB24  YPWM,TMP

0530 6A88      MOVF  YPWM+B0,wreg    ; get lowest byte of YPWM into w
0531 0518      SUBWF TMP+B0          ; sub lowest byte of TMP, save in TMP(B0)
0532 6A89      MOVF  YPWM+B1,wreg    ; get 2nd byte of YPWM into w
0533 0319      SUBWFB TMP+B1          ; sub 2nd byte of TMP, save in TMP(B1)
0534 6A8A      MOVF  YPWM+B2,wreg    ; get 3rd byte of YPWM into w
0535 031A      SUBWFB TMP+B2          ; sub 3rd byte of TMP, save in TMP(B2)

0536 971A      btfss  TMP+B2,MSB
0537 C53D      goto   testmin

MOV16  YPWMAX,YPWM          ; saturate to max

0538 6A8E      MOVF  YPWMAX+B0,wreg    ; get byte of YPWMAX into w
0539 0188      MOVWF YPWM+B0          ; move to YPWM(B0)
053A 6A8F      MOVF  YPWMAX+B1,wreg    ; get byte of YPWMAX into w
053B 0189      MOVWF YPWM+B1          ; move to YPWM(B1)

053C C54E      goto   limitok
testmin
053D 291A      clrf  TMP+B2          ;check pwm minimum limit
053E 298A      clrf  YPWM+B2
053F 298B      clrf  YPWM+B3
MOVFP16 YPWMIN,TMP

0540 788C      MOVF  YPWMIN+B0,TMP+B0 ; move YPWMIN(B0) to TMP(B0)
0541 798D      MOVF  YPWMIN+B1,TMP+B1 ; move YPWMIN(B1) to TMP(B1)

SUB24  YPWM,TMP

0542 6A88      MOVF  YPWM+B0,wreg    ; get lowest byte of YPWM into w
0543 0518      SUBWF TMP+B0          ; sub lowest byte of TMP, save in TMP(B0)
0544 6A89      MOVF  YPWM+B1,wreg    ; get 2nd byte of YPWM into w
0545 0319      SUBWFB TMP+B1          ; sub 2nd byte of TMP, save in TMP(B1)
0546 6A8A      MOVF  YPWM+B2,wreg    ; get 3rd byte of YPWM into w
0547 031A      SUBWFB TMP+B2          ; sub 3rd byte of TMP, save in TMP(B2)

0548 9F1A      btfs  TMP+B2,MSB
0549 C54E      goto   limitok
MOV16  YPWMIN,YPWM          ; saturate to min

054A 6A8C      MOVF  YPWMIN+B0,wreg    ; get byte of YPWMIN into w
054B 0188      MOVWF YPWM+B0          ; move to YPWM(B0)
054C 6A8D      MOVF  YPWMIN+B1,wreg    ; get byte of YPWMIN into w
054D 0189      MOVWF YPWM+B1          ; move to YPWM(B1)

limitok
054E B803      movlb  bank3          ; set new duty cycle
054F 7088      movfp  YPWM+B0,pwldcl
0550 7289      movfp  YPWM+B1,pwldch

0551 0002      return
;*****
;*****
; NAME:          doIntegral
;
; DESCRIPTION:   Evaluates the integral for the servo calculations.
;
;               doIntegral
;
;               ADD16  U0,INTEGRAL    ; do integral

0552 6A84      MOVF  U0+B0,wreg    ; get lowest byte of U0 into w

```

Servo Control of a DC-Brush Motor

```
0553 0F96          ADDWF  INTEGRAL+B0    ; add lowest byte of INTEGRAL, save in
0554 6A85          MOVFP  U0+B1,wreg      ; get 2nd byte of U0 into w
0555 1197          ADDWFC INTEGRAL+B1    ; add 2nd byte of INTEGRAL, save in

0556 0002          return

;*****

endif

if _SERIAL_IO
include "serial.asm"          ; Serial I/O Routines
;*****
;
;                               Serial I/O & Utility Functions
;
;*****

;*****
; NAME:          IdleFunction

; DESCRIPTION:   This routine will perform work while doing waits in
;               serial I/O functions.
;

IdleFunction

0557 0004          CLRWDT
0558 0002          return

;*****

;*****
; NAME:          DoCommand

; DESCRIPTION:   Search command table for command and execute it.
;

DoCommand

0559 B059          movlw  (CMD_TABLE & 0xff)    ; CMD_TABLE LSB
055A 4A0D          movpf  wreg,tblptrl
055B B007          movlw  page CMD_TABLE        ; CMD_TABLE MSB
055C 4A0E          movpf  wreg,tblptrh

055D AB3A          tablrd 1,1,CMDTEMP
tryNextCmd

055E A93A          tablrd 0,1,CMDTEMP          ; read entry from table
055F A23B          tlrld 1,CMDPTRH
0560 A93C          tablrd 0,1,CMDPTRL

0561 6A3A          movfp  CMDTEMP,wreg
0562 30C1          cpfslt ZERO

0563 C56E          goto   noCommand          ; error if end of table

0564 3139          cpfseq CMDCHAR
0565 C55E          goto   tryNextCmd

0566 E679          call   PutChar          ; echo command

0567 633B          movfp  CMDPTRH,pclath      ; indirect jump to command routine
0568 623C          movfp  CMDPTRL,pcl
0569 0000          NOP
```


Servo Control of a DC-Brush Motor

```

cmdFinish
056A E679      call    PutChar          ; send response character from
                                           ; command routine followed by CR
056B B00D      movlw   CR
056C E679      call    PutChar
056D C124      goto    PollingLoop

                noCommand
056E B03F      movlw   CMD_BAD          ; send error character
056F C56A      goto    cmdFinish

;*****
;*****
; NAME:        do_null
;
; DESCRIPTION: The do nothing command used to determine if the chip is
;              working. Initiated by a carriage return.

do_null
0570 B021      movlw   CMD_OK
0571 C56A      goto    cmdFinish

;*****
;*****
; NAME:        do_move
;
; DESCRIPTION: Commands the axis to move to a new position or velocity.
;              Position data is relative, and in encoder counts. Velocity
;              data is absolute, and in encoder counts/sample time multi-
;              plied by 256. All moves are performed by the controller such
;              that velocity and acceleration limits set into parameter
;              memory will not be violated. All move commands are kept in a
;              one deep FIFO buffer. The command in the buffer is executed
;              as soon as the currently executed command is complete.
;
;
; ARGUMENTS:   M [800000,7FFFFFF]

do_move
                if    DECIO

0572 E6CC      call    GetDecVal

                else

                call    GetVal

                endif

0573 9F93      btfscc MOVSTAT,bit7          ; test if buffer available
0574 C57E      goto    bufoverflow

                MOV24  VALBUF,NMOVVAL          ; if so, accept value into NMOVVAL

0575 6A31      MOVFP  VALBUF+B0,wreg          ; get byte of VALBUF into w
0576 4A5B      MOVFP  wreg,NMOVVAL+B0          ; move to NMOVVAL(B0)
0577 6A32      MOVFP  VALBUF+B1,wreg          ; get byte of VALBUF into w
0578 4A5C      MOVFP  wreg,NMOVVAL+B1          ; move to NMOVVAL(B1)
0579 6A33      MOVFP  VALBUF+B2,wreg          ; get byte of VALBUF into w
057A 4A5D      MOVFP  wreg,NMOVVAL+B2          ; move to NMOVVAL(B2)

057B 8793      bsf    MOVSTAT,bit7          ; set buffer full flag

057C B021      movlw   CMD_OK

```

Servo Control of a DC-Brush Motor

```

057D C56A          goto    cmdFinish

bufoverflow
057E B03F          movlw   CMD_BAD ; else, return error
057F C56A          goto    cmdFinish

;*****
;*****
; NAME:            do_mode
;
; DESCRIPTION:     An argument of "P" will cause all subsequent move commands
;                  to be incremental position moves. A "V" argument will cause
;                  all subsequent moves to be absolute velocity moves.
;
; ARGUMENTS:      O [P,V]
;

do_mode

0580 E557          call    IdleFunction ; get single character loop
0581 E681          call    GetChk
0582 31C2          cpfseq  ONE
0583 C580          goto    do_mode
0584 E676          call    GetChar
0585 4A4D          movpf  wreg,STRVALL
0586 2991          clrf   MODETYPE ; MODETYPE=0 for position moves

testP
0587 B050          movlw  'P' ; position moves for type P
0588 314D          cpfseq STRVALL
0589 C58B          goto  testV
058A C598          goto  modeok

testV
058B B056          movlw  'V' ; velocity moves for type V
058C 314D          cpfseq STRVALL
058D C590          goto  testT
058E 1591          incf  MODETYPE ; MODETYPE=1 for velocity moves
058F C598          goto  modeok

testT
0590 B054          movlw  'T' ; TORQUE Moves for type 'T'
0591 314D          cpfseq STRVALL
0592 C596          goto  modeerror
0593 2B91          setf  MODETYPE ; MODETYPE=-1 for torque moves
0594 2990          clrf  SERVOFLAG ; disable servo
0595 C598          goto  modeok

modeerror
0596 B03F          movlw  CMD_BAD ; mode error
0597 C56A          goto  cmdFinish

modeok
0598 6A4D          movfp  STRVALL,wreg ; echo type character
0599 E679          call  PutChar

059A B021          movlw  CMD_OK
059B C56A          goto  cmdFinish

;*****
;*****
; NAME:            do_setparameter
;
; DESCRIPTION:     Sets controller parameters to the value given.
;
; Parameter      #           Range
;
; VL=velocity limit    0       [0,7FFFFFF]
; AL=acceleration limit 1       [0,7FFFFFF]
;
; KP=proportional gain 2       [8000,7FFF]
; KP=velocity gain     3       [8000,7FFF]

```

Servo Control of a DC-Brush Motor

```

;      KP=integral gain      4      [8000,7FFF]
;
;      IM=integrator mode    5      [0,3]
;
;      FV=velocity FF        6      [8000,7FFF] : Not Imple
;      FA=acceleration FF    7      [8000,7FFF] : Not Imple
;
;
; ARGUMENTS:  S [0,FF] [800000,7FFFFFFF]
;
do_setparameter

059C E669          call    GetPar          ; get parameter number

059D B008          movlw   NUMPAR          ; check if in range [0,NUMPAR]
059E 3031          cpfslt  VALBUF+B0
059F C5C1          goto    Serror

05A0 B07C          movlw   (PAR_TABLE & 0xff) ; PAR_TABLE LSB
05A1 4A0D          movpf   wreg,tblptrl
05A2 B007          movlw   page PAR_TABLE      ; PAR_TABLE MSB
05A3 4A0E          movpf   wreg,tblptrh

05A4 AB3D          tablrd  1,1,PARTEMP

setNextPar

05A5 A23D          tlrld  1,PARTEMP          ; read entry from table
05A6 A93E          tablrd  0,1,PARLEN
05A7 A93F          tablrd  0,1,PARPTR

05A8 B008          movlw   NUMPAR          ; error if end of table
05A9 303D          cpfslt  PARTEMP
05AA C5C1          goto    Serror

05AB 6A3D          movfp  PARTEMP,wreg
05AC 3131          cpfseq  VALBUF+B0
05AD C5A5          goto    setNextPar

05AE 6A3F          movfp  PARPTR,wreg          ; pointer to parameter in fsr1
05AF 690A          movfp  wreg,fsr1

if DECIO          ; get new value in VALBUF

05B0 E6CC          call    GetDecVal

else

call    GetVal

endif

05B1 B031          movlw   VALBUF          ; pointer to VALBUF in fsr0
05B2 610A          movfp  wreg,fsr0
AUTOINC          ; set autoincrement

05B3 8404          BSF    _fs0
05B4 8D04          BCF    _fs1
05B5 8604          BSF    _fs2
05B6 8F04          BCF    _fs3

setGetMore

05B7 6800          movfp  indf0,indf1      ; move new value to parameter
05B8 073E          decf  PARLEN
05B9 333E          tstfsz PARLEN

```

Servo Control of a DC-Brush Motor

```

05BA C5B7          goto    setGetMore

                    AUTONO          ; no autoincrement

05BB 8404          BSF     _fs0
05BC 8504          BSF     _fs1
05BD 8604          BSF     _fs2
05BE 8704          BSF     _fs3

05BF B021          movlw   CMD_OK
05C0 C56A          goto    cmdFinish

Error

05C1 B03F          movlw   CMD_BAD
05C2 C56A          goto    cmdFinish

;*****
;*****
; NAME:            do_readparameter
;
; DESCRIPTION:     Returns the present value of a parameter.
;
; ARGUMENTS:      R [0,FF]
;
; RETURNS:        The present value of the requested parameter is returned.

do_readparameter

05C3 E669          call    GetPar          ; get parameter number

05C4 B008          movlw   NUMPAR          ; check if in range [0,NUMPAR]
05C5 3031          cpfslt VALBUF+B0
05C6 C5EB          goto    Rerror

05C7 B07C          movlw   (PAR_TABLE & 0xff) ; PAR_TABLE LSB
05C8 4A0D          movpf   wreg,tblptrl
05C9 B007          movlw   page PAR_TABLE   ; PAR_TABLE MSB
05CA 4A0E          movpf   wreg,tblptrh

05CB AB3D          tablrd  1,1,PARTEMP

readNextPar

05CC A23D          tlrld  1,PARTEMP          ; read entry from table
05CD A93E          tablrd  0,1,PARLEN
05CE A93F          tablrd  0,1,PARPTR

05CF B008          movlw   NUMPAR          ; error if end of table
05D0 303D          cpfslt PARTEMP
05D1 C5EB          goto    Rerror

05D2 6A3D          movfp  PARTEMP,wreg
05D3 3131          cpfseq  VALBUF+B0
05D4 C5CC          goto    readNextPar

05D5 6A3F          movfp  PARPTR,wreg          ; pointer to parameter in fsr1
05D6 690A          movfp  wreg,fsr1

05D7 B031          movlw  VALBUF          ; pointer to VALBUF in fsr1
05D8 610A          movfp  wreg,fsr0
                    AUTOINC          ; set autoincrement

05D9 8404          BSF     _fs0
05DA 8D04          BCF     _fs1
05DB 8604          BSF     _fs2

```

Servo Control of a DC-Brush Motor

```

05DC 8F04          BCF      _fs3

                CLR24   VALBUF          ; clear old VALBUF

05DD 2931          CLRF    VALBUF+B0
05DE 2932          CLRF    VALBUF+B1
05DF 2933          CLRF    VALBUF+B2

                readGetMore
05E0 6008          movfp   indf1,indf0   ; read parameter into VALBUF
05E1 073E          decf    PARLEN
05E2 333E          tstfsz  PARLEN
05E3 C5E0          goto    readGetMore

                AUTONO          ; no autoincrement

05E4 8404          BSF     _fs0
05E5 8504          BSF     _fs1
05E6 8604          BSF     _fs2
05E7 8704          BSF     _fs3

                if      DECIO          ; send parameter value
05E8 E728          call    PutDecVal

                else
                call    PutVal

                endif

05E9 B021          movlw   CMD_OK
05EA C56A          goto    cmdFinish

                Rerror
05EB B03F          movlw   CMD_BAD
05EC C56A          goto    cmdFinish

;*****
;*****
; NAME: do_shutter
;
; DESCRIPTION: Returns the time (in sample time counts [0,FFFF]) since the
;              start of the present move and captures the commanded and
;              measured values of position and velocity at the time of the
;              command.
;
;
; ARGUMENTS:   C
;
; RETURNS:    The time since the start of the present move is returned.
;
do_shutter

                MOV24   POSITION,CPOSITION   ; capture commanded position

05ED 6A55          MOVFP   POSITION+B0,wreg   ; get byte of POSITION into w
05EE 4A40          MOVFP   wreg,CPOSITION+B0   ; move to CPOSITION(B0)
05EF 6A56          MOVFP   POSITION+B1,wreg   ; get byte of POSITION into w
05F0 4A41          MOVFP   wreg,CPOSITION+B1   ; move to CPOSITION(B1)
05F1 6A57          MOVFP   POSITION+B2,wreg   ; get byte of POSITION into w

```

Servo Control of a DC-Brush Motor

```

05F2 4A42          MOVFP   wreg,CPOSITION+B2      ; move to CPOSITION (B2)

MOV24 VELOCITY,CVELOCITY      ; capture commanded velocity

05F3 6A58          MOVFP   VELOCITY+B0,wreg      ; get byte of VELOCITY into w
05F4 4A43          MOVFP   wreg,CVELOCITY+B0    ; move to CVELOCITY (B0)
05F5 6A59          MOVFP   VELOCITY+B1,wreg    ; get byte of VELOCITY into w
05F6 4A44          MOVFP   wreg,CVELOCITY+B1    ; move to CVELOCITY (B1)
05F7 6A5A          MOVFP   VELOCITY+B2,wreg    ; get byte of VELOCITY into w
05F8 4A45          MOVFP   wreg,CVELOCITY+B2    ; move to CVELOCITY (B2)

MOV24 MPOSITION,CMPOSITION    ; capture measured position

05F9 6A72          MOVFP   MPOSITION+B0,wreg     ; get byte of MPOSITION into w
05FA 4A46          MOVFP   wreg,CMPOSITION+B0    ; move to CMPOSITION (B0)
05FB 6A73          MOVFP   MPOSITION+B1,wreg    ; get byte of MPOSITION into w
05FC 4A47          MOVFP   wreg,CMPOSITION+B1    ; move to CMPOSITION (B1)
05FD 6A74          MOVFP   MPOSITION+B2,wreg    ; get byte of MPOSITION into w
05FE 4A48          MOVFP   wreg,CMPOSITION+B2    ; move to CMPOSITION (B2)

MOV24 MVELOCITY,CMVELOCITY    ; capture measured velocity

05FF 6A75          MOVFP   MVELOCITY+B0,wreg     ; get byte of MVELOCITY into w
0600 4A49          MOVFP   wreg,CMVELOCITY+B0    ; move to CMVELOCITY (B0)
0601 6A76          MOVFP   MVELOCITY+B1,wreg    ; get byte of MVELOCITY into w
0602 4A4A          MOVFP   wreg,CMVELOCITY+B1    ; move to CMVELOCITY (B1)
0603 6A77          MOVFP   MVELOCITY+B2,wreg    ; get byte of MVELOCITY into w
0604 4A4B          MOVFP   wreg,CMVELOCITY+B2    ; move to CMVELOCITY (B2)

0605 2933          clrf   VALBUF+B2
MOV16 MOVTIME,VALBUF          ; capture move time, move to VALBUF

0606 6A67          MOVFP   MOVTIME+B0,wreg     ; get byte of MOVTIME into w
0607 0131          MOVFP   VALBUF+B0              ; move to VALBUF (B0)
0608 6A68          MOVFP   MOVTIME+B1,wreg     ; get byte of MOVTIME into w
0609 0132          MOVFP   VALBUF+B1              ; move to VALBUF (B1)

if DECIO
060A E728          call   PutDecVal
else
call   PutVal
endif

060B B021          movlw  CMD_OK
060C C56A          goto  cmdFinish

;*****
;*****
; NAME: do_readcomposition
;
; DESCRIPTION: Returns the commanded position count which was captured
;              during the last shutter command.
;
; ARGUMENTS:   P
;
; RETURNS:    The last captured position count is returned. [800000,7FFFFFF]
;

```

Servo Control of a DC-Brush Motor

```
do_readcomposition
MOV24  CPOSITION,VALBUF          ; move CPOSITION to VALBUF
060D  6A40                      MOVFP  CPOSITION+B0,wreg      ; get byte of CPOSITION into w
060E  4A31                      MOVFP  wreg,VALBUF+B0        ; move to VALBUF (B0)
060F  6A41                      MOVFP  CPOSITION+B1,wreg      ; get byte of CPOSITION into w
0610  4A32                      MOVFP  wreg,VALBUF+B1        ; move to VALBUF (B1)
0611  6A42                      MOVFP  CPOSITION+B2,wreg      ; get byte of CPOSITION into w
0612  4A33                      MOVFP  wreg,VALBUF+B2        ; move to VALBUF (B2)

if  DECIO

0613  E728                      call   PutDecVal

else

call   PutVal

endif

0614  B021                      movlw  CMD_OK
0615  C56A                      goto   cmdFinish

;*****
;*****
; NAME: do_readcomvelocity
;
; DESCRIPTION: Returns the commanded velocity multiplied by 256 which was
;              captured during the last shutter command.
;
; ARGUMENTS:   V
;
; RETURNS:    The last captured commanded velocity times 256 is returned.
;              [800000,7FFFFF]
;

do_readcomvelocity
MOV24  CVELOCITY,VALBUF          ; move commanded velocity to VALBUF
0616  6A43                      MOVFP  CVELOCITY+B0,wreg     ; get byte of CVELOCITY into w
0617  4A31                      MOVFP  wreg,VALBUF+B0        ; move to VALBUF (B0)
0618  6A44                      MOVFP  CVELOCITY+B1,wreg     ; get byte of CVELOCITY into w
0619  4A32                      MOVFP  wreg,VALBUF+B1        ; move to VALBUF (B1)
061A  6A45                      MOVFP  CVELOCITY+B2,wreg     ; get byte of CVELOCITY into w
061B  4A33                      MOVFP  wreg,VALBUF+B2        ; move to VALBUF (B2)

if  DECIO

061C  E728                      call   PutDecVal

else

call   PutVal

endif

061D  B021                      movlw  CMD_OK
061E  C56A                      goto   cmdFinish

;*****
;*****
```

Servo Control of a DC-Brush Motor

```
; NAME: do_readactposition
;
; DESCRIPTION: Returns the measured position count which was captured
;              during the last shutter command.
;
; ARGUMENTS:   p
;
; RETURNS:     The last captured measured position count is returned.
;              [800000,7FFFFFFF]
;

do_readactposition

MOV24  CMPOSITION, VALBUF          ; move measured position to

061F 6A46      MOVFP  CMPOSITION+B0,wreg      ; get byte of CMPOSITION into w
0620 4A31      MOVFP  wreg, VALBUF+B0        ; move to VALBUF (B0)
0621 6A47      MOVFP  CMPOSITION+B1,wreg      ; get byte of CMPOSITION into w
0622 4A32      MOVFP  wreg, VALBUF+B1        ; move to VALBUF (B1)
0623 6A48      MOVFP  CMPOSITION+B2,wreg      ; get byte of CMPOSITION into w
0624 4A33      MOVFP  wreg, VALBUF+B2        ; move to VALBUF (B2)

;
;
; if DECIO
0625 E728      call   PutDecVal
;
; else
;
; call PutVal
;
; endif

0626 B021      movlw  CMD_OK
0627 C56A      goto   cmdFinish

;*****
;*****
; NAME: do_readactvelocity
;
; DESCRIPTION: Returns the measured velocity multiplied by 256 which was
;              captured during the last shutter command.
;
; ARGUMENTS:   v
;
; RETURNS:     The last captured measured velocity times 256 is returned.
;              [800000,7FFFFFFF]
;

do_readactvelocity

MOV24  CMVELOCITY, VALBUF          ; move measured velocity to

0628 6A49      MOVFP  CMVELOCITY+B0,wreg      ; get byte of CMVELOCITY into w
0629 4A31      MOVFP  wreg, VALBUF+B0        ; move to VALBUF (B0)
062A 6A4A      MOVFP  CMVELOCITY+B1,wreg      ; get byte of CMVELOCITY into w
062B 4A32      MOVFP  wreg, VALBUF+B1        ; move to VALBUF (B1)
062C 6A4B      MOVFP  CMVELOCITY+B2,wreg      ; get byte of CMVELOCITY into w
062D 4A33      MOVFP  wreg, VALBUF+B2        ; move to VALBUF (B2)

;
;
; if DECIO
062E E728      call   PutDecVal
;
; else
```


Servo Control of a DC-Brush Motor

```

        call    PutVal
    endif

062F B021        movlw  CMD_OK
0630 C56A        goto   cmdFinish

;*****
;*****
; NAME: do_externalstatus
;
; DESCRIPTION: Returns a two digit hex number which defines the state of
;              the bits in the external status register. Issuing this
;              command will clear all the bits in the external status
;              register unless the event which set the bit is still true.
;
;
; ARGUMENTS:   X
;
; RETURNS:     The external status register is returned.
;

do_externalstatus

0631 8406        bsf    _glintd
0632 6A92        movfp  EXTSTAT,wreg
0633 2992        clrf  EXTSTAT
0634 8C06        bcf    _glintd
0635 E688        call   PutHex

0636 B021        movlw  CMD_OK
0637 C56A        goto   cmdFinish

;*****
;*****
; NAME: do_movestatus
;
; DESCRIPTION: Returns a two digit hex number which defines the state of
;              the bits in the move status register. Issuing this command
;              will clear all the bits in the move status register unless
;              the event which set the bit is still true.
;
;
; ARGUMENTS:   Y
;
; RETURNS:     The move status register is returned.
;

do_movestatus

0638 6A93        movfp  MOVSTAT,wreg
0639 E688        call   PutHex

063A B021        movlw  CMD_OK
063B C56A        goto   cmdFinish

;*****
;*****
; NAME: do_readindposition
;
; DESCRIPTION: Returns the last index position captured in position counts.
;
;
; ARGUMENTS:   I
;
; RETURNS:     The last captured index position is returned.
;

do_readindposition

```



Servo Control of a DC-Brush Motor

```

CLR32    Y

0652 2980          CLRf    Y+B0
0653 2981          CLRf    Y+B1
0654 2982          CLRf    Y+B2
0655 2983          CLRf    Y+B3

0656 B021          movlw   CMD_OK
0657 C56A          goto    cmdFinish

;*****
;*****
; NAME:           do_reset
;
; DESCRIPTION:    Performs a software reset.
;
; ARGUMENTS:      Z
;

do_reset

0658 B021          movlw   CMD_OK
0659 E679          call    PutChar
065A C021          goto    Startup

;*****
;*****
; NAME:           do_stop
;
; DESCRIPTION:    Stops servo by clearing SERVOFLAG.
;
do_stop

065B 2990          clrf    SERVOFLAG

065C B021          movlw   CMD_OK
065D C56A          goto    cmdFinish

;*****
;*****
; NAME:           do_capture
;

do_capture

    if    ((_PICMASTER_DEBUG == 1) && (DECIO == 1))

065E E6CC          call    GetDecVal

    endif

    if    ((_PICMASTER_DEBUG == 1) && (DECIO == 0))

        call    GetVal

    endif

    if    _PICMASTER_DEBUG == 1

        MOV16    VALBUF,CAPCOUNT

065F 6A31          MOVFP    VALBUF+B0,wreg ; get byte of VALBUF into w
0660 01BC          MOVWF    CAPCOUNT+B0 ; move to CAPCOUNT(B0)
0661 6A32          MOVFP    VALBUF+B1,wreg ; get byte of VALBUF into w
0662 01BD          MOVWF    CAPCOUNT+B1 ; move to CAPCOUNT(B1)

```



Servo Control of a DC-Brush Motor

```

                                MOV16  VALBUF,CAPTMP

0663 6A31          MOVFP  VALBUF+B0,wreg ; get byte of VALBUF into w
0664 01BE          MOVWF  CAPTMP+B0      ; move to CAPTMP (B0)
0665 6A32          MOVFP  VALBUF+B1,wreg ; get byte of VALBUF into w
0666 01BF          MOVWF  CAPTMP+B1      ; move to CAPTMP (B1)

0667 B021          movlw  CMD_OK
0668 C56A          goto   cmdFinish

                                endif

;*****
; NAME:          GetPar
;
; DESCRIPTION:   Get a parameter number [0,FF] from the serial port and place
;               it in VALBUF+B0.
;
GetPar

                                CLR24  VALBUF

0669 2931          CLRF   VALBUF+B0
066A 2932          CLRF   VALBUF+B1
066B 2933          CLRF   VALBUF+B2

066C E6B2          call   GetHex
066D 6A4E          movfp  HEXVAL,wreg
066E B50F          andlw  0x0F
066F 4A31          movpf  wreg,VALBUF+B0
0670 1D31          swapf  VALBUF+B0

0671 E6B2          call   GetHex
0672 6A31          movfp  VALBUF+B0,wreg

0673 0E4E          addwf  HEXVAL,W
0674 4A31          movpf  wreg,VALBUF+B0

0675 0002          return

;*****
;*****
; NAME:          GetChar
;
; DESCRIPTION:   Get character from receive buffer.
;
GetChar

0676 B800          movlb  bank0      ; set bank0
0677 540A          movpf  rcreg,wreg ; receive character

0678 0002          return

;*****
;*****
; NAME:          PutChar
;
; DESCRIPTION:   send character out the serial port
;
; ARGUMENTS:    wreg contains byte to be transmitted
;

```

Servo Control of a DC-Brush Motor

```

PutChar
0679 B801          movlb  bank1          ; set bank1
                  bufwait          ; is transmit buffer empty?
067A 9116          btfss  _tbmt
067B C67A          goto    bufwait

067C B800          movlb  bank0          ; set bank0
                  shfwait
067D 9115          btfss  _trmt          ; is transmit shift register empty?
067E C67D          goto    shfwait

067F 4A16          movpf  wreg,txreg        ; if so, send character

0680 0002          return

;*****
;*****
; NAME:           GetChk
;
; DESCRIPTION:    Check if character is in receive buffer.
;
GetChk
0681 B801          movlb  bank1          ; set bank1
0682 560A          movpf  pir,wreg
0683 B501          andlw  CHARREADY        ; return status in wreg
0684 0002          return

;*****
;*****
; NAME:           PutDec
;
; DESCRIPTION:    Converts a hex value [0,F] in wreg to its ASCII equivalent.
;                 The upper nibble of wreg is assumed to be zero.
;
; ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII decimal
;
                  if    DECIO

PutDec
0685 B130          addlw  0x30          ; convert to ASCII
0686 E679          call   PutChar
0687 0002          return

                  endif

;*****
;*****
; NAME:           PutHex
;
; DESCRIPTION:    Convert the wreg value to ASCII hexadecimal. The output
;                 format is two digits with the A-F parts in upper case and
;                 leading zeros. The result is sent out the serial port with
;                 PutChar.
;
; ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII hex
;
PutHex
0688 4A4E          movpf  wreg,HEXVAL
0689 1D0A          swapf  wreg
068A B50F          andlw  0x0F
068B 4A4F          movpf  wreg,HEXTMP

```



Servo Control of a DC-Brush Motor

```

068C 2D0A          negw   wreg
068D B109          addlw  0x09
068E 970A          btfsz  wreg,MSB
068F C693          goto   puth20
0690 B037          movlw  'A'-0x0A
0691 0E4F          addwf  HEXTMP,W
0692 C695          goto   puth25

           puth20
0693 B030          movlw  '0'
0694 0E4F          addwf  HEXTMP,W

           puth25
0695 E679          call   PutChar

0696 6A4E          movfp  HEXVAL,wreg
0697 B50F          andlw  0x0F
0698 4A4F          movfp  wreg,HEXTMP
0699 2D0A          negw   wreg
069A B109          addlw  0x09
069B 970A          btfsz  wreg,MSB
069C C6A0          goto   put120
069D B037          movlw  'A'-0x0A
069E 0E4F          addwf  HEXTMP,W
069F C6A2          goto   put125

           put120
06A0 B030          movlw  '0'
06A1 0E4F          addwf  HEXTMP,W

           put125
06A2 E679          call   PutChar

06A3 0002          return

;*****
;*****
; NAME:           PutStr
;
; DESCRIPTION:    Sends a character string out the serial port.
;

PutStr
06A4 AB4C          tabldr  1,1,STRVALH
GetNextPair

06A5 A24C          tlrdr  1,STRVALH
06A6 A94D          tabldr  0,1,STRVALL

06A7 6A4C          movfp  STRVALH,wreg
06A8 31C1          cpfseq ZERO
06A9 C6AB          goto   puth
06AA 0002          return

           puth
06AB E679          call   PutChar

06AC 6A4D          movfp  STRVALL,wreg
06AD 31C1          cpfseq ZERO
06AE C6B0          goto   putL
06AF 0002          return

           putL
06B0 E679          call   PutChar

06B1 C6A5          goto   GetNextPair

;*****
;*****
; NAME:           GetHex
;
; DESCRIPTION:    Receive an ASCII hex character from the serial port and
;                 convert to numerical value.

```

Servo Control of a DC-Brush Motor

```

;
; RETURNS:      numerical value in HEXVAL

GetHex

getnxt
06B2 E557      call    IdleFunction
06B3 E681      call    GetChk
06B4 31C2      cpfseq  ONE
06B5 C6B2      goto   getnxt

06B6 2950      clr    HEXSTAT
06B7 E676      call    GetChar
06B8 4A4E      movpf  wreg,HEXVAL
06B9 E679      call    PutChar
06BA B00D      movlw  CR
06BB 044E      subwf  HEXVAL,W
06BC 330A      tstfsz wreg
06BD C6BF      goto   gth10
06BE C6C9      goto   gthCR

gth10

06BF 6A4E      movfp  HEXVAL,wreg
06C0 B239      sublw  '9'
06C1 970A      btfsz  wreg,MSB
06C2 C6C5      goto   gth20

06C3 B009      movlw  0x09
06C4 0F4E      addwf  HEXVAL

gth20

06C5 B00F      movlw  0x0F
06C6 0B4E      andwf  HEXVAL
06C7 2950      clr    HEXSTAT
06C8 0002      return

gthCR

06C9 B001      movlw  0x01
06CA 4A50      movpf  wreg,HEXSTAT
06CB 0002      return

;*****

;*****
; NAME:      getval
;
; DESCRIPTION: Get a value [80000,7FFFFFF] from the serial port and place
;              it in VALBUF.
;
;              if DECIO
;              else

GetVal
CLR24 VALBUF

getnext
call    GetHex

movlw  0x01
cpfseq HEXSTAT
goto   shift
return

shift
swapf  VALBUF+B2
movfp  VALBUF+B2,wreg
andlw  0xF0
movpf  wreg,VALBUF+B2
swapf  VALBUF+B1
movfp  VALBUF+B1,wreg
andlw  0x0F

```



Servo Control of a DC-Brush Motor

```

        addwf  VALBUF+B2
        movfp  VALBUF+B1,wreg
        andlw  0xF0
        movpf  wreg,VALBUF+B1
        swapf  VALBUF+B0
        movfp  VALBUF+B0,wreg
        andlw  0x0F
        addwf  VALBUF+B1
        movfp  VALBUF+B0,wreg
        andlw  0xF0
        addwf  HEXVAL,W
        movpf  wreg,VALBUF+B0

        goto  getnext

    endif

;*****
;*****
; NAME:          GetDecVal
;
; DESCRIPTION:   Get a value [-8388608,8388607] from the serial port and
;                place it in VALBUF
;
; RETURNS:      numerical value is returned in VALBUF

        if      DECIO

GetDecVal
        CLR24  VALBUF

06CC 2931          CLRF  VALBUF+B0
06CD 2932          CLRF  VALBUF+B1
06CE 2933          CLRF  VALBUF+B2

06CF E708          call  GetDec
06D0 2B9B          setf  DECSIGN
06D1 B001          movlw DEC_MN
06D2 3199          cpfseq DECSTAT
06D3 299B          clrf  DECSIGN

        getdecnext
06D4 E708          call  GetDec

06D5 B002          movlw DEC_CR
06D6 3199          cpfseq DECSTAT
06D7 C6D9          goto  mull10
06D8 C6FD          goto  fixsign

        mull10

        RLC24  VALBUF          ; multiply VALBUF by two

06D9 8804          BCF   _carry
06DA 1B31          RLCF  VALBUF+B0
06DB 1B32          RLCF  VALBUF+B1
06DC 1B33          RLCF  VALBUF+B2

        MOV24  VALBUF,DVALBUF          ; save in DVALBUF

06DD 6A31          MOVFP  VALBUF+B0,wreg          ; get byte of VALBUF into w
06DE 4A34          MOVFP  wreg,DVALBUF+B0          ; move to DVALBUF (B0)
06DF 6A32          MOVFP  VALBUF+B1,wreg          ; get byte of VALBUF into w
06E0 4A35          MOVFP  wreg,DVALBUF+B1          ; move to DVALBUF (B1)
06E1 6A33          MOVFP  VALBUF+B2,wreg          ; get byte of VALBUF into w
06E2 4A36          MOVFP  wreg,DVALBUF+B2          ; move to DVALBUF (B2)

        RLC24  VALBUF

```


Servo Control of a DC-Brush Motor

```

06E3 8804          BCF      _carry
06E4 1B31          RLCF     VALBUF+B0
06E5 1B32          RLCF     VALBUF+B1
06E6 1B33          RLCF     VALBUF+B2

                RLC24   VALBUF          ; VALBUF now multiplied by eight

06E7 8804          BCF      _carry
06E8 1B31          RLCF     VALBUF+B0
06E9 1B32          RLCF     VALBUF+B1
06EA 1B33          RLCF     VALBUF+B2

                ADD24   DVALBUF,VALBUF ; VALBUF now multiplied by ten

06EB 6A34          MOVFP   DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
06EC 0F31          ADDWF   VALBUF+B0      ; add lowest byte of VALBUF, save in
06ED 6A35          MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
06EE 1132          ADDWFC  VALBUF+B1      ; add 2nd byte of VALBUF, save in
06EF 6A36          MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
06F0 1133          ADDWFC  VALBUF+B2      ; add 3rd byte of VALBUF, save in

                CLR24   DVALBUF

06F1 2934          CLRF    DVALBUF+B0
06F2 2935          CLRF    DVALBUF+B1
06F3 2936          CLRF    DVALBUF+B2

06F4 6A98          movfp   DECVAL,wreg
06F5 4A34          movpf   wreg,DVALBUF+B0
                ADD24   DVALBUF,VALBUF

06F6 6A34          MOVFP   DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
06F7 0F31          ADDWF   VALBUF+B0      ; add lowest byte of VALBUF, save in
06F8 6A35          MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
06F9 1132          ADDWFC  VALBUF+B1      ; add 2nd byte of VALBUF, save in VALBUF (B1)
06FA 6A36          MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
06FB 1133          ADDWFC  VALBUF+B2      ; add 3rd byte of VALBUF, save in VALBUF (B2)

06FC C6D4          goto    getdecnxt
                fixsign

06FD 290A          clrf    wreg
06FE 329B          cpfsgt  DECSIGN
06FF 0002          return

                NEG24   VALBUF

0700 1331          COMF    VALBUF+B0
0701 1332          COMF    VALBUF+B1
0702 1333          COMF    VALBUF+B2
0703 290A          CLRF    wreg
0704 1531          INCF    VALBUF+B0
0705 1132          ADDWFC  VALBUF+B1
0706 1133          ADDWFC  VALBUF+B2

0707 0002          return

                endif

```

Servo Control of a DC-Brush Motor

```
*****
;*****
; NAME:          GetDec
;
; DESCRIPTION:   Receive an ASCII decimal character from the serial port and
;               convert to its numerical value.
;
; ARGUMENTS:    numerical value is returned in DECVAL
;
                if    DECIO

GetDec

getdecnxt
    call    IdleFunction
    call    GetChk
    cpfseq  ONE
    goto   getdecnxt

    call    GetChar
    movpf  wreg,DECVAL
    call    PutChar

    movlw  CR
    subwf  DECVAL,W
    cpfslt ZERO
    goto   gtdCR

    movlw  MN
    subwf  DECVAL,W
    cpfslt ZERO
    goto   gtdMN

    movlw  SP
    subwf  DECVAL,W
    cpfslt ZERO
    goto   gtdSP

gtd09
    movlw  0x0F
    andwf  DECVAL
    clrf   DECSTAT
    return

gtdCR
    movlw  DEC_CR
    movpf  wreg,DECSTAT
    return

gtdMN
    movlw  DEC_MN
    movpf  wreg,DECSTAT
    return

gtdSP
    movlw  DEC_SP
    movpf  wreg,DECSTAT
    return

    endif

;*****
;*****
; NAME:          PutVal
;
; DESCRIPTION:   Sends the value in VALBUF [800000,7FFFFFFF] out the serial
;
;
                if    DECIO
                else
```

Servo Control of a DC-Brush Motor

```

PutVal
    movfp  VALBUF+B2,wreg
    call   PutHex
    movfp  VALBUF+B1,wreg
    call   PutHex
    movfp  VALBUF+B0,wreg
    call   PutHex

    return

endif

;*****
;*****
; NAME:          PutDecVal
;
; DESCRIPTION:   Send the value in VALBUF [-8388608,8388607] out the serial
;

    if  DECIO

PutDecVal
0728 9733          btfs  VALBUF+B2,MSB
0729 C734          goto  pdpos
NEG24 VALBUF

072A 1331          COMF  VALBUF+B0
072B 1332          COMF  VALBUF+B1
072C 1333          COMF  VALBUF+B2
072D 290A          CLRF  wreg
072E 1531          INCF  VALBUF+B0
072F 1132          ADDWFC VALBUF+B1
0730 1133          ADDWFC VALBUF+B2

0731 B02D          movlw  MN
0732 E679          call  PutChar
0733 C736          goto  pddigits

pdpos
0734 B020          movlw  SP
0735 E679          call  PutChar

pddigits
0736 B08D          movlw  (DEC_TABLE & 0xff) ; DEC_TABLE LSB
0737 4A0D          movpf  wreg,tblptrl
0738 B007          movlw  page DEC_TABLE ; DEC_TABLE MSB
0739 4A0E          movpf  wreg,tblptrh

073A A934          tablrd 0,1,DVALBUF+B0
readNextDec

073B A034          tlrld 0,DVALBUF+B0 ; read entry from table
073C AB35          tablrd 1,1,DVALBUF+B1
073D A936          tablrd 0,1,DVALBUF+B2

073E 2B0A          setf  wreg ; unitsposition if end of table
073F 3134          cpfseq DVALBUF+B0
0740 C742          goto  getdigit
0741 C756          goto  unitsposition

getdigit
0742 1534          incf  DVALBUF+B0 ; restore to power of 10
0743 2B98          setf  DECVAL ; set DECVAL to -1

inc
0744 1598          incf  DECVAL ; increment DECVAL
SUB24 DVALBUF,VALBUF ; check if in range
    
```

Servo Control of a DC-Brush Motor

```

0745 6A34      MOVFP  DVALBUF+B0,wreg      ; get lowest byte of DVALBUF into w
0746 0531      SUBWF  VALBUF+B0              ; sub lowest byte of VALBUF, save in
0747 6A35      MOVFP  DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0748 0332      SUBWFB VALBUF+B1        ; sub 2nd byte of VALBUF, save in VALBUF (B1)
0749 6A36      MOVFP  DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
074A 0333      SUBWFB VALBUF+B2        ; sub 3rd byte of VALBUF, save in VALBUF (B2)

074B 9733      btfss  VALBUF+B2,MSB
074C C744      goto   inc

                ADD24  DVALBUF,VALBUF      ; if so, correct VALBUF for next digit

074D 6A34      MOVFP  DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
074E 0F31      ADDWF  VALBUF+B0              ; add lowest byte of VALBUF, save in
074F 6A35      MOVFP  DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0750 1132      ADDWFC VALBUF+B1        ; add 2nd byte of VALBUF, save in VALBUF (B1)
0751 6A36      MOVFP  DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
0752 1133      ADDWFC VALBUF+B2        ; add 3rd byte of VALBUF, save in VALBUF (B2)

0753 6A98      movfp  DECVAL,wreg      ; send DECVAL
0754 E685      call   PutDec

0755 C73B      goto   readNextDec      ; get next table entry

unitsposition
0756 6A31      movfp  VALBUF+B0,wreg ; units position value now in VALBUF
0757 E685      call   PutDec

0758 0002      return

                endif

;*****

;*****
;
;
;          TABLES:

CMD_START CMD_TABLE

CMD_TABLE

CMD_DEF do_null,DO_NULL

0759 000D      DATA  DO_NULL
075A 0570      DATA  do_null

CMD_DEF do_move,DO_MOVE

075B 004D      DATA  DO_MOVE
075C 0572      DATA  do_move

CMD_DEF do_mode,DO_MODE

075D 004F      DATA  DO_MODE
075E 0580      DATA  do_mode

CMD_DEF do_setparameter,DO_SETPARAMETER

075F 0053      DATA  DO_SETPARAMETER
0760 059C      DATA  do_setparameter

CMD_DEF do_readparameter,DO_READPARAMETER

```

Servo Control of a DC-Brush Motor

```
0761 0052          DATA    DO_READPARAMETER
0762 05C3          DATA    do_readparameter

                                CMD_DEF  do_shutter,DO_SHUTTER

0763 0043          DATA    DO_SHUTTER
0764 05ED          DATA    do_shutter

                                CMD_DEF  do_readcomposition,DO_READCOMPOSITION

0765 0050          DATA    DO_READCOMPOSITION
0766 060D          DATA    do_readcomposition

                                CMD_DEF  do_readcomvelocity,DO_READCOMVELOCITY

0767 0056          DATA    DO_READCOMVELOCITY
0768 0616          DATA    do_readcomvelocity

                                CMD_DEF  do_readactposition,DO_READACTPOSITION

0769 0070          DATA    DO_READACTPOSITION
076A 061F          DATA    do_readactposition

                                CMD_DEF  do_readactvelocity,DO_READACTVELOCITY

076B 0076          DATA    DO_READACTVELOCITY
076C 0628          DATA    do_readactvelocity

                                CMD_DEF  do_externalstatus,DO_EXTERNALSTATUS

076D 0058          DATA    DO_EXTERNALSTATUS
076E 0631          DATA    do_externalstatus

                                CMD_DEF  do_movestatus,DO_MOVESTATUS

076F 0059          DATA    DO_MOVESTATUS
0770 0638          DATA    do_movestatus

                                CMD_DEF  do_readindposition,DO_READINDPOSITION

0771 0049          DATA    DO_READINDPOSITION
0772 063C          DATA    do_readindposition

                                CMD_DEF  do_setposition,DO_SETPOSITION

0773 0048          DATA    DO_SETPOSITION
0774 0645          DATA    do_setposition

                                CMD_DEF  do_reset,DO_RESET

0775 005A          DATA    DO_RESET
0776 0658          DATA    do_reset

                                CMD_DEF  do_stop,DO_STOP

0777 0073          DATA    DO_STOP
0778 065B          DATA    do_stop

                                if      _PICMASTER_DEBUG
                                CMD_DEF  do_capture,DO_CAPTURE

0779 0063          DATA    DO_CAPTURE
077A 065E          DATA    do_capture

                                endif

                                CMD_END

;                                                                    ;
077B 0000          DATA    0x00
```



Servo Control of a DC-Brush Motor

```

doCaptureRegs
; !end! hdr !skip start!
079A B000      movlw  (CaptureAddr & 0xff)
079B 010D      movwf  tblptrl
079C B080      movlw  CaptureAddr/256
079D 010E      movwf  tblptrh      ; setup table pointer address

079E AC79      tablwt 0,0,POSERROR+B0      ; dummy tablwt
079F A67A      tlwt 1,POSERROR+B1      ; now table latch = 16 bits contents
of POSERROR

capPerr
07A0 AC79      tablwt 0,0,POSERROR+B0      ; perform actual table write of
POSERROR

07A1 AC7C      tablwt 0,0,VELERROR+B0
07A2 A67D      tlwt 1,VELERROR+B1      ; capture Velocity error

capVerr
07A3 AC7C      tablwt 0,0,VELERROR+B0

07A4 AC72      tablwt 0,0,MPOSITION+B0
07A5 A673      tlwt 1,MPOSITION+B1      ; capture measured position

capMpos
07A6 AC72      tablwt 0,0,MPOSITION+B0

07A7 AC55      tablwt 0,0,POSITION+B0
07A8 A656      tlwt 1,POSITION+B1      ; capture commanded position

capPos
07A9 AC55      tablwt 0,0,POSITION+B0

07AA AC75      tablwt 0,0,MVELOCITY+B0
07AB A676      tlwt 1,MVELOCITY+B1      ; capture measured velocity

capMvel
07AC AC75      tablwt 0,0,MVELOCITY+B0

07AD AC58      tablwt 0,0,VELOCITY+B0
07AE A659      tlwt 1,VELOCITY+B1      ; capture commanded velocity

capVel
07AF AC58      tablwt 0,0,VELOCITY+B0

07B0 AC88      tablwt 0,0,YPWM+B0
07B1 A689      tlwt 1,YPWM+B1      ; capture commanded velocity

capPwm
07B2 AC88      tablwt 0,0,YPWM+B0

DEC16  CAPTMP

07B3 290A      CLRF  wreg
07B4 07BE      DECF  CAPTMP+B0
07B5 03BF      SUBWFB CAPTMP+B1

TFSZ16  CAPTMP

07B6 6ABE      MOVFP  CAPTMP+B0,wreg
07B7 08BF      IORWF  CAPTMP+B1,W
07B8 330A      TSTFSZ wreg

07B9 0002      return

07BA 0001      DATA  0x0001      ; HALT instruction (avail only in

HaltTrace
07BB 29BB      clrf  CAPFLAG
MOV16  CAPCOUNT,CAPTMP

07BC 6ABC      MOVFP  CAPCOUNT+B0,wreg      ; get byte of CAPCOUNT into w
07BD 01BE      MOVWF  CAPTMP+B0      ; move to CAPTMP (B0)
07BE 6ABD      MOVFP  CAPCOUNT+B1,wreg      ; get byte of CAPCOUNT into w

```

Servo Control of a DC-Brush Motor

```
07BF 01BF          MOVWF  CAPTMP+B1          ; move to CAPTMP (B1)

07C0 0002          return

;*****

endif

END

Errors : 0
Warnings : 0
```


SECTION 5

INTERFACING PIC16/17 WITH SERIAL EEPROMS

Communicating with I ² C Bus Using PIC16C5X - AN515	5-	1
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X - AN530	5-	11
Logic Powered Serial EEPROMs - AN535	5-	29

Communicating with the I²C™ Bus Using the PIC16C5X

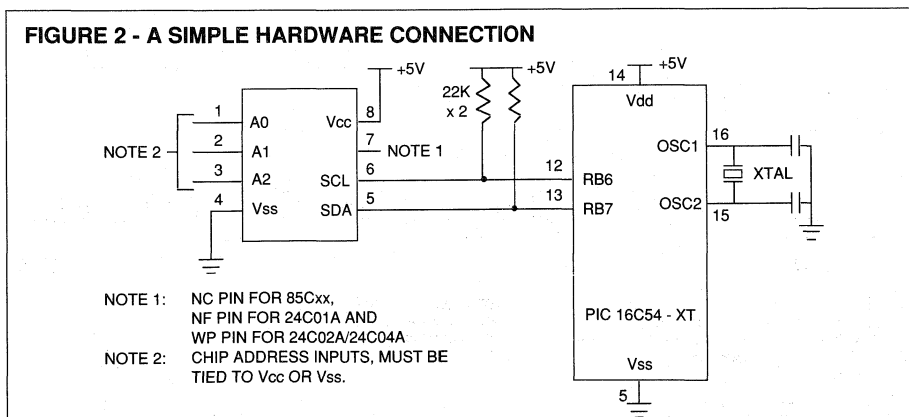
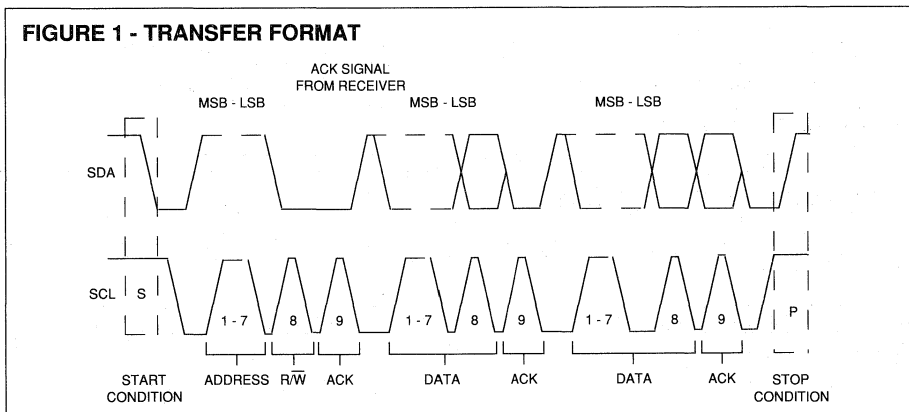
INTRODUCTION

The Microchip Technology Inc.'s 24CXX and 85CXX serial EEPROMs feature a two wire serial interface bus. The bus protocol is I²C compatible. Interface to a serial port with I²C bus protocol in a microcontroller is trivial. This application note is intended for design engineers who want to develop their software programs to communicate a microcontroller with a 2-wire bus serial EEPROM through a general purpose I/O port.

Unlike the 3-wire bus serial EEPROMs, the 24CXX/85CXX communicate with any microcontroller only by a serial data I/O line (SDA) and a serial clock (SCL). Chip select is not required. Data transfer may be initiated only when the bus is not busy. During such transfer, the data

line (SDA) must remain stable whenever the clock line (SCL) is high. Changes in the data line while the clock line is high are interpreted as a START or STOP condition. A typical transfer format is shown in Figure 1.

After the START condition, a slave address is sent. This address is 7-bits long, the eighth bit is a data direction bit. (R/W - a logical '0' indicates a transmission WRITE, a logical '1' represents a request for data READ). A data transfer is always terminated by a STOP condition generated by the master controller. However, if a master still wishes to communicate on the bus, it can generate another START condition and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such transfer.



Communicating with I²C Bus

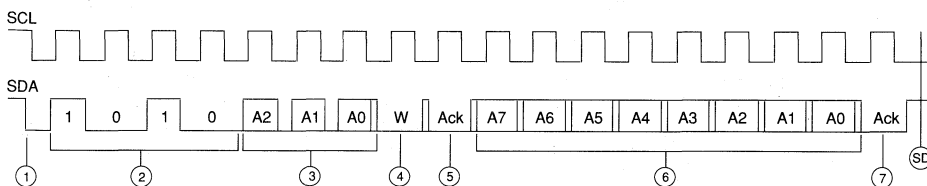
An example program has been provided in Appendix A containing all PIC16C54 routines needed to exercise a 24CXX or 85CXX device. A simple hardware connection is illustrated in Figure 2. A maximum of eight 24C01A/24C02A/85C72/85C82's, or four 24C04A/85C92's can be addressed by a microcontroller on the same two wire bus without additional interfaces. Each device is identified by its Chip Address and will only respond to the correct slave address. A detailed bus flow is shown in Figure 3.

Figure 3 as shown below describes how the bit stream is set up for READ and WRITE mode in the microcomputer programming software prior to sending it on the two wire serial bus.

The stop condition, after the write sequence, starts the internal self-timed write cycle which may last up to 6 milliseconds (.7 ms per byte). Acknowledge signal should be monitored during this period.

*CONTACT: Bruce Negley
Memory Products Division*

FIGURE 3A - SETTING THE INTERNAL WORD ADDRESS OF THE 24CXX/85CXX



- | | |
|--|--|
| <ol style="list-style-type: none"> 1. START CONDITION 2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX 3. DEVICE ADDRESS A0=PA, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92 4. 0=WRITE TO ADDRESS REGISTER | <ol style="list-style-type: none"> 5. EEPROM DRIVES BUS LOW WITH ACKNOWLEDGEMENT (TIMEOUT IF NO RESPONSE) 6. ADDRESS FOR DATA, A7-A0 (MSB-LSB) 7. BUS LOW FOR ACKNOWLEDGEMENT |
|--|--|

FIGURE 3B - BYTE WRITE SEQUENCE

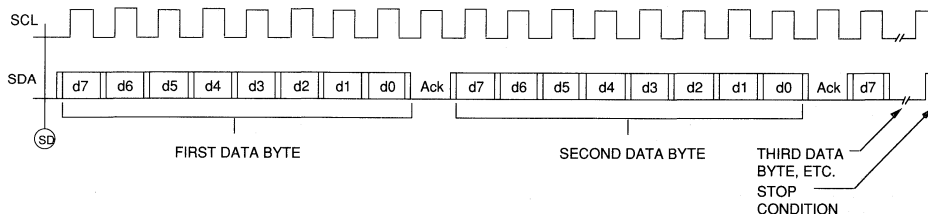
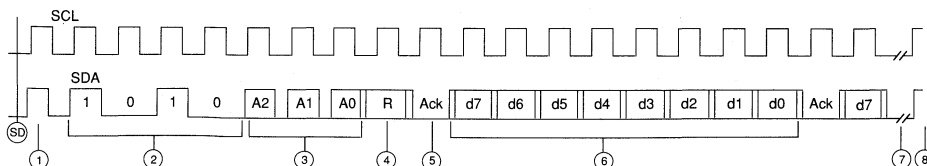


FIGURE 3C - READ MODE SEQUENCE



- | | |
|--|--|
| <ol style="list-style-type: none"> 1. START CONDITION 2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX 3. DEVICE ADDRESS A2, A1, A0. A0 = PA, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92 4. 1 = READ MODE | <ol style="list-style-type: none"> 5. ACKNOWLEDGE FROM EEPROM 6. FIRST DATA BYTE 7. SECOND DATA BYTE, ETC. 8. STOP CONDITION |
|--|--|

READ UP TO 128 BYTES (24C01A, 85C72)
READ UP TO 256 BYTES (24C02A/04A, 85C82/92)

Communicating with I²C Bus

Appendix A:

MPALC CROSS ASSEMBLER 2.00 d:\seeprom\appnotes\i2cbus.asm
Apr 11 15:36:02 1990 PAGE 1

TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

```
0001          TITLE "TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x"
0002          ;
0003          LIST          P=16C54
0004          ;
0005          ;*****
0006          ;** Two wire/I2C Bus READ/WRITE Sample Routines
          ;    of Microchip's 24CXX/85CXX serial CMOS
0007          ;** EEPROM interfacing to a PIC16C54 8-bit CMOS
0008          ;** single chip microcomputer
0009          ;**
0010          ;** Part use = PIC16C54-XT/JW
0011          ;** Note: 1) All timings are based on a
          ;    reference crystal frequency of 2 MHz which
          ;    is equivalent to an instruction cycle
0012          ;** time of 2 usec.
0013          ;** 2) Address and literal values are read
          ;    in octal unless otherwise specified.
          ;    3) The following sample program is
          ;    intended to interface a two wire/I2C
          ;    serial EEPROM with a PIC16C54 on a
          ;    stand-alone application only.
          ;    In the case where the two wire bus is
          ;    multiplexing with other circuitry, it is
          ;    recommended to check the 24CXX/85CXX in
          ;    standby mode to avoid bus contention.
0014          ;**
0015          ;*****
0016          ;
0017          ;-----
0018          ;    Files Assignment
0019          ;-----
0020          ;
0021          0002    PC    EQU    2          ; Program counter
0022          0004    FSR   EQU    4          ; File Select Register
0023          0005    RA    EQU    5          ; Port A use to select
          ;                                ; device address
0024          0006    RB    EQU    6          ; RB7 = SDA, RB6 = SCL
0025          ;
0026          0010    STATUS EQU    10        ; Status register
0027          0011    FLAG  EQU    11        ; Common flag bits
          ;                                ; register
0028          0012    EEPROM EQU    12       ; Bit buffer
0029          0013    ERCODE EQU    13       ; Error code (to indicate
          ;                                ; bus status)
0030          0020    ADDR  EQU    20        ; Address register
```

Communicating with I²C Bus

```

0031  0021  DATAI EQU 21      ; Stored data input
                                ; register
0032  0022  DATAO EQU 22      ; Stored data output
                                ; register
0033  0023  SLAVE  EQU 23      ; Device address
                                ; (1010xxx0)
0034  0024  TXBUF  EQU 24      ; TX buffer
0035  0025  RXBUF  EQU 25      ; RX buffer
0036  0026  COUNT  EQU 26      ; Bit counter
0037      ;
0038  0030  TIMER0 EQU 30      ; Delay timer0
0039  0031  TIMER1 EQU 31      ; Delay timer1
0040      ;
0041      ;
0042      ;-----
0043      ;                      Bit Assignments
0044      ;-----
0045      ;
0046      ;   FLAG Bits
0047      ;
0048  0000  ERROR  EQU 0       ; Error flag
0049      ;
0050      ;   EEPROM Bits
0051      ;
0052  0007  DI      EQU 7       ; EEPROM input
0053  0006  DO      EQU 6       ; EEPROM output
0054      ;
0055      ;   I2C Device Bits
0056      ;
0057  0007  SDA      EQU 7       ; RB7, data in/out
0058  0006  SCL      EQU 6       ; RB6, serial clock
0059      ;
0060      ;   END FILES/BITS EQUATE
0061      ;
0062      ;
0063      ;-----
0064      ;   Two wire/I2C - CPU communication error status table
                                ; and subroutine
0065      ;-----
0066      ;   input : W-reg          = error code
0067      ;   output : ERCODE       = error code
0068      ;           FLAG(ERROR)   = 1
0069      ;
0070      ;   code          error status mode
0071      ;   -----
0072      ;   1           :   SCL locked low by device (bus is still
                                ; busy)

```

Communicating with I²C Bus

```

0073 ; 2 : SDA locked low by device (bus is still
        busy)
0074 ; 3 : No acknowledge from device (no
        handshake)
0075 ; 4 : SDA bus not released for master to
        generate STOP bit

0076 ;-----
0077 ;
0078 ; Subroutine to identify the status of the serial clock ;
        (SCL) and serial data
0079 ; (SDA) condition according to the error status table. 0080 ;
        Codes generated are useful for bus/device diagnosis.

0081 ;
0082 ERR
0083 0000 3411 BTFSS FLAG,ERROR ; Remain as first error
        ; encountered
0084 0001 0053 MOVWF ERCODE ; Save error code
0085 0002 2411 BSF FLAG,ERROR ; Set error flag
0086 0003 4000 RETLW 0
0087 ;
0088 ;-----
0089 ; START bus communication routine
0090 ;-----
0091 ; input : none
0092 ; output : initialize bus communication
0093 ;-----
0094 ;
0095 ; Generate START bit (SCL is high while SDA goes from
        ; high to low transition) and check status of the
0096 ; serial clock.
0097 BSTART
0098 0004 6077 MOVLW B'00111111' ; Put SCL, SDA line in
        ; output state
0099 0005 0006 TRIS RB
0100 0006 2706 BSF RB,SCL ; Set clock high
0101 0007 6001 MOVLW 1 ; Ready error status
        ; code 1
0102 0010 3706 BTFSS RB,SCL ; Locked?
0103 0011 4400 CALL ERR ; SCL locked low by device
0104 0012 2346 BCF RB,SDA ; SDA goes low during SCL
        ; high
0105 0013 0000 NOP ; Timing adjustment
0106 0014 0000 NOP
0107 0015 0000 NOP
0108 0016 2306 BCF RB,SCL ; Start clock train
0109 0017 4000 RETLW 0
0110 ;
0111 ;END SUB
0113 ;

```

Communicating with I²C Bus

```
0114      ;-----
0115      ;          STOP bus communication routine
0116      ;-----
0117      ; Input : None
0118      ; Output      : Bus communication, STOP condition
0119      ;-----
0120      ;
0121      ; Generate STOP bit (SDA goes from low to high during
0122      ; SCL high state)
0123      ; and check bus conditions.
0124      ;
0125      ; BSTOP
0126      0020    2346    BCF      RB,SDA      ; Return SDA to low
0127      0021    0000    NOP
0128      0022    0000    NOP
0129      0023    2706    BSF      RB,SCL      ; Set SCL high
0130      0024    6001    MOVLW   1           ; Ready error code 1
0131      0025    3706    BTFSS   RB,SCL      ; High?
0132      0026    4400    CALL    ERR        ; No, SCL locked low by
0133      ; device
0134      0027    2746    BSF      RB,SDA      ; SDA goes from low to
0135      ; high during SCL high
0136      0030    6004    MOVLW   4           ; Ready error code 4
0137      0031    3746    BTFSS   RB,SDA      ; High?
0138      0032    4400    CALL    ERR        ; No, SDA bus not release
0139      ; for STOP
0140      0033    4000    RETLW   0
0141      ;
0142      ;END SUB
0143      ;
0144      ;-----
0145      ; Serial data send from PIC16CXX to serial EEPROM,
0146      ; bit-by-bit subroutine
0147      ;-----
0148      ; Input : None
0149      ; Output      : To (DI) of serial EEPROM device
0150      ;-----
0151      ; BITIN
0152      0034    6277    MOVLW   B'10111111' ; Force SDA line as input
0153      0035    0006    TRIS    RB
0154      0036    2746    BSF      RB,SDA      ; Set SDA for input
0155      0037    2352    BCF      EEPROM,DI
0156      0040    2706    BSF      RB,SCL      ; Clock high
0157      0041    6001    MOVLW   1
0158      0042    3306    BTFSC   RB,SCL      ; Skip if SCL is high
0159      0043    5047    GOTO    BIT1
0160      0044    3411    BTFSS   FLAG,ERROR ; Remain as first error
0161      ; encountered
```


Communicating with I²C Bus

```
0157 0045 0053 MOVWF ERCODE ; Save error code
0158 0046 2411 BSF FLAG,ERROR ; Set error flag
0159 BIT1
0160 0047 3346 BTFSC RB,SDA ; Read SDA pin
0161 0050 2752 BSF EEPROM,DI ; DI = 1
0162 0051 0000 NOP ; Delay
0163 0052 2306 BCF RB,SCL ; Return SCL to low
0164 0053 4000 RETLW 0
0165 ;
0166 ;END SUB
0168 ;
0169 ;-----
0170 ; Serial data receive from serial EEPROM to PIC16CXX,
; bit-by-bit subroutine
0171 ;-----
0172 ; Input : EEPROM file
0173 ; Output : From (DO) of serial EEPROM device
; to PIC
0174 ;-----
0175 ;
0176 BITOUT
0177 0054 6077 MOVLW B'00111111' ; Set SDA, SCL as outputs
0178 0055 0006 TRIS RB
0179 0056 3712 BTFSS EEPROM,DO
0180 0057 5070 GOTO BIT0
0181 0060 2746 BSF RB,SDA ; Output bit 0
0182 0061 6002 MOVLW 2
0183 0062 3346 BTFSC RB,SDA ; Check for error code 2
0184 0063 5074 GOTO CLK1
0185 0064 3411 BTFSS FLAG,ERROR ; Remain as first error
; encountered
0186 0065 0053 MOVWF ERCODE ; Save error code
0187 0066 2411 BSF FLAG,ERROR ; Set error flag
0188 0067 5074 GOTO CLK1 ; SDA locked low by device
0189 ;
0190 BIT0
0191 0070 2346 BCF RB,SDA ; Output bit 0
0192 0071 0000 NOP ; Delay
0193 0072 0000 NOP
0194 0073 0000 NOP
0195 CLK1
0196 0074 2706 BSF RB,SCL
0197 0075 6001 MOVLW 1 ; Error code 1
0198 0076 3306 BTFSC RB,SCL ; SCL locked low?
0199 0077 5103 GOTO BIT2 ; No.
0200 0100 3411 BTFSS FLAG,ERROR ; Yes.
0201 0101 0053 MOVWF ERCODE ; Save error code
```

Communicating with I²C Bus

```
0202 0102 2411 BSF FLAG,ERROR ; Set error flag
0203          BIT2
0204 0103 0000 NOP
0205 0104 0000 NOP
0206 0105 2306 BCF RB,SCL ; Return SCL to low
0207 0106 4000 RETLW 0
0208      ;
0209      ;END SUB
0211      ;
0212      ;
0213      ;-----
0214      ; RECEIVE DATA subroutine
0215      ;-----
0216      ; Input : None
0217      ; Output : RXBUF = Receive 8-bit data
0218      ;-----
0219      ;
0220      RX
0221 0107 6010 MOVLW .8 ; 8 bits of data
0222 0110 0066 MOVWF COUNT
0223 0111 0165 CLRF RXBUF
0224      ;
0225      RXLP
0226 0112 1565 RLF RXBUF ; Shift data to buffer
0227 0113 SKPC
0228 0113 3403 + BTFSS 3,0
0228 0114 2025 BCF RXBUF,0 ; carry -> f(0)
0229 0115 SKPNC
0230 0115 3003 + BTFSC 3,0
0230 0116 2425 BSF RXBUF,0
0231 0117 4434 CALL BITIN
0232 0120 3352 BTFSC EEPROM,DI
0233 0121 2425 BSF RXBUF,0 ; Input bit =1
0234 0122 1366 DECFSZ COUNT ; 8 bits?
0235 0123 5112 GOTO RXLP
0236 0124 2712 BSF EEPROM,DO ; Set acknowledge bit = 1
0237 0125 4454 CALL BITOUT ; to STOP further input
0238 0126 4000 RETLW 0
0239      ;
0240      ;END SUB
0241      ;
0242      ;-----
0243      ; TRANSMIT DATA subroutine
0244      ;-----
0245      ; Input : TXBUF
0246      ; Output : Data X'mitted to EEPROM device
0247      ;-----
0248      ;
0249      TX
0250 0127 6010 MOVLW .8
```

Communicating with I²C Bus

```

0251 0130 0066 MOVWF COUNT
0252          ;
0253          TXLP
0254 0131 2312 BCF   EEPROM,DO ; Shift databit out.
0255 0132 3364 BTFSC TXBUF,7  ; If shiftedbit=0, data
          ; bit = 0
0256 0133 2712 BSF   EEPROM,DO ; Otherwise data bit = 1
0257 0134 4454 CALL  BITOUT  ; Serial data out
0258 0135 1564 RLF   TXBUF   ; Rotate TXBUF left
0259 0136          SKPC     ; f(6) -> f(7)
0260 0136 3403 + BTFSS 3,0
0260 0137 2024 BCF   TXBUF,0  ; f(7) -> carry
0261 0140          SKPNC    ; carry -> f(0)
0262 0140 3003 + BTFSC 3,0
0262 0141 2424 BSF   TXBUF,0
0263 0142 1366 DECFSZ COUNT  ; 8 bits done?
0264 0143 5131 GOTO  TXLP   ; No.
0265 0144 4434 CALL  BITIN  ; Read acknowledge bit
0266 0145 6003 MOVLW 3
0267 0146 3352 BTFSC EEPROM,DI ; Check for
          ; acknowledgement
0268 0147 4400 CALL  ERR    ; No acknowledge from
          ; device
0269 0150 4000 RETLW 0
0270          ;
0271          ;END SUB
0273          ;
0274          ;-----
0275          ; BYTE-WRITE, write one byte to EEPROM device
0276          ;-----
0277          ; Input   : DATA0 = data to be written
0278          ;          ADDR = destination address
0279          ;          SLAVE = device address (1010xxx0)
0280          ; Output  : Data written to EEPROM device
0281          ;-----
0282          ;
0283 0200          ORG    200   ; The location for BYTE-
          ; WRITE routine can be
0284          ;          ; assigned anywhere
          ;          ; between (377- 777)
          ;          ; octal.
0285          WRBYTE
0286 0200 1023 MOVF  SLAVE,W  ; Get SLAVE address
0287 0201 0064 MOVWF TXBUF   ; to TXbuffer
0288 0202 4404 CALL  BSTART  ; Generate START bit
0289 0203 4527 CALL  TX      ; Output SLAVE address
0290 0204 1020 MOVF  ADDR,W  ; Get WORD address
0291 0205 0064 MOVWF TXBUF   ; intobuffer
0292 0206 4527 CALL  TX      ; Output WORD address
0293 0207 1022 MOVF  DATA0,W ; Move DATA
0294 0210 0064 MOVWF TXBUF   ; intobuffer

```

Communicating with I²C Bus

```
0295 0211 4527 CALL TX ; Output DATA and detect
; acknowledgement
0296 0212 4420 CALL BSTOP ; Generate STOP bit
0297 ;
0298 ;
0299 ;
0300 ;-----
0301 ; BYTE-READ, read one byte from serial EEPROM
; device
0302 ;-----
0303 ; Input : ADDR = source address
0304 ; SLAVE = device address (1010xxx0)
0305 ; Output : DATAI = data read from serial
; EEPROM
0306 ;-----
0307 ;
0308 0300 ORG 300 ; The location for BYTE-
; READ routine can be
; assigned anywhere
0309 ; ; between (377-777) octal.
0310 RDBYTE
0311 0300 1023 MOVF SLAVE,W ; Move SLAVE address
0312 0301 0064 MOVWF TXBUF ; into buffer (R/W = 0)
0313 0302 4404 CALL BSTART ; Generate START bit
0314 0303 4527 CALL TX ; Output SLAVE address.
; Check ACK.
0315 0304 1020 MOVF ADDR,W ; Get WORD address
0316 0305 0064 MOVWF TXBUF
0317 0306 4527 CALL TX ; Output WORD address.
; Check ACK.
0318 0307 4404 CALL BSTART ; START READ (if only one
; device
0319 0310 1023 MOVF SLAVE,W ; is connected to the I2C
; bus)
0320 0311 0064 MOVWF TXBUF
0321 0312 2424 BSF TXBUF,0 ; Specify READ mode
; (R/W = 1)
0322 0313 4527 CALL TX ; Output SLAVE address
0323 0314 4507 CALL RX ; READ in data and
; acknowledge
0324 0315 4420 CALL BSTOP ; Generate STOP bit
0325 0316 1065 MOVF RXBUF ; Save data from buffer
0326 0317 0061 MOVWF DATAI ; to DATAI file.
0327 ;
0328 ;
0329 ;
0330 ; END
```

%ASM-I, No Errors, No Warnings

Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers

INTRODUCTION

Microchip Technology Inc.'s popular 93C46/56/66 and 93LC46/56/66 Serial EEPROMs feature a three/four wire serial interface bus. The attractive price and simple interface make it the ideal device for additional memory space. This application note is intended for design engineers who wish to incorporate a pre-packaged serial EEPROM interface driver into their application.

THE HARDWARE CONNECTION

A typical 4-wire hardware connection is illustrated in Figure 1a and a typical 3-wire connection is illustrated in Figure 1b. Since all I/O ports on the PIC16C5X are configurable as input and/or output, a 3-wire interface makes optimum utilization of the I/O pins by having a common connection for the DI and DO lines of the serial EEPROM. The port pin on the PIC16C5X connected to these pins, has a default setting as an output and is configured, when needed, as an input during program execution.

THE SOFTWARE CONNECTION

An example interface driver is listed in Appendix A. A flow diagram is given in Figure 2. The interface driver is written to minimize both overhead to the calling program as well as the program space necessary for its inclusion into the user's code. The driver has been written as a generic driver to service all 93 Series serial EEPROMs made by Microchip. Processor resources which must be made available to the driver prior to being called are: 1) Two levels of processor stack. 2) Six register locations (four for command/data passing and two for software counters). 3) The File Select Register (FSR), which is used to pass a command/data string pointer to the driver.

Note: The four command/data passing registers have to be defined consecutively in order for the FSR to access them successfully in the program execution.

The user should take the following steps when using the routines provided in Appendix A.

- Specify and define a 3-/4-wire interface by defining the common connection to the DI/DO lines and setting the equate 'wire3' TRUE or FALSE (4-wire is automatically assumed if 3-wire is false).
- Specify and define if 16-bit or 8-bit data organization is used, by setting equate 'org8' TRUE or FALSE.
- The user should assemble the source file by specifying which type of serial EEPROM is being used.

FIGURE 1A - 4 WIRE CONNECTION

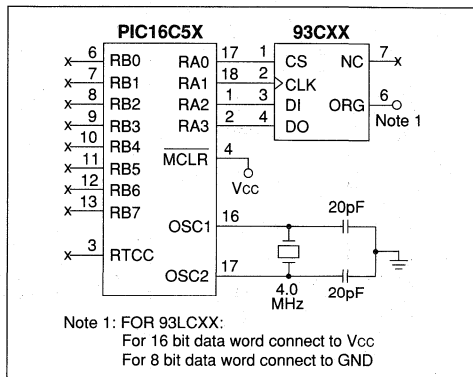
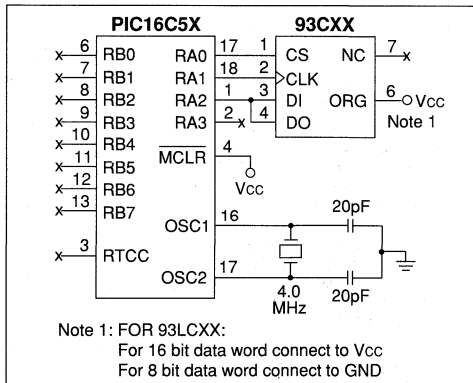


FIGURE 1B - 3 WIRE CONNECTION



This is done by defining the equate S93C46, S93LC46 etc. as TRUE. Only one device can be TRUE, the rest have to be defined FALSE.

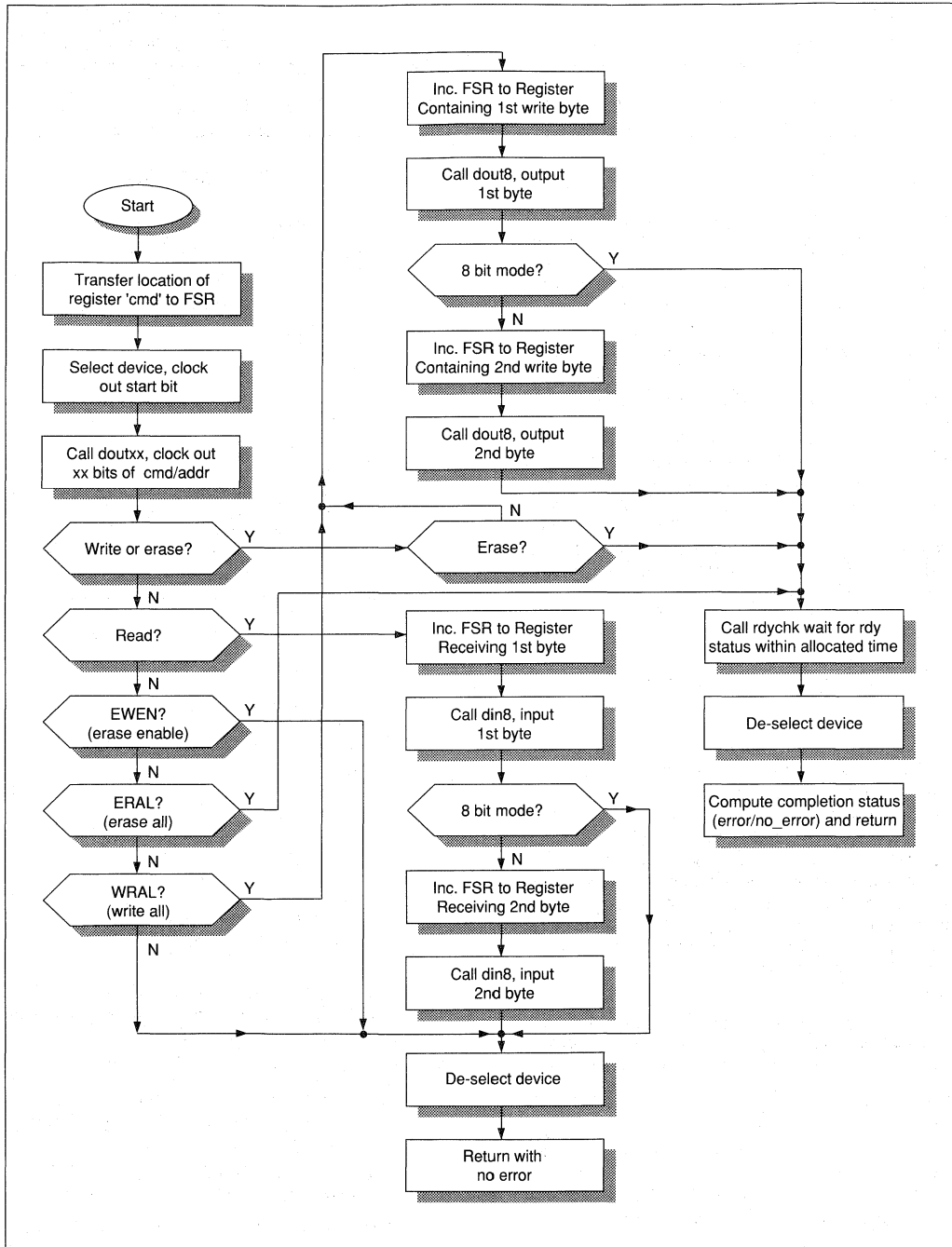
D) The user would invoke the driver as follows:

- Load the register defined as 'cmd' with the 93CX6 Command Opcode (only the four upper bits are used in this register; see Figure 3).
- If necessary, load the register defined 'addr' with the lower 8/7/6 bit address of the location.
- If necessary, load the 9th bit of the address as bit 3 of the register defined as 'cmd' (see Figure 3).

Note: READ, WRITE and ERASE commands need to have an address associated with the command and the 9th bit of the address is only

Interfacing 93 Series Serial EEPROMs

FIGURE 2 - FLOW CHART



Interfacing 93 Series Serial EEPROMs

required when using the 93C56/66 or 93LC56/66 devices in the 8 bit mode (ORG tied to GND).

4. If necessary, load the register defined as 'highb' and 'lowb' with the 16 bits of data, the most significant byte loaded in 'highb'. In 8 bit mode, 'highb' should be loaded with the 8 bit data.

Note: Only the WRITE and WRAL commands require data to be loaded in the 'highb', 'lowb' locations.

5. Call the driver sub-routine 'see'.
6. Upon completion, the driver will return a completion status in the W register (error/no error). Only commands requiring a status check are capable of returning a valid error/no error status, in all other cases a no error is returned.
7. If the READ command is executed, the 16/8 bit data will be loaded in the 'highb' and 'lowb' registers, where 'highb' contains the MSB in the 16 bit mode and 8 bit data in the 8-bit mode.

The Example interface assumes a 4 MHz oscillator clock which gives us a 1 μ S instruction cycle. If a higher clock speed is used, additional NOPs have to be included in the code in order to meet the minimum clock speed requirements of the 93 Series serial EEPROMs (see data sheet for further details).

Listing in Appendix B is for an interface to 93C46 Serial EEPROM only.

SUMMARY

The 93 Series serial EEPROMs are a simple and versatile method of increasing read/write memory capability in a PIC16C5X application. The 'generic' code in Appendix A makes it easy to incorporate in any PIC16C5X application. Any of the 93 series serial EEPROM can be applied, while at the same time using a minimal amount of I/O, code and RAM resources.

Code size: 6 bytes of RAM.

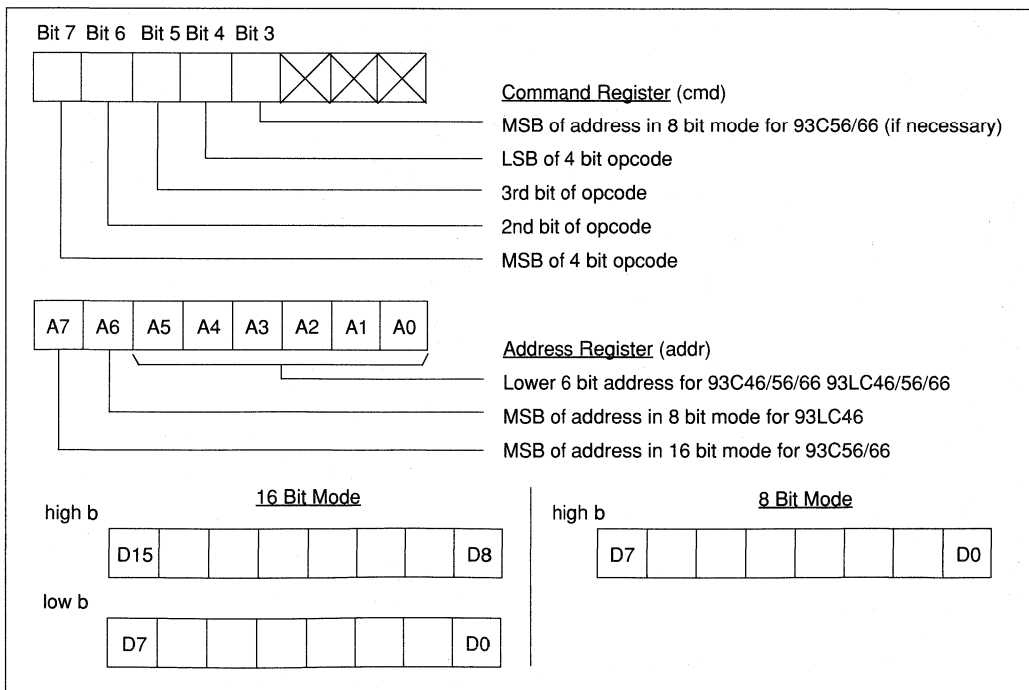
Appendix A listing: 127 bytes program code (max.)

100 bytes program code (min.)

Appendix B listing: 86 bytes program code

*AUTHORS: Stan D'Souza
Logic Products Division
Bob Ward
Field Applications/Sales Organization*

FIGURE 3 - 'CMD', 'ADDR', DATA BYTE DEFINITION



Interfacing 93 Series Serial EEPROMs

Appendix A

Microchip Technology 16c5X Assembler 3.03A Wed Feb 26 11:09:07 1992 Page 1
R/W EEPROM

```
Line  PC  Opcode
0001          LIST  P = 16C54
0002          ;
0003          ;Define Equates:
0004          ;
0005      01FF      PIC54  EQU   1FFH
0006          ;
0007          ;
0008      0000          ORG   0
0009      START
0010      0000      0A7B  goto  main      ;run test program
0011          ;
0012          ;
0013      0001      TRUE  EQU   1
0014      0000      FALSE EQU   0
0015      0000      S93C46 EQU  FALSE
0016      0000      S93LC46 EQU  FALSE
0017      0000      S93C56 EQU  FALSE
0018      0000      S93LC56 EQU  FALSE
0019      0001      S93C66 EQU  TRUE
0020      0000      S93LC66 EQU  FALSE
0021      0001      wire3 equ   TRUE      ;for four-wire setup equate to FALSE
0022      0000      org8  EQU   FALSE
0023          ;
0024      0001      H16   EQU  TRUE
0025      0000      H8    EQU  FALSE
0026      0000      LC468 EQU  FALSE
0027      0000      XC46 EQU  FALSE
0028
0029
0030
0031          ;*****
0032          ;*          Register Assignments
0033          ;*
0034          ;*****
0035      0000      indir equ   0          ;Use this register as source/destination
0036          ;for indirect addressing.
0037      0002      pc    equ   2          ;PIC Program Counter.
0038      0003      status equ  3          ;PIC Status Register.
0039      0004      fsr   equ   4          ;File Select Register.
0040      0005      serial equ  5          ;Port used for 93CX6 control.
0041          ;The following four registers must be
0042          ;located consecutively in memory.
0043      001A      cmd   equ   1a         ;This register contains the 4 bit
0044          ;command op code for 93CX6 as follows:
0045          ;bit 7 msb of command op code
0046          ;bit 6 next bit of op code
0047          ;bit 5 next bit of op code
0048          ;bit 4 lsb of op code
0049          ;bit 3 A8 of address in case of
0050          ;56/66 in 8 bit mode.
0051      001B      addr  equ   1b         ;memory address of lower 7/8 bits
0052
0053      001C      highb equ   1c         ;Used in read/write routines to store the
0054          ;upper byte of a 16 bit data word,
0055      001D      lowb  equ   1d         ;for the data in a 8 bit data word
0056          ;Used in read/write routines to store the
0057          ;lower byte of a 16 bit data word,
0058          ;or not used in 8 bit data word.
0059      001E      cnthi equ   1e         ;Used as the upper byte of a sixteen bit
0060          ;loop counter in RDYCHK routine.
```


Interfacing 93 Series Serial EEPROMs

```

0061      001F      cnt      equ      1f          ;Used as the lower byte of a sixteen bit
0062                                          ;loop counter in RDYCHK routine, and
0063                                          ;elsewhere as an eight bit counter.
0064      001E      temp_cmd   equ      1e          ;doubles as a temp register for cmd
0065      001F      temp_addr   equ      1f          ;doubles as a temp register for addr
0066
0067      ;*****
0068      ;*
0069      ;*          Bit Assignments
0070      ;* The following assignments are for 3-wire. For 4-wire please assign
0071      ;* din and dout to two separate pins.
0072      0000      carry      equ      0          ;Carry Flag of Status Register.
0073      0002      zflag      equ      2          ;Zero Flag of Status Register.
0074      0002      cs         equ      2          ;Port pin tied to CS on 93CX6.
0075      0001      din        equ      1          ;Port pin tied to DI on 93CX6. 3-wire setup
0076      0001      dout       equ      1          ;Port pin tied to DO on 93CX6. 3-wire setup
0077      0003      clock      equ      3          ;Port pin tied to CLK on 93CX6.
0078
0079      ;*****
0080      ;*          General Assignments
0081      ;*
0082      ;*****
0083      0000      no_err     equ      0          ;After issuing a WRITE, ERASE, ERAL, or WRAL
0084      0001      error      equ      1          ;command, the approximate number of machine
0085      0020      tries      equ      20         ;cycles X 256 to wait for the RDY status.
0086                                          ;This value must be adjusted for operating
0087                                          ;frequencies other than 4 MHz.
0088
0089
0090
0091      0080      read       equ      b'10000000' ;read command op code
0092      0040      write      equ      b'01000000' ;write command op code
0093      00C0      erase      equ      b'11000000' ;erase command op code
0094      0030      ewen      equ      b'00110000' ;erase enable command opcode
0095      0000      ewds      equ      b'00000000' ;erase disable command opcode
0096      0020      eral      equ      b'00100000' ;erase all command op code
0097      0010      wral      equ      b'00010000' ;write all command op code
0098
0099      ;*****
0100      ;*          Macro Definitions
0101      ;*
0102      ;*****
0103      sel        MACRO          ;Selects the 93CX6 device
0104      bsf        serial,cs      ;Chip Select (CS) = '1' to select
0105      ENDM          ;the device
0106
0107      dsel       MACRO          ;De-select the 93CX6 device.
0108      bcf        serial,cs      ;Chip Select (CS) = '0' to de-select
0109      ENDM          ;the device.
0110
0111
0112      strtbt     MACRO          ;Issue the Start Bit to the 93CX6.
0113      bsf        serial,din     ;Start Bit = '1'.
0114      clkkit     clkkit         ;Clock it out.
0115      ENDM
0116
0117      clkkit     MACRO          ;Clocks a serial data bit into or out
0118      ENDM          ;of the 93CX6 device.
0119      bsf        serial,clock   ;Clock (CLK) = '1'.
0120
0121      nop
0122      ;Adjust the number of nop instructions
0123      ;between the assertion and de-assertion of
0124      ;CLK in proportion to the PIC operating
0125      ;frequency. Refer to the 93CX6 data for the
0126      ;minimum CLK period.

```



Interfacing 93 Series Serial EEPROMs

```

0127          bcf      serial,clock          ;Clock (CLK) = '0'.
0128          ENDM
0129          ;
0130          ;*****
0131          ;*          DOUTx
0132          ;*
0133          ;*****
0134          ;doutxx, outputs up to 11 bits of op code/data, depending on whether
0135          ;a 46/56/66 serial eeprom is being used. The number of bits over 8 are
0136          ;saved in the cmd register and the rest in the addr register. Before
0137          ;calling this routine the cmd and the addr registers should be loaded
0138          ;as follows:
0139          ;cmd reg.bits 7|6|5|4|3|2|1|0
0140          ;-----|---|---|---|---|
0141          ;          X|X|X|X|X|X|X|Y|  -> not used
0142          ;          X|X|X|X|X|X|Y|X|  -> not used
0143          ;          X|X|X|X|X|Y|X|X|  -> not used
0144          ;          X|X|X|X|Y|X|X|X|  -> 9th bit of address if necessary
0145          ;          X|X|X|Y|X|X|X|X|  -> lsb of command op code
0146          ;          X|X|Y|X|X|X|X|X|  -> 3rd bit of command opcode
0147          ;          X|Y|X|X|X|X|X|X|  -> 2nd bit of command opcode
0148          ;          Y|X|X|X|X|X|X|X|  -> msb of command op code
0149          ;
0150          ;addr reg. 6/7/8 bits of address if necessary.
0151
0152          dout10
0153          rlf      indir          ;rotate thru carry
0154          bcf      serial,din      ;set output to 0
0155          btfscc  status,carry     ;set?
0156          bsf      serial,din      ;else set output to 1
0157          clkkit
0158          ;
0159          m
0160          m
0161          m          bsf      serial,clock      ;Clock (CLK) = '1'.
0162          m
0163          m          nop
0164          m          ;Adjust the number of nop instructions
0165          m          ;between the assertion and de-assertion of
0166          m          ;CLK in proportion to the PIC operating
0167          m          ;frequency. Refer to the 93CX6 data for the
0168          m          ;minimum CLK period.
0169          0007  0465          bcf      serial,clock      ;Clock (CLK) = '0'.
0170          0008  0360          dout9
0171          0009  0425          rlf      indir          ;rotate thru carry
0172          000A  0603          bcf      serial,din      ;set output to 0
0173          000B  0525          btfscc  status,carry     ;set?
0174          ;
0175          m          bsf      serial,din      ;else set output to 1
0176          m          clkkit
0177          m
0178          m
0179          m          bsf      serial,clock      ;Clock (CLK) = '1'.
0180          m
0181          m          nop
0182          m          ;Adjust the number of nop instructions
0183          m          ;between the assertion and de-assertion of
0184          m          ;CLK in proportion to the PIC operating
0185          m          ;frequency. Refer to the 93CX6 data for the
0186          m          ;minimum CLK period.
0187          000E  0465          bcf      serial,clock      ;Clock (CLK) = '0'.
0188          000F  02A4          incf      fsr          ;inc pointer
0189          ;
0190          0010  0C08          dout8
0191          0011  003F          movlw   8          ;Initialize loop counter.
0192          ;
0193          0012  0425          d_o_8  bcf      serial,din      ;Assume that the bit to be transferred is a
0194          0013  0360          rlf      indir          ;'0'. Hence, de-assert DI.
0195          0014  0603          btfscc  status,carry     ;Rotate the actual bit to be transferred into
0196          ;
0197          ;the carry bit.
0198          ;Test the carry, if our assumption was
0199          ;correct, skip the next instruction.

```

Interfacing 93 Series Serial EEPROMs

```

0196 0015 0525      bsf    serial,din    ;No, actual bit was a '1'. Assert DI.
0197                clkkit    ;Clock the 93CX6.
0198                m
0199                m
0200 0016 0565      m        bsf    serial,clock    ;of the 93CX6 device.
0201                m        ;Clock (CLK) = '1'.
0202 0017 0000      m        nop
0203                m        ;Adjust the number of nop instructions
                                ;between the assertion and de-assertion of

0204                m        ;CLK in proportion to the PIC operating
0205                m        ;frequency. Refer to the 93CX6 data for the
0206                m        ;minimum CLK period.
0207                m
0208 0018 0465      m        bcf    serial,clock    ;Clock (CLK) = '0'.
0209 0019 02FF      m        decfsz cnt    ;Repeat until cnt = 0.
0210 001A 0A12      m        goto   d_o_8    ;Cnt still > 0.
0211 001B 0360      m        rlf    indir    ;Restore register to its original condition.
0212 001C 0425      m        bcf    serial,din    ;make sure din is low
0213 001D 0800      m        retlw  no_err    ;Exit with good status.
0214
0215                ;*****
0216                ;*                               DIN8
                                ;*
                                ;*****
0217                ;*****
0218                ;Din8 will input 8 bits of data from the
0219                ;93CX6. Before calling this routine, the FSR
0220                ;must point to the register being used to
0221                ;hold the incoming data.
0222                din8
0223                ;set up the RAL as a input before proceeding
0224 001E 0C02      m        movlw  b'00000010'    ;set up porta
0225 001F 0005      m        tris   serial
0226 0020 0C08      m        movlw  8
0227 0021 003F      m        movwf  cnt
0228
0229                d_i_8
0230 0022 0360      m        rlf    indir    ;Make room for the incoming bit in the
0231                m        ;destination register.
0232 0023 0400      m        bcf    indir,0    ;Assume that the incoming bit is a '0' and
0233                m        ;clear the LSB of the destination register.
0234                m        clkkit    ;Clock a bit in the 93CX6.
0235                m
0236                m        ;of the 93CX6 device.
0237 0024 0565      m        bsf    serial,clock    ;Clock (CLK) = '1'.
0238                m
0239 0025 0000      m        nop
0240                m        ;Adjust the number of nop instructions
0241                m        ;between the assertion and de-assertion of
0242                m        ;CLK in proportion to the PIC operating
0243                m        ;frequency. Refer to the 93CX6 data for the
0244                m        ;minimum CLK period.
0245                m
0245 0026 0465      m        bcf    serial,clock    ;Clock (CLK) = '0'.
0246 0027 0625      m        btfsz  serial,dout    ;Test the incoming bit, if our assumption
0247                m        ;was correct, skip the next instruction.
0248 0028 0500      m        bsf    indir,0    ;No, actual bit is a '1'. Set the LSB of the
0249                m        ;destination register.
0250 0029 02FF      m        decfsz cnt    ;Repeat until cnt = 0.
0251 002A 0A22      m        goto   d_i_8    ;Cnt still > 0.
0252                m        ;setup RAL back to output
0253 002B 0C00      m        movlw  0
0254 002C 0005      m        tris   serial    ;set RAL as output
0255 002D 0800      m        retlw  no_err    ; /
                                ;Exit with good status.
0256
0257                ;*****
0258                ;*                               RDYCHK
                                ;*
                                ;*****
0259                ;*****
0260                ;Rdychk will read the 93CX6 READY/BUSY status
0261                ;and wait for RDY status within the allotted
0262                ;number of processor cycles. If RDY status

```



Interfacing 93 Series Serial EEPROMs

```

0263                                     ;is not present after this set period, the
0264                                     ;routine will return with an error status.
0265
0266         rdychk
0267                                     ;set up RAL as a input before proceeding
0268     002E  0C02             movlw  b'00000010'      ;set up porta
0269     002F  0005             tris   serial         ;
0270     0030  0C20             movlw  tries          ;Initialize time-out counter
0271     0031  003E             movwf  cnt           ;
0272     0032  007F             clrfs  cnt           ;
0273                                     ;De-select the 93CX6.
0274
0275     0033  0445             m       bcf   serial,cs      ;Chip Select (CS) = '0' to de-select
0276                                     ;the device.
0277
0278     ;                       nop                    ;NOTE: Check the 93CX6 data sheet for
0279                                     ;minimum CS low time. Depending upon
0280                                     ;processor frequency, a nop(s) may be
0281                                     ;between the assertion and de-assertion of
0282                                     ;Chip Select.
0283
0284                                     sel             ;Re-select the 93CX6.
0285
0286     0034  0545             m       bsf   serial,cs      ;Chip Select (CS) = '1' to select
0287     0035  0625             notrdy  btfsc  serial,dout    ;If DO is a '0', 93CX6 has yet to completed
0288                                     ;the last operation (still busy).
0289     0036  0A3E             goto   rdynoerr      ;skip to no error
0290     0037  02FF             decfsc  cnt          ;No, not yet ready. Decrement the LSB of our
0291                                     ;16 bit timer and check for expiration.
0292     0038  0A35             goto   notrdy       ;Still some time left. Try again.
0293     0039  02FE             decfsc  cnt         ;Least significant byte expired - decrement
0294                                     ;and check for expiration of the MSB.
0295     003A  0A35             goto   notrdy       ;Still some time left. Try again.
0296                                     ;setup RAL back to output
0297     003B  0C00             movlw  0            ;set RAL as output
0298     003C  0005             tris   serial         ;
0299     003D  0801             retlw  error        ;RDY status was not present in the allotted
0300                                     ;time, return with error status.
0301
0302         rdynoerr
0303                                     ;setup RAL back to output
0304     003E  0C00             movlw  0            ;set porta as output
0305     003F  0005             tris   serial         ;
0306     0040  0800             retlw  no_err       ;
0307
0308     ;*****
0309     ;*                               SEE
0310     ;*
0311     ;*****
0312
0313                                     ;See will control the entire operation of a
0314                                     ;93CX6 device. Prior to calling the routine,
0315                                     ;load a valid command/memory address into
0316                                     ;location cmd, and for WRITE or WRAL
0317                                     ;commands, load registers highb and lowb with
0318                                     ;16 bits of write data. Upon exit, the W
0319                                     ;register will contain the completion status.
0320                                     ;Only 93CX6 instructions which require a
0321                                     ;status check can return with an error as the
0322                                     ;completion status. The values that denote
0323                                     ;the completion status are defined as
0324                                     ;variables 'error' and 'no_err' in the
0325                                     ;general assignments section.
0326
0327         see
0328     0041  021A             movf   cmd,w         ;save cmd
0329     0042  003E             movwf  temp_cmd      ;
0330     0043  021B             movf   addr,w        ;save addr
0331     0044  003F             movwf  temp_addr     ;
0332     0045  0C1A             movlw  cmd           ;Load W with the location of the cmd
0333                                     ;register.

```

Interfacing 93 Series Serial EEPROMs

```

0332 0046 0024      movwf  fsr          ;Transfer that information into the File
0333                                     ;Select Register. The fsr now points to
0334                                     ;location cmd.
0335                                     ;Select the 93CX6.
0336                                     m
0337 0047 0545      bsf     serial,cs    ;Chip Select (CS) = '1' to select
0338                                     strtbt                ;Send a start bit.
0339                                     m
0340 0048 0525      m       bsf     serial,din   ;Start Bit = '1'.
0341                                     clklt                ;Clock it out.
0342                                     m
0343                                     m
0344 0049 0565      m       bsf     serial,clock ;of the 93CX6 device.
0345                                     m                   ;Clock (CLK) = '1'.
0346 004A 0000      m       nop
0347                                     m                   ;Adjust the number of nop instructions
0348                                     m                   ;between the assertion and de-assertion of
0349                                     m                   ;CLK in proportion to the PIC operating
0350                                     m                   ;frequency. Refer to the 93CX6 data for the
0351                                     m                   ;minimum CLK period.
0352 004B 0465      bcf     serial,clock    ;Clock (CLK) = '0'.
0353                                     ;
0354 004C 06FA      btfscl cmd,7           ;bit 7 = 0?
0355 004D 0A73      goto    scal0                ;xfer 10 bit cmd/adr
0356                                     ;
0357                                     goto    set_cmd_addr   ;no then set cmd/addr
0358 004E 07DA      btfscl cmd,6           ;bit 6 = 0 ?
0359 004F 0A75      goto    scl0                ;xfer 10 bit cmd/adr
0360                                     ;
0361 0050 0A73      goto    set_cmd                ;yes then set cmd
0362                                     goto    scl0                ;xfer 10 bit cmd/adr
0363                                     ;
0364 see1          goto    set_cmd_addr   ;else set cmd/addr
0365 0051 021E      movf    temp_cmd,w        ;retore cmd
0366 0052 003A      movwf  cmd                ; /
0367 0053 021F      movf    temp_addr,w      ;restore addr
0368 0054 003B      movwf  addr                ; /
0369 0055 06DA      btfscl cmd,6           ;Check for a WRITE or ERASE command.
0370 0056 0A5F      goto    see2                ;Yes, parse the command further.
0371 0057 06FA      btfscl cmd,7           ;Check for a READ command.
0372 0058 0A69      goto    read                ;Yes, process READ command.
0373 0059 06BA      btfscl cmd,5           ;Check for a EWEN or ERAL command.
0374 005A 0A66      goto    see3                ;Yes, parse the command further.
0375 005B 069A      btfscl cmd,4           ;Check for a WRAL command.
0376 005C 0A6E      goto    write_            ;Yes, process WRITE/WRAL command.
0377
0378                                     exit_   dsel                ;No further processing required; 93CX6
0379                                     m
0380 005D 0445      m       bcf     serial,cs    ;Chip Select (CS) = '0' to de-select
0381                                     m                   ;the device.
0382                                     m                   ;command completed.
0383                                     ;
0384 005E 0800      retlw  no_err            ;Return with good completion status.
0385
0386 005F 07FA      see2    btfscl cmd,7           ;Check for a ERASE command.
0387 0060 0A6E      goto    write_            ;No, process WRITE command.
0388 0061 092E      exit2_ call    rdychk        ;ERASE command requires a status check.
0389                                     m                   ;De-select the 93CX6.
0390                                     m
0391 0062 0445      m       bcf     serial,cs    ;Chip Select (CS) = '0' to de-select
0392                                     m                   ;the device.
0393 0063 01E2      addwf  pc                ;Compute completion status from results of
0394                                     m                   ;status check.
0395 0064 0800      retlw  no_err            ;Return with good completion status.
0396 0065 0801      retlw  error            ;Return with bad completion status.
0397
0398 0066 069A      see3    btfscl cmd,4           ;Check for a EWEN command
0399 0067 0A5D      goto    exit_            ;Yes, no further processing required, exit
0400                                     m                   ;now.
0401 0068 0A61      goto    exit2_          ;No, ERAL command which requires a status

```

Interfacing 93 Series Serial EEPROMs

```

0402                                     ;check.
0403
0404 0069 02A4      read_  incf  fsr      ;Increment the File Select Register to point
0405                                     ;to the register receiving the upper byte of
0406                                     ;the incoming 93CX6 data word.
0407 006A 091E      call   din8      ;Input the upper byte.
0408 006B 02A4      incf  fsr      ;Increment the File Select Register to point
0409                                     ;to the register receiving the lower byte.
0410 006C 091E      call   din8      ;Input 8 more bits.
0411 006D 0A5D      goto   exit_     ;No further processing required, exit now.
0412
0413 006E 02A4      write_ incf  fsr      ;Increment the File Select Register to point
0414                                     ;to the upper byte of the 16 bit 93CX6 data
0415                                     ;word to be transmitted.
0416 006F 0910      call   dout8     ;Output that byte.
0417 0070 02A4      incf  fsr      ;Increment the File Select Register to point
0418                                     ;to the lower byte.
0419 0071 0910      call   dout8     ;Output the lower byte of the 16 bit 93CX6
0420                                     ;data word.
0421 0072 0A61      goto   exit2_    ;Exit with a status check.
0422
0423      ;
0424      ;
0425      ;
0426      scala0
0427 0073 0901      call   dout10    ;output cmd reg
0428 0074 0A51      goto   seel     ;return
0429      ;
0430      ;
0431      ;
0432      ;
0433      ;
0434      sc10
0435 0075 037A      rlf   cmd      ;rotate cmd
0436 0076 035A      rlf   cmd,w    ;left twice
0437 0077 003B      movwf addr     ;save in addr
0438 0078 007A      clrf  cmd      ;clear command
0439 0079 0901      call   dout10    ;xfer 10 bits
0440 007A 0A51      goto   seel     ;return
0441      ;
0442      ;*****
0443      ;*                               Test Program                               *
0444      ;*                               *****                               *
0445      ;*****
0446      main                                     ;We've include a sample program to exercise
0447                                     ;the PIC to 93C66 interface using a simple
0448                                     ;erase, write and verify routine.
0449                                     ;8 bit organization has been used
0450                                     ;with a 3 wire interface.
0451 007B 0065      clrf  serial     ;Clear the port tied to the 93C66 device.
0452 007C 0CF0      movlw b'11110000' ;Intialize the data direction register for
0453 007D 0005      tris  serial     ;that port.
0454
0455 007E 0C30      movlw ewen      ;Load W with the Erase/Write Enable command.
0456 007F 003A      movwf cmd      ;Transfer W into cmd register.
0457 0080 0941      call  see      ;Enable the 93C66 device.
0458
0459 0081 0C20      movlw eral      ;Load W with the Erase All command.
0460 0082 003A      movwf cmd      ;Transfer W into cmd register.
0461 0083 0941      call  see      ;Erase the 93C66.
0462 0084 0F01      xorlw error     ;Check completion status.
0463 0085 0643      btfsc status, zflag ;Test for error condition.
0464 0086 0AA6      goto  errloop   ;Yes, bad completion status, error-out.
0465
0466                                     ;Write loop:
0467 00AA          tstptrn      equ  0xAA      ;Define the test pattern to be written.
0468
0469 0087 0C40      movlw write     ;Load W with the Write command.
0470 0088 003A      movwf cmd      ;Transfer W into cmd register.

```

Interfacing 93 Series Serial EEPROMs

```

0471 0089 0CAA          movlw  tstptrn      ;Initialize the 93C66 data registers with
0472                                     ;write data.
0473 008A 003C          movwf  highb        ;load in high byte only
0474                                     ;since 8 bit low byte is ignored
0475 008B 007B          clrf   addr         ;start at addr 0
0476 008C 0941 test1    call   see          ;Write data word into 93C66 device.
0477 008D 0F01          xorlw  error        ;Check completion status.
0478 008E 0643          btfsz status,zflag ;Test for error condition.
0479 008F 0AA6          goto  errloop      ;Yes, bad completion status, error-out.
0480 0090 03FB          incfsz addr        ;No, increment the 8 bit memory address
0481                                     ;field.
0482 0091 0A8C          goto  test1        ;write another location
0483 0092 077A          btfss cmd,3        ;see if all done
0484 0093 0A95          goto  wrt_nxt_pg   ;no then write next page
0485 0094 0A97          goto  read_tst     ;read written data
0486                                     ;
0487 0095 057A          bsf   cmd,3        ;set page bit
0488 0096 0A8C          goto  test1        ;No, write another location.

0489                                     ;
0490                                     ;Read loop:
0491 read_tst
0492 0097 0C80          movlw  read         ;Load W with the Read command.
0493 0098 003A          movwf  cmd          ;Transfer W into cmd register.
0494 0099 0941 test2    call   see          ;Read addressed data word from 93C66 device.
0495 009A 0CAA          movlw  tstptrn     ;Load W with the pattern written.
0496 009B 009C          subwf  highb,w     ;Verify the data read against what was
0497                                     ;written.
0498 009C 0743          btfss status,zflag ;Same?
0499 009D 0AA6          goto  errloop      ;No, error-out.
0500 009E 03FB          incfsz addr        ;Yes, both byte correct, increment the 8 bit
0501                                     ;memory address field.
0502 009F 0A99          goto  test2        ;do next byte
0503 00A0 077A          btfss cmd,3        ;check page bit
0504 00A1 0AA3          goto  rd_nxt_pg    ;no then chk next page
0505 00A2 0AA5          goto  alloc        ;all done!!!
0506 rd_nxt_pg
0507 00A3 057A          bsf   cmd,3        ;set page bit
0508 00A4 0A99          goto  test2        ;No, read another location.
0509                                     ;
0510 00A5 0AA5 alloc     goto  alloc        ;Home safe!
0511                                     ;
0512                                     ;
0513 errloop
0514 00A6 0AA6          goto  errloop
0515                                     ;
0516                                     ;
0517                                     ;
0518                                     ;KEY DEFINITIONS
0519                                     ;
0520 0000          ORG    PIC54
0521 01FF 0A00          SYS_RESET GOTO  START
0522                                     ;
0523                                     ;
0524 0000          END

```

Interfacing 93 Series Serial EEPROMs

Appendix B

Microchip Technology PIC16C5X Assembler 3.03A Wed Feb 26 11:11:15 1992 Page 1

```
Line  PC  Opcode
0001          ;*****
0002          ;*                MPALC Directives Section
0003          ;*
0004          ;*****
0005      LIST      P=16C54,N=40,C=132
0006
0007
0008          ;*****
0009          ;*                Register Assignments                *
0010          ;*****
0011
0012      0000  indir  equ    0x00      ;Use this register as source/destination for
0013          ;indirect addressing.
0014      0002  pc     equ    0x02      ;PIC16/17 Program Counter.
0015      0003  status equ    0x03      ;PIC16/17 Status Register.
0016      0004  fsr   equ    0x04      ;File Select Register.
0017      0005  serial equ    0x05      ;Port used for 93C46 control. Since Port A
0018          ;is 4 bits wide, we'll use 3 or 4 pins of Port A.
0019
0020          ;The following three registers must be
0021          ;located consecutively in memory.
0022      0010  cmd   equ    0x10      ;This register contains the 2 bit 93C46
0023          ;command is the upper 2 bit positions and
0024          ;memory address in the lower 6.
0025      0011  highb  equ    0x11      ;Used in read/write routines to store the
0026          ;upper byte of a 16 bit 93C46 data word.
0027      0012  lowb  equ    0x12      ;Used in read/write routines to store the
0028          ;lower byte of a 16 bit 93C46 data word.
0029
0030      0013  cnthi  equ    0x13      ;Used as the upper byte of a sixteen bit loop
0031          ;counter in RDYCHK routine.
0032      0014  cnt   equ    0x14      ;Used as the lower byte of a sixteen bit loop
0033          ;counter in RDYCHK routine, and elsewhere as
0034          ;an eight bit counter.
0035
0036          ;*****
0037          ;*  Bit Assignments:  The following assignment is for 3-wire setup.
0038          ;*  For four wire setup, assign dout equ 2.
0039
0040      0000  carry  equ    0          ;Carry Flag of Status Register.
0041      0002  zflag  equ    2          ;Zero Flag of Status Register.
0042
0043      0000  cs     equ    0          ;Port pin tied to CS on 93C46.
0044      0001  din   equ    1          ;Port pin tied to DI on 93C46. 3-wire setup.
0045      0001  dout  equ    1          ;Port pin tied to DO on 93C46. 3-wire setup.
0046      0003  clock equ    3          ;Port pin tied to CLK on 93C46.
0047
0048          ;*****
0049          ;*                General Assignments                *
0050          ;*****
0051
0052      0000  no_err equ    0          ;
0053      0001  error  equ    1          ;
0054      0004  tries  equ    0x04      ;After issuing a WRITE, ERASE, ERAL, or WRAL
0055          ;command, the approximate number of machine
0056          ;cycles X 256 to wait for the RDY status.
0057          ;This value must be adjusted for operating
0058          ;frequencies other than 4 MHz.
0059
0060      0080  read   equ    0x80      ;93C46 Read command.
0061      0040  write  equ    0x40      ;93C46 Write command.
0062      00C0  erase  equ    0xC0      ;93C46 Erase command.
0063      0030  ewen  equ    0x30      ;93C46 Erase/Write Enable command.
```


Interfacing 93 Series Serial EEPROMs

```

0064      0000      ewds      equ      0x00      ;93C46 Erase/Write Disable command.
0065      0020      eral      equ      0x20      ;92CXX Erase All command.
0066      0010      wral      equ      0x10      ;92CXX Write All command.
0067
0068      ;*****
0069      ;*          Macro Definitions          *
0070      ;*****
0071
0072      sel          MACRO          ;Selects the 93C46 device.
0073      bsf          serial,cs      ;Chip Select (CS) = '1' to select the device.
0074      ENDM
0075
0076      dsel         MACRO          ;De-select the 93C46 device.
0077      bcf          serial,cs      ;Chip Select (CS) = '0' to de-select the
0078      ;device.
0079      ENDM
0080
0081      strtbt      MACRO          ;Issue the Start Bit to the 93C46.
0082      bsf          serial,din     ;Start Bit = '1'.
0083      clkkit      ;Clock it out.
0084      ENDM
0085
0086      clkkit      MACRO          ;Clocks a serial data bit into or out of the
0087      ;93C46 device.
0088      bsf          serial,clock   ;Clock (CLK) = '1'.
0089
0090      nop          ;Adjust the number of nop instructions
0091      ;between the assertion and de-assertion of
0092      ;CLK in proportion to the PIC operating
0093      ;frequency. Refer to the 93C46 data for the
0094      ;minimum CLK period.
0095
0096      bcf          serial,clock   ;Clock (CLK) = '0'.
0097      ENDM
0098
0099      ;*****
0100      ;*          Power-On/Reset Entry Point          *
0101      ;*****
0102
0103      0000      reset_      org      0x1FF
0104      01FF      0A57      goto      main
0105
0106      ;*****
0107      ;*          93C46 Routines          *
0108      ;*****
0109      0000      org      0x000      ;Locate all subroutines in the lower half of
0110      ;a Program Memory Page.
0111
0112      ;*****
0113      ;*          DOUT8          *
0114      ;*****
0115      ;Dout8 will output 8 bits of data to the
0116      ;93C46. Before calling this routine, the FSR
0117      ;must point to the byte being transmitted.
0118
0119      0000      0C08      dout8      movlw   0x08      ;Initialize loop counter.
0120      0001      0034      movwf   cnt          ;
0121
0122      0002      0425      d_o_8      bcf     serial,din ;Assume that the bit to be transferred is a
0123      ;'0'. Hence, de-assert DI.
0124      0003      0360      rlf     indir   ;Rotate the actual bit to be transferred into
0125      ;the carry bit.
0126      0004      0603      btfscl status,carry ;Test the carry, if our assumption was
0127      ;correct, skip the next instruction.
0128      0005      0525      bsf     serial,din ;No, actual bit was a '1'. Assert DI.
0129      clkkit      ;Clock the 93C46.
0130      m
0131      m          ;93C46 device.
0132      0006      0565      m          bsf     serial,clock ;Clock (CLK) = '1'.
0133      m
0134      0007      0000      m          nop          ;Adjust the number of nop instructions

```

Interfacing 93 Series Serial EEPROMs

```

0135          m          ;between the assertion and de-assertion of
0136          m          ;CLK in proportion to the PIC operating
0137          m          ;frequency. Refer to the 93C46 data for the
0138          m          ;minimum CLK period.
0139          m
0140 0008 0465          bcf    serial,clock  ;Clock (CLK) = '0'.
0141 0009 02F4          decfsz cnt          ;Repeat until cnt = 0.
0142 000A 0A02          goto   d_o_8          ;Cnt still > 0.
0143 000B 0360          rlf    indir          ;Restore register to its original condition.
0144 000C 0425          bcf    serial,din    ;make din low before return
0145 000D 0800          retlw  no_err          ;Exit with good status.
0146
0147          ;*****
0148          ;*                DIN8                *
0149          ;*****
0150          ;Din8 will input 8 bits of data from the
0151          ;93C46. Before calling this routine, the FSR
0152          ;must point to the register being used to
0153          ;hold the incoming data.
0154          din8
0155          ;*****
0156          ;set up RAL as a input before proceeding
0157 000E 0C02          movlw  b'00000010'          ;set up PORTA
0158 000F 0005          tris   serial          ; /
0159          ;*****
0160 0010 0C08          movlw  0x08          ;Initialize loop counter.
0161 0011 0034          movwf  cnt          ;
0162
0163          d_i_8      clklt          ;Clock a bit out of the 93C46.
0164          m
0165          m          ;93C46 device.
0166 0012 0565          bsf    serial,clock  ;Clock (CLK) = '1'.
0167          m
0168 0013 0000          nop          ;Adjust the number of nop instructions
0169          m          ;between the assertion and de-assertion of
0170          m          ;CLK in proportion to the PIC operating
0171          m          ;frequency. Refer to the 93C46 data for the
0172          m          ;minimum CLK period.
0173          m
0174 0014 0465          bcf    serial,clock  ;Clock (CLK) = '0'.
0175 0015 0360          rlf    indir          ;Make room for the incoming bit in the
0176          ;destination register.
0177 0016 0400          bcf    indir,0        ;Assume that the incoming bit is a '0' and
0178          ;clear the LSB of the destination register.
0179 0017 0625          btfsz  serial,dout    ;Test the incoming bit, if our assumption
0180          ;was correct, skip the next instruction.
0181 0018 0500          bsf    indir,0        ;No, actual bit is a '1'. Set the LSB of the
0182          ;destination register.
0183 0019 02F4          decfsz cnt          ;Repeat until cnt = 0.
0184 001A 0A12          goto   d_i_8          ;Cnt still > 0
0185          ;*****
0186          ;Restore RAL back as output
0187 001B 0C00          movlw  0          ;set RAL as output
0188 001C 0005          tris   serial          ; /
0189          ;*****
0190 001D 0800          retlw  no_err          ;Exit with good status.
0191
0192          ;*****
0193          ;*                RDYCHK                *
0194          ;*****
0195          ;Rdychk will read the 93C46 READY/BUSY status
0196          ;and wait for RDY status within the allotted
0197          ;number of processor cycles. If RDY status
0198          ;is not present after this set period, the
0199          ;routine will return with an error status.
0200
0201          rdychk
0202          ;*****
0203          ;set up RAL as a input before proceeding
0204 001E 0C02          movlw  b'00000010'          ;set up PORTA

```

Interfacing 93 Series Serial EEPROMs

```

0205 001F 0005          tris    serial    ; /
0206                  ;*****
0207 0020 0C04          movlw   tries      ;Initialize time-out counter.
0208 0021 0033          movwf   cnthi     ;
0209 0022 0074          clrf    cnt       ;
0210                  dsel     ;De-select the 93C46.
0211                  m
0212 0023 0405 m        bcf     serial,cs  ;Chip Select (CS) = '0' to de-select the
0213                  ;device.
0214
0215                  ;
0216                  nop          ;NOTE: Check the 93C46 data sheet for
0217                  ;minimum CS low time. Depending upon
0218                  ;processor frequency, a nop(s) may be
0219                  ;between the assertion and de-assertion of
0220                  ;Chip Select.
0221
0222                  sel          ;Re-select the 93C46.
0223                  m
0224 0024 0505          bsf     serial,cs  ;Chip Select (CS) = '1' to select the device.
0225 0025 0625 notrdy     btfsz   serial,dout ;If DO is a '0', 93C46 has yet to completed
0226                  ;the last operation (still busy).
0227 0026 0A2E          goto    rdynoerr   ;skip to no error
0228 0027 02F4          decfsz  cnt       ;No, not yet ready. Decrement the LSB of our
0229                  ;16 bit timer and check for expiration.
0230 0028 0A25          goto    notrdy     ;Still some time left. Try again.
0231 0029 02F3          decfsz  cnthi     ;Least significant byte expired - decrement
0232                  ;and check for expiration of the MSB.
0233 002A 0A25          goto    notrdy     ;Still some time left. Try again.
0234                  ;*****
0235                  ;Restore RAL back as output
0236 002B 0C00          movlw   0         ;set RAL as output
0237 002C 0005          tris    serial    ; /
0238 002D 0801          retlw  error     ;RDY status was not present in the allotted
0239                  ;time, return with error status.
0240                  rdynoerr
0241                  ;*****
0242                  ;Restore RAL back as output
0243 002E 0C00          movlw   0         ;set RAL as output
0244 002F 0005          tris    serial    ; /
0245                  ;*****
0246 0030 0800          retlw  no_err
0247                  ;
0248                  ;
0249                  ;*****
0250                  ;* SEE *
0251                  ;*****
0252
0253                  ;See will control the entire operation of a
0254                  ;93C46 device. Prior to calling the routine,
0255                  ;load a valid command/memory address into
0256                  ;location cmd, and for WRITE or WRAL
0257                  ;commands, load registers highb and lowb with
0258                  ;16 bits of write data. Upon exit, the W
0259                  ;register will contain the completion status.
0260                  ;Only 93C46 instructions which require a
0261                  ;status check can return with an error as the
0262                  ;completion status. The values that denote
0263                  ;the completion status are defined as
0264                  ;variables 'error' and 'no_err' in the
0265                  ;general assignments section.
0266
0267 0031 0C10 see        movlw   cmd       ;Load W with the location of the cmd
0268                  ;register.
0269 0032 0024          movwf   fsr       ;Transfer that information into the File
0270                  ;Select Register. The fsr now points to
0271                  ;location cmd.
0272                  sel          ;Select the 93C46.
0273                  m
0274 0033 0505          bsf     serial,cs  ;Chip Select (CS) = '1' to select the device.
0275                  strtbt    ;Send a start bit.

```



Interfacing 93 Series Serial EEPROMs

```

0276          m
0277 0034 0525 m      bsf    serial,din      ;Start Bit = '1'.
0278          m      clklt          ;Clock it out.
0279          m
0280          m      ;93C46 device.
0281 0035 0565 m      bsf    serial,clock    ;Clock (CLK) = '1'.
0282          m
0283 0036 0000 m      nop          ;Adjust the number of nop instructions
0284          m      ;between the assertion and de-assertion of
0285          m      ;CLK in proportion to the PIC operating
0286          m      ;frequency. Refer to the 93C46 data for the
0287          m      ;minimum CLK period.
0288          m
0289 0037 0465 m      bcf    serial,clock    ;Clock (CLK) = '0'.
0290 0038 0900 m      call   dout8          ;Transmit the 2 bit command and six bit
0291          m      ;address.
0292 0039 06D0 m      btfs   cmd,6          ;Check for a WRITE or ERASE command.
0293 003A 0A43 m      goto   see2          ;Yes, parse the command further.
0294 003B 06F0 m      btfs   cmd,7          ;Check for a READ command.
0295 003C 0A4D m      goto   read_          ;Yes, process READ command.
0296 003D 06B0 m      btfs   cmd,5          ;Check for a EWEN or ERAL command.
0297 003E 0A4A m      goto   see3          ;Yes, parse the command further.
0298 003F 0690 m      btfs   cmd,4          ;Check for a WRAL command.
0299 0040 0A52 m      goto   write_         ;Yes, process WRITE/WRAL command.
0300
0301          exit_dsel          ;No further processing required; 93C46
0302          m
0303 0041 0405 m      bcf    serial,cs      ;Chip Select (CS) = '0' to de-select the
0304          m      ;device.
0305          m      ;command completed.
0306 0042 0800 m      retlw  no_err        ;Return with good completion status.
0307
0308 0043 07F0 see2    btfs   cmd,7          ;Check for a ERASE command.
0309 0044 0A52 m      goto   write_         ;No, process WRITE command.
0310 0045 091E exit2_ call   rdychk        ;ERASE command requires a status check.
0311          m      dsel          ;De-select the 93C46.
0312          m
0313 0046 0405 m      bcf    serial,cs      ;Chip Select (CS) = '0' to de-select the
0314          m      ;device.
0315 0047 01E2 m      addwf  pc          ;Compute completion status from results of
0316          m      ;status check.
0317 0048 0800 m      retlw  no_err        ;Return with good completion status.
0318 0049 0801 m      retlw  error        ;Return with bad completion status.
0319
0320 004A 0690 see3    btfs   cmd,4          ;Check for a EWEN command.
0321 004B 0A41 m      goto   exit_          ;Yes, no further processing required, exit
0322          m      ;now.
0323 004C 0A45 m      goto   exit2_         ;No, ERAL command which requires a status
0324          m      ;check.
0325
0326 004D 02A4 read_   incf   fsr          ;Increment the File Select Register to point
0327          m      ;to the register receiving the upper byte of
0328          m      ;the incoming 93C46 data word.
0329 004E 090E m      call   din8          ;Input the upper byte.
0330 004F 02A4 m      incf   fsr          ;Increment the File Select Register to point
0331          m      ;to the register receiving the lower byte.
0332 0050 090E m      call   din8          ;Input 8 more bits.
0333 0051 0A41 m      goto   exit_          ;No further processing required, exit now.
0334
0335 0052 02A4 write_  incf   fsr          ;Increment the File Select Register to point
0336          m      ;to the upper byte of the 16 bit 93C46 data
0337          m      ;word to be transmitted.
0338 0053 0900 m      call   dout8        ;Output that byte.
0339 0054 02A4 m      incf   fsr          ;Increment the File Select Register to point
0340          m      ;to the lower byte.
0341 0055 0900 m      call   dout8        ;Output the lower byte of the 16 bit 93C46
0342          m      ;data word.
0343 0056 0A45 m      goto   exit2_         ;Exit with a status check.
0344
0345          ;*****

```

Interfacing 93 Series Serial EEPROMs

```

0346          ;*                               Test Program                               *
0347          ;*****
0348          main                               ;We've include a sample program to exercise
0349                                               ;the PIC to 93C46 interface using a simple
0350                                               ;erase, write and varify routine.
0351
0352          0057 0065          clrfs          serial          ;Clear the port tied to the 93C46 device.
0353          0058 00F4          movlw         b'11110100'       ;Intialize the data direction register for
0354          0059 0005          tris          serial          ;that port.
0355
0356          005A 0C30          movlw         ewen          ;Load W with the Erase/Write Enable command.
0357          005B 0030          movwf         cmd          ;Transfer W into cmd register.
0358          005C 0931          call          see          ;Enable the 93C46 device.
0359
0360          005D 0C20          movlw         eral          ;Load W with the Erase All command.
0361          005E 0030          movwf         cmd          ;Transfer W into cmd register.
0362          005F 0931          call          see          ;Erase the 93C46.
0363          0060 0F01          xorlw        error         ;Check completion status.
0364          0061 0643          btfsc        status, zflag ;Test for error condition.
0365          0062 0A82          goto         errloop      ;Yes, bad completion status, error-out.
0366
0367                                               ;Write loop:
0368          001F loopcnt          equ          0x1F         ;Define an unused location for our test
0369                                               ;program loop counter.
0370          00AA tstptrn          equ          0xAA         ;Define the test pattern to be written.
0371
0372          0063 0C40          movlw         .64         ;Initialize that counter.
0373          0064 003F          movwf        loopcnt      ;
0374          0065 0C40          movlw         write       ;Load W with the Write command.
0375          0066 0030          movwf        cmd          ;Transfer W into cmd register.
0376          0067 0CAA          movlw         tstptrn     ;Intialize the 93C46 data registers with
0377                                               ;write data.
0378
0378          0068 0031          movwf        highb        ;
0379          0069 0032          movwf        lowb         ;
0380          006A 0931          call          see          ;Write data word into 93C46 device.
0381          006B 0F01          xorlw        error         ;Check completion status.
0382          006C 0643          btfsc        status,zflag ;Test for error condition.
0383          006D 0A82          goto         errloop      ;Yes, bad completion status, error-out.
0384          006E 02B0          incf         cmd          ;No, increment the 6 bit memory address
0385                                               ;field.
0386          006F 02FF          decfsz       loopcnt      ;Have we written all 64 locations?
0387          0070 0A6A          goto         test1        ;No, write another location.
0388
0389                                               ;Read loop:
0390
0391          0071 0C40          movlw         .64         ;Initialize loop counter.
0392          0072 003F          movwf        loopcnt      ;
0393          0073 0C80          movlw         read        ;Load W with the Read command.
0394          0074 0030          movwf        cmd          ;Transfer W into cmd register.
0395          0075 0931          call          see          ;Read addressed data word from 93C46 device.
0396          0076 0CAA          movlw         tstptrn     ;Load W with the pattern written.
0397          0077 0091          subwf        highb,0       ;Verify the data read against what was
0398                                               ;written.
0399          0078 0743          btfss       status,zflag  ;Same?
0400          0079 0A82          goto         errloop      ;No, error-out.
0401          007A 0CAA          movlw         tstptrn     ;Repeat with the lower byte read.
0402          007B 0092          subwf        lowb,0       ;
0403          007C 0743          btfss       status,zflag  ;Same?
0404          007D 0A82          goto         errloop      ;No, error-out.
0405          007E 02B0          incf         cmd          ;Yes, both byte correct, increment the 6 bit
0406                                               ;memory address field.
0407          007F 02FF          decfsz       loopcnt      ;Have we read all 64 locations?
0408          0080 0A75          goto         test2        ;No, read another location.
0409
0410          0081 0A81          allok          goto         allok          ;Home safe!
0411
0412          0082 0A82          errloop        goto         errloop        ;Bad news!
0413
0414          0000          END              ;Thats all folks!

```

Interfacing 93 Series Serial EEPROMs

NOTES:



Logic Powered Serial EEPROMs

Embedded applications increasingly want more integration and power, in less space for less cost. Using low power Serial EEPROMs (SEE) for application firmware, lookup tables, and microcode coupled with small footprints makes for permanent storage at respectable savings. One additional method of saving on the power budget is selectively powering off components when not needed, a basic for embedded power management. The low-power SEEs offered by Microchip Technology Inc., offer an additional benefit, powering the SEE from a microcontroller port. This allows the controller to manipulate not only when the SEE reads and writes, but also when it is powered on. Satellite communications use this technique to save power and total dose accumulation. We call this technique POWER PORT™. The microcontroller port must have sufficient Ioh (source current) to sustain the voltage and current for all memory functions, READ, ERASE, and WRITE. Obviously, not all memory or peripheral devices could be powered thusly, but Microchip's SEE devices will function in this environment.

The microcontroller, using its internal software and hardware decision functions, determines when it needs to communicate with the memory device, then acts accordingly. Any standard wake-up sequence will accomplish this task. The wake-up code needs only power up the memory and wait for the power to become stable before doing a read or write by driving the POWER PORT high. Then all serial communication executes normally. The SEEs are powered off for additional power savings and the data or code is utilized from RAM. Obviously, the port output must be allowed to settle, but normal operation of the output structures would guarantee that this would be met. The I/O port Tpd for the Microchip Technology Inc. PIC16C5X, is specified at 40ns maximum.

The 24LCXX and 93LCXX CMOS SEE series parts from Microchip were designed to achieve low current consumption across all ranges of operation.

The four primary Icc parameters for these products are:

Parameter	Conditions
Icc STANDBY	Not in an active operation while Vcc is supplied.
Icc READ	The part is in a READ operation.

Icc PEAK WRITE The BYTE / PAGE WRITE and ERASE operations have self timed cycles of 10 ms. A typical of 4 ms is the actual time of the operation. This is the amount of time when the Icc requires the most current (PEAK WRITE). The part is drawing STANDBY Icc during the remaining 6ms of the cycle.

Icc AVG WRITE The avg of the PEAK WRITE Icc and STANDBY Icc during the self-timed 10ms write cycle.

The attached characteristic curves (Figures A and B) indicate that Icc PEAK WRITE current consumes the most current. The worst case condition is at 6.0V and -40°C. The 24LCXX series parts draw a typical 3.2 mA and the 93LCXX series parts draw a typical of 2.0 mA. These low Icc characteristics offer a unique current saving benefit for battery applications. Figure C and D illustrate the sink and source current capabilities of the PIC16C5X family of controllers. It is clear from these characterization curves that the microcontroller can deliver sufficient current across all temperature ranges to power a SEE using the POWER PORT technique.

Figure E shows the connection scheme for the Microchip Technology Inc. PIC16C54. It should be noted that not all versions of competitive microcontrollers are capable of powering a device in this manner and the specific data sheets for the controller being considered must be consulted for maximum source current. The microcontroller port must be capable of sourcing sufficient current for the duration of the write cycle or 10ms, worse case. The peak write requirement for the 24LCXX product family is 3.2 mA at 5.5 Vdc (-40°C).

Listing A demonstrates the appropriate code sequences when using the PIC16C54 microcontroller. The sequences included are power control, start bit, stop bit, send and receive bit, Tx and Rx, and a general addressing routine.



Logic Powered Serial EEPROMs

FIGURE A - TYPICAL I_{cc} FOR 24LCXX

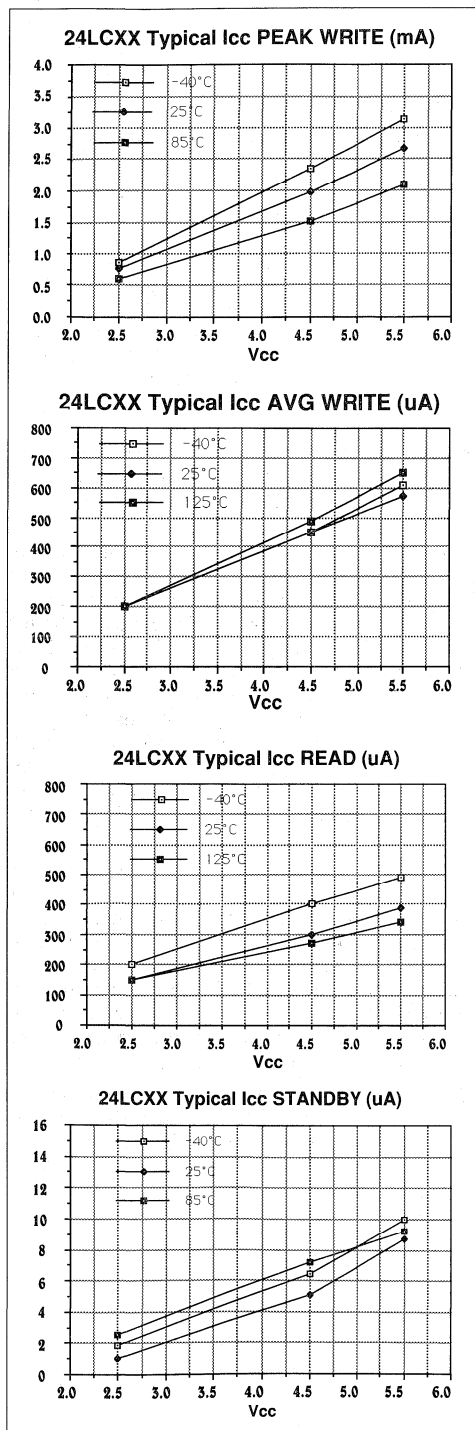
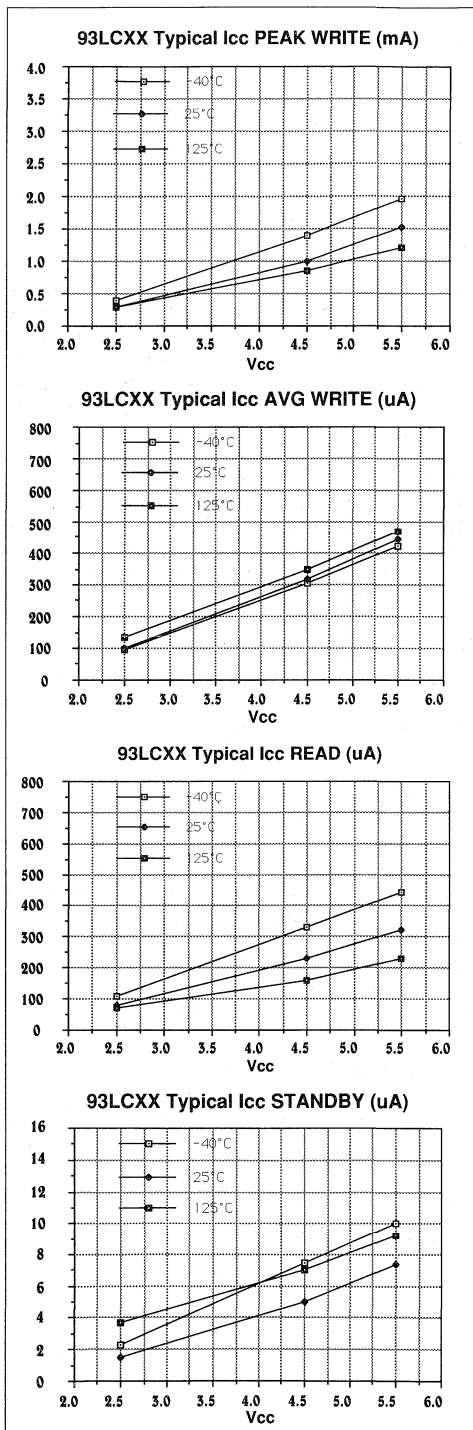


FIGURE B - TYPICAL I_{cc} FOR 93CXX



Logic Powered Serial EEPROMs

FIGURE C - PIC16C5X I_{OL} AT 5V

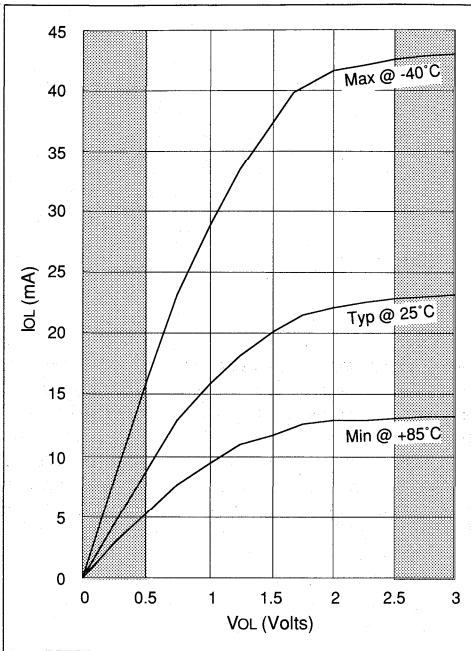


FIGURE D - PIC16C5X I_{OH} AT 5V

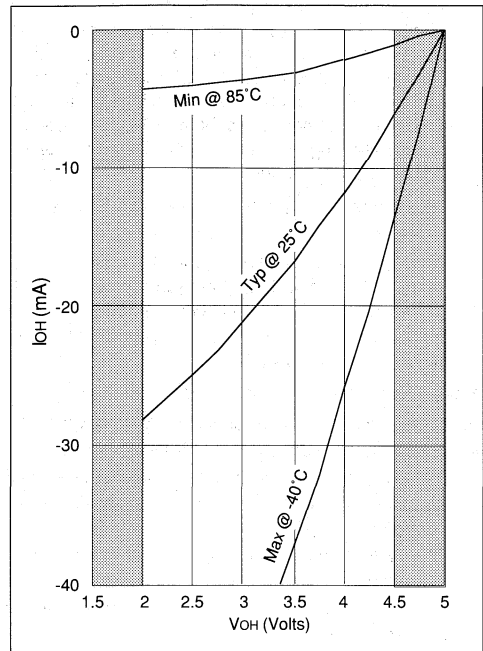
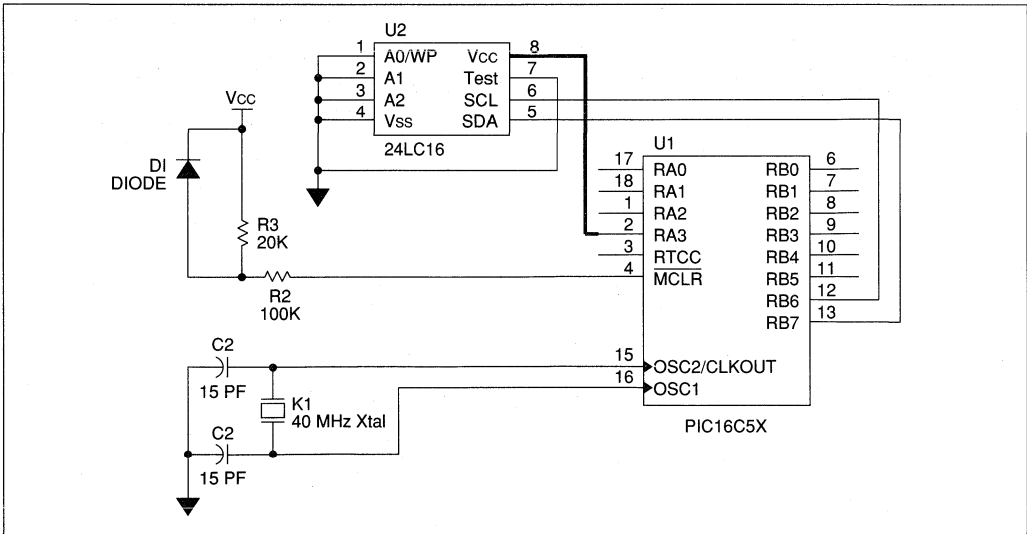


FIGURE E - 24LC16/PIC16C5X INTERFACE SCHEMATIC



Logic Powered Serial EEPROMs

The primary benefits of this application are:

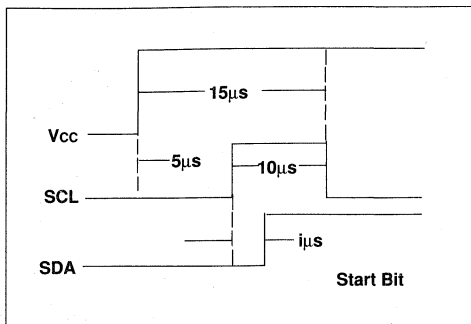
- The SEE is completely powered down to save power when the SEE is not executing an operation. This will directly effect the total system power consumption. This means that the SEE is in a total quiescent state and even the standby current savings is realized, greatly increasing usable battery life, and consequently allowing for a more sophisticated design on the same power budget.
- The very fast 5 μ s power-up time minimizes power-up delay.
- Since the serial operation is gated by a stable microcontroller VOH, risk of data being corrupted by a glitch is minimized. This, in effect, is a regulated Vcc supply and provides a reliable power source to ensure data integrity.

Several cautions need to be noted:

1. *Gang powering multiple devices must not exceed the I/O port IOH or capacitive load specifications.*
2. *The total power requirements vs. power budget must be considered, including the extra drain on the microcontroller.*
3. *The microcontroller Icc max must not be exceeded.*
4. *Normal decoupling methods must be employed.*
5. *The microcontroller IOH for the port in use must not be exceeded.*

Figure F shows a typical power on to start bit sequence. Notice that the device is available to receive a clock at 5 μ s after Vcc has become stable.

FIGURE F



Many applications, especially remote or handheld data acquisition applications, where power consumption is at a premium or battery life is critical can use the POWER PORT technique with the PIC16/17 microcontrollers and possibly other microcontrollers. Remote metering applications where the controller must wake up and report previously stored data or periodically sample inputs, such as gas, electrical, or water monitoring systems are good examples where POWER PORT would be beneficial. Underground monitoring equipment for fuel storage and environmental monitoring systems are also suitable applications.

*AUTHORS: Richard J. Fisher
Memory Products Division
Bruce Negley
Memory Products Division*

Logic Powered Serial EEPROMs

LISTING A

```
LIST P=16C54
;
;
;   Sample test program to power up serial EEPROM
;   using PIC16/17 port A, then write one byte and read same byte, then repeat forever.
;
;*****
port_a equ 5h      ; port 5 used for device
                ; address select
port_b equ 6h      ; port 6 used for data and
                ; clock lines
eeprom equ 0ah     ; bitbuffer
addr equ 0ch       ; address register
datai equ 0dh      ; stored data input reg.
datao equ 0eh      ; stored data output reg.
slave equ 0fh      ; device address
                ; (1010xxx0)
txbuf equ 10h     ; txbuffer
count equ 11h     ; bit counter
bcount equ 12h    ; byte counter
rxbuf equ 13h     ; receive buffer
loops equ 15h     ; delay loop counter
loops2 equ 16h    ; delay loop counter 2
;
;   Bit Assignments
;
d equ 7           ; eeprom input
do equ 6          ; eeprom output
sdata equ 7       ; data line (port_b)
sclk equ 6        ; clock line (port_b)
vcc equ 3         ; vcc for dut (port_a)
;
org 01ffh
begin goto PWRUP
      org 000h
      goto PWRUP
;
;*****
;   DELAY ROUTINE
;   this routine takes the value in loops and loops that many times. Every
;   increase in 'loops' yields approx 1 more millisecond.
;   i.e., if 'loops' is 10 then the wait period is approx 10 milliseconds.
;
;-----
;-----
WAIT
;
top2 movlw .110
     movwf loops2
top  nop      ; sit and wait
     nop
     nop
     nop
     nop
     nop
     decfsz loops2 ; inner loop done?
     goto top     ; no, go again
     decfsz loops ; outer loop done?
     goto top2    ; no, go again
     retlw 0      ; yes, return from sub
;
```

Logic Powered Serial EEPROMs

```
,*****
;      Start Bit Subroutine
;      this routine generates a start bit
;-----
;
BSTART
    movlw    b'00111111'
    tris     port_b           ; port b for output
    bsf     port_b,sdata     ; set clock high
    nop
    nop
    bsf     port_b,sclk      ; set clock high
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    bcf     port_b,sdata     ; data line goes low during high clock for start bit
    nop
    nop
    nop
    nop
    bcf     port_b,sclk      ; start clock train
    nop
    nop
    nop
    retlw   0
;
;      End of Subroutine
,*****
;
;      Stop Bit Subroutine
;      this routine generates a stop bit
;-----
;
BSTOP
    movlw   b'00111111'      ;
    tris    port_b           ; set data/clock lines as outputs
    bcf     port_b,sdata     ; make sure data line is low
    nop
    nop
    nop
    nop
    nop
    bsf     port_b,sclk      ; set clock high
    nop
    nop
    nop
    nop
    nop
    bsf     port_b,sdata     ; data goes high while clock high
                                ; for stop bit
    nop
    nop
    nop
    nop
    bcf     port_b,sclk      ; set clock low again
    nop
    nop
    nop
    retlw   0
```

Logic Powered Serial EEPROMs

```
;
;   End of Subroutine
;*****
;   Serial data send 1 bit from PIC16/17 to dut
;-----
BITOUT
    movlw    b'00111111'    ; set data, clock as outputs
    tris     port_b
    btfss   eeprom, do
    goto    BIT0
    bsf     port_b, sdata    ; output bit 0
    goto    CLK1            ; data line clocked low by device
;
BIT0
    bcf     port_b, sdata    ; output bit 0
CLK1
    nop
    nop
    bsf     port_b, sclk     ; set clock line high
BIT2
    nop
    nop
    nop
    nop
    bcf     port_b, sclk     ; return clock line low
    retlw   0
;
;   End of Subroutine
;*****
;   Bit in routine
;   this routine gets a bit of data from the part
;   into the 'eeprom' register, bit 'di'
;-----
BITIN
    movlw    b'10111111'    ; make sdata an input line
    tris     port_b
    bcf     eeprom, di       ; assume input bit low
    bsf     port_b, sclk     ; set clock line high
    nop
    nop                    ; just sit here a sec
    nop
    nop
    nop
    nop
    nop
    nop
    btfsc   port_b, sdata    ; read data line
    bsf     eeprom, di       ; set input bit if needed
    bcf     port_b, sclk     ; set clock line low
    retlw   0                ; hit the road
;
;*****
;
;   Transmit Data Subroutine
;-----
TX
    movlw    .8
    movwf   count           ; set the #bits to 8
;
TXLP
    bcf     eeprom, do
    btfsc   txbuf, 7
    bsf     eeprom, do      ; otherwise data bit =1
    call    BITOUT         ; serial data out
    rlf     txbuf           ; rotate txbuf left
    decfsz  count          ; 8 bits done?
```



Logic Powered Serial EEPROMs

```

        goto    TXLP    ; no - go again
        call    BITIN   ; read ack bit
        ;
        retlw   0
;
; End of Subroutine
;*****
;
; Receive data Routine
; this routine gets a byte of data from the part into 'rxbuf'
;
;-----
RX
        movlw   .8           ; set # bits to 8
        movwf  count
        clrf   rxbuf        ; clear receive buffer
RXLP    rlf     rxbuf        ; rotate buffer left 1 bit
        bcf   rxbuf,0       ; assume bit is zero
        call  BITIN        ; read a bit
        btfsc eeeprom,di   ; input bit high?
        bsf   rxbuf,0       ; yes, set buffer bit high
        decfsz count       ; 8 bits done?
        goto  RXLP        ; no, do another
        bcf   eeeprom,do   ; set ack bit = 0
        call  BITOUT       ; to finish transmission
        retlw  0
;
;*****
;
; Power up routine
; this routine blinks the lights
;
;-----
PWRUP
        movlw  b'00000001'
        tris  port_a        ; set RA0 as input, rest output
        bsf   port_a,vcc    ; turn on power to dut
        nop
        nop
        nop
        nop
        nop
;
;*****
;
; Byte Write Routine
; this writes the data in "55h" to the first byte
; in the serial EEPROM.
;
;-----
;
;
WRBYTE
        ;
        movlw  b'10100000'  ; set slave address and write mode
        movwf  slave
        movlw  b'01010101'  ; set data to 55h
        movwf  data0
        ;
        clrf  addr          ; set address to 00h
        ;
        call  BSTART        ; generate start bit
        movf  slave,w       ; get slave address
        movwf txbuf         ; into transmit buffer
        call  TX            ; and send it
        movf  addr,w       ; get word address
        movwf txbuf         ; into transmit buffer
        call  TX            ; and send it
        movf  data0,w      ; move data
        movwf txbuf         ; to transmit buffer
        call  TX            ; and transmit it
        call  BSTOP        ; generate stop bit
        ;
        movlw  .10

```


Logic Powered Serial EEPROMs

NOTES:

SECTION 6

SERIAL EEPROMS

TUTORIALS AND APPLICATION NOTES

Basic Serial EEPROM Operation - AN536	6- 1
Everything a System Engineer Needs to Know About	
Serial EEPROM Endurance - AN537	6- 15
Using the Microchip Endurance Predictive Software - AN562	6- 23
Interfacing 24LCXX Serial EEPROMs to the PIC16C54 - AN567	6- 27
Using the 24C65 and 24C32 with Stand-alone PIC16/17 Code - AN558	6- 51
24C01A Compatibility Issues and Its Mobility for Memory Upgrade - AN517	6- 91
Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559	6- 93
Using the 93LC56 and 93LC66 - AN560	6- 97
ER59256/93C06 and NMC9306/NMC93C06 Compatibility Issues - AN516	6- 115
1.8 Volt Technology - Benefits	6- 117
Serial EEPROM Solutions vs. Parallel Solutions	6- 119

Basic Serial EEPROM Operation

BASIC SERIAL EEPROM OPERATION

Looking for the optimum nonvolatile memory product for your system that requires a small footprint, byte level flexibility, low power, and is highly cost effective? Serial EEPROM technology is one of the nonvolatile memory technologies that has emerged as a leading embedded control solution. Unfortunately, most system designers are not aware of the Serial EEPROM benefits. Also, the supporting documentation in data books is most often not adequate due to incomplete or ambiguous information. As a result, the system designer often selects a nonvolatile solution that does not meet their requirements or the designer must face a more complicated design-in with a Serial EEPROM.

This article is to address two issues that exist today for designers considering Serial EEPROM products:

First, to provide awareness of the application benefits.

Second, to provide a primer on the operating principles and instructions. These items are often buried in data book text or not adequately addressed. Also included are common default conditions to significantly reduce the system designers learning curve.

TABLE OF CONTENTS

- Serial EEPROM Applications
- Overview of the Primary Protocol Benefits
- 3 Wire Bus Operation Primer
- 2 Wire Bus Operation Primer
- Microchip 2 Wire Default Conditions
- Timing Diagram Attachments

SERIAL EEPROM APPLICATIONS

Serial EEPROMS are ideal nonvolatile cost effective memory solutions in applications that require:

Small footprint and board space as in cellular phone applications;

BYTE level ERASE, WRITE, and READ of data as in a TV tuner;

Low voltage and current for hand held battery applications as in a keyless entry transmitter;

Multiple nonvolatile functions in the same application such as a VCR;

Low availability of microcontroller I/O lines.

The common applications for Serial EEPROMS are shown below:

Market	Common Applications
Consumer	TV tuners, VCRs, CD players, cameras, radios, and remote controls
Automotive	Airbags, anti-lock brakes, odometers, radios, and keyless entry
Office Automation	Printers, copiers, PC's, and portable PC's
Telecom	Cellular, cordless and full feature phones, Faxes, modems, pagers, and satellite receivers
Industrial	Bar code readers, point of sale terminals, smart cards, lock boxes, garage door openers, and test measurement equipment.

The typical functions that Serial EEPROMS are utilized for are:

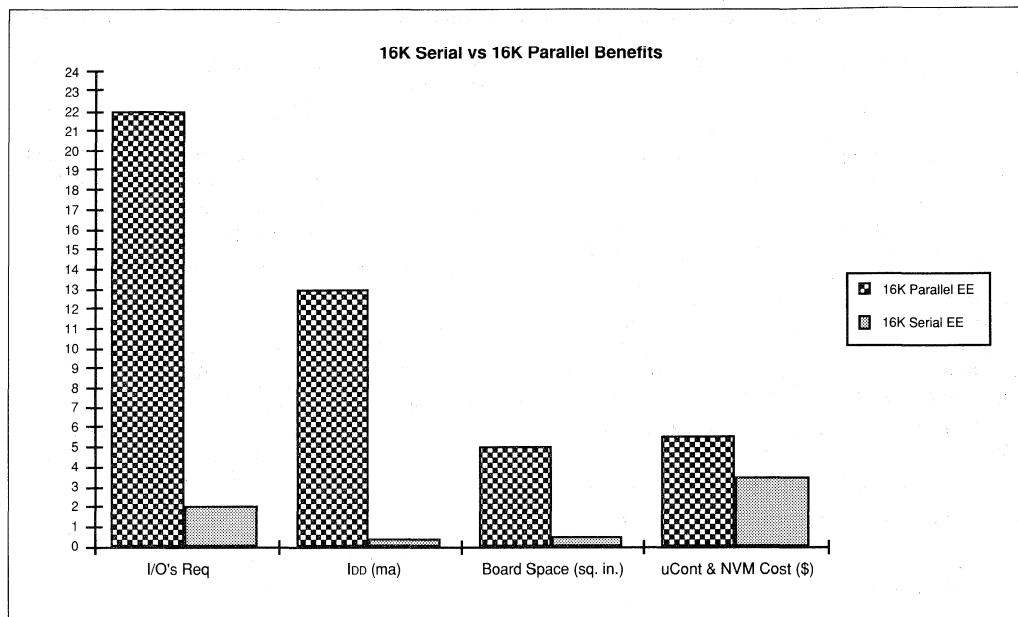
- Memory storage of channel selectors or analog controls (volume, tone, etc.) in consumer electronics products.
- Power down storage and retrieval of events such as fault detection or error diagnostics in automotive products.
- Electronic real time event or maintenance logs such as page counting in office automation products. Also, configuration or dip switch storage in office automation products.
- Last number redial storage and speed dial number storage in telecom products.
- User in-circuit reprogrammable look up tables such as bar code readers, point of sale terminals, environmental controls and other industrial products.

Other application examples include:

- Data storage from a learn function as in a remote control transmitter.
- ID number storage for security or remote access for electronic keys and entry databases.
- Reprogrammable calibration data for test equipment or analog interface products.

Basic Serial EEPROM Operation

As a result of density and architectural evolution, Serial EEPROMs offer significant benefits in some applications that previously could only utilize Parallel EEPROM products. The diagram below illustrates the footprint and board space differences.



The Serial EEPROM requires only 10% of the board space that a Parallel EEPROM requires. Also, the Serial EEPROM requires fewer I/O lines from the microcontroller which significantly reduces the overall system cost and board space.

A very fast READ speed is the only significant limitation of a Serial EEPROM for a decision between a Serial and a Parallel EEPROM. It is very interesting to note that the Serial EEPROM READ speed is restricted more by the protocol than the process technology. The 2-wire I²C (Inter-Integrated Circuit) products must add large internal delays to slow down the part to meet the 100 kHz protocol requirements, which will be reviewed later. Characterization of 3-wire bus Serial EEPROMs have indicated clock frequencies in excess of 6 MHz.

OVERVIEW OF THE PRIMARY PROTOCOL BENEFITS

After a designer decides to use a Serial EEPROM solution, the next step is to select one of the two primary Serial EEPROM protocols. Unfortunately, most system designers select the type of Serial EEPROM (2 or 3-wire) that they are most familiar with, regardless of the benefits associated with each type.

Basic Serial EEPROM Operation

The benefits of each protocol are shown below:

3 Wire Bus Serial EEPROMS	2 Wire Bus Serial EEPROMS
Single VDD supply of <2V to 5.5V	Single VDD supply of <2V to 5.5V
Very low current consumption	Very low current consumption
Reduced overall component cost	Reduced overall component cost
Four pins (other than Vcc & GND) are required for operation	Two pins (other than Vcc & GND) are required for operation
X16 bit and X8 bit data widths	X8 data bit width
Software WRITE Protection	Hardware WRITE Protection
Edge triggered clocks and signals	Level triggered clocks and signals and input glitch filters for high noise immunity
2Mhz+ operation	I ² C standard 100 KHz and 400 KHz protocols with a 1 MHz option
Ready/Busy data polling	Page WRITE capability to 16 bytes
Security options available	Software and hardware compatible from 2k to 16K densities
Less complex protocol	

A 2 wire product is utilized in applications that require an I²C bus, noise immunity, limited microcontroller I/O pin availability, or a WRITE buffer for multiple bytes to be stored with one instruction. A 3 wire product is utilized in applications that have limited protocol requirements, an SPI protocol, higher clock frequency requirements, or a x16 data width applications.

The next two sections describe the basic operation and Microchip's default conditions for the 3-wire and 2-wire Serial EEPROMs to allow the system designer to utilize the benefits of Serial EEPROMs.

3 WIRE BUS OPERATION PRIMER

Many Serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are unfortunately either vaguely embedded in the data sheet text or not included. Serial EEPROMs are not conventional memories due to the Serial communication protocols involved. This section is a PRIMER for the data sheet to familiarize the system designer with the basic principles of the 3 wire bus operation.

Basic Principles

Common device nomenclature is 93XXXX.

- The 93XX06 is a 256 bit product.
- The 93XX46 is a 1k bit product.
- The 93XX56 is a 2k bit product.
- The 93XX66 is a 4k bit product.

Four pins are required:

- | | |
|------------------|---------------|
| CS (Chip Select) | DI (data in) |
| CLK (Clock) | DO (data out) |

All 93XXXX parts are hardware compatible for these four pins. However, there may be compatibility issues for the other pins.

SOFTWARE compatibility is a key issue since there may be subtle differences in each manufacturers protocol. These subtle differences referred to as default conditions are addressed later in the attached 3 Wire Timing Diagram examples for Microchip's products. Software compatibility for density migration should also be reviewed by the designer on a case by case basis. There is no software industry compatibility for a 256 bit part to a 4k part.

Data is available in x8 or x16 organizations. This selection is determined either by the ORG pin or by purchasing a standard x16 organization.

Units will power-up in a EWDS (ERASE/WRITE Disable State). All ERASE and WRITE functions are disabled until the EWEN (ERASE/WRITE Enable) instruction is performed. This is to prevent accidental data corruption.

An Auto-ERASE (logical "1") cycle is performed during each WRITE Cycle.

The 7 instructions are shown in the attached instruction set table. These instructions are for Microchip's 93LCXX family products.

After an instruction is loaded, the CLK and DI pins are in a DON'T CARE state until the next START bit.

Basic Serial EEPROM Operation

The following is required for each instruction set (all input bits are triggered by the positive clock edges):

Start Bit	The first Data-in high signal clocked in after CS is high.
Opcode	Two Bits to identify the instruction
Address	Refer to the Instruction Set table for the number of bits required.
Data	Separate data-in and data-out pins. However, these two pins may be tied together for true 3-wire operation. Please refer to the attached 3-wire Bus READ timing diagram example.

READ, WRITE, and ERASE

The attached 93LC66 timing diagrams illustrate the key concepts and timing parameters for each of these operations. Please refer to the instruction set tables and the AC parameters in the databook for supplemental information.

ERASE ALL (ERAL)

An ERASE ALL (ERAL) operation is identified by a "00" opcode. The ERAL instruction requires the next two bits to be clocked in as "10" in the address block of the instruction set. All bits in the array will be set to a logic "1" state by one command in typically less than 10ms.

WRITE ALL (WRAL)

A WRITE ALL (WRAL) operation is also identified by a "00" opcode. The WRAL requires the next two bits to be clocked in as "01" in the address block of the instruction set. The data-in block will contain the data for a SINGLE BYTE which is to be repeated throughout the entire array. For example, if a 4F5A is loaded in the 16 data-in bits of the instruction set, a 4F5A will be written into every word in the array.

EWEN and EWDS

As stated before, all units will power up in to an ERASE/ WRITE DISABLE (EWDS) state to prevent data corruption. All future ERASE/WRITE operations must execute an ERASE/WRITE ENABLE (EWEN) op-code until the next power down is detected or until other EWDS opcodes are executed. Please refer to the instruction set table.

Basic Serial EEPROM Operation

INSTRUCTION SET FOR 93LC46: ORG = 1 (x 16 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A5 A4 A3 A2 A1 A0	—	D15 - D0	25
EWEN	1	00	1 1 X X X X	—	High-Z	9
ERASE	1	11	A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	9
ERAL	1	00	1 0 X X X X	—	(RDY/BSY)	9
WRITE	1	01	A5 A4 A3 A2 A1 A0	D15 - D0	(RDY/BSY)	25
WRAL	1	00	0 1 X X X X	D15 - D0	(RDY/BSY)	25
EWDS	1	00	0 0 X X X X	—	High-Z	9

INSTRUCTION SET FOR 93LC46: ORG = 0 (x 8 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A6 A5 A4 A3 A2 A1 A0	—	D7 - D0	18
EWEN	1	00	1 1 X X X X	—	High-Z	10
ERASE	1	11	A6 A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	10
ERAL	1	00	1 0 X X X X	—	(RDY/BSY)	10
WRITE	1	01	A6 A5 A4 A3 A2 A1 A0	D7 - D0	(RDY/BSY)	18
WRAL	1	00	0 1 X X X X	D7 - D0	(RDY/BSY)	18
EWDS	1	00	0 0 X X X X	—	High-Z	10

INSTRUCTION SET FOR 93LC56: ORG = 1 (x 16 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	X A6 A5 A4 A3 A2 A1 A0	—	D15 - D0	27
EWEN	1	00	1 1 X X X X X X	—	High-Z	11
ERASE	1	11	X A6 A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	11
ERAL	1	00	1 0 X X X X X X	—	(RDY/BSY)	11
WRITE	1	01	X A6 A5 A4 A3 A2 A1 A0	D15 - D0	(RDY/BSY)	27
WRAL	1	00	0 1 X X X X X X	D15 - D0	(RDY/BSY)	27
EWDS	1	00	0 0 X X X X X X	—	High-Z	11

INSTRUCTION SET FOR 93LC56: ORG = 0 (x 8 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	X A7 A6 A5 A4 A3 A2 A1 A0	—	D7 - D0	20
EWEN	1	00	1 1 X X X X X X	—	High-Z	12
ERASE	1	11	X A7 A6 A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	12
ERAL	1	00	1 0 X X X X X X	—	(RDY/BSY)	12
WRITE	1	01	X A7 A6 A5 A4 A3 A2 A1 A0	D7 - D0	(RDY/BSY)	20
WRAL	1	00	0 1 X X X X X X	D7 - D0	(RDY/BSY)	20
EWDS	1	00	0 0 X X X X X X	—	High-Z	12

INSTRUCTION SET FOR 93LC66: ORG = 1 (x 16 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A7 - A0	—	D15 - D0	27
EWEN	1	00	11XXXXXX	—	High-Z	11
ERASE	1	11	A7 - A0	—	(RDY/BSY)	11
ERAL	1	00	10XXXXXX	—	(RDY/BSY)	11
WRITE	1	01	A7 - A0	D15 - D0	(RDY/BSY)	27
WRAL	1	00	01XXXXXX	D15 - D0	(RDY/BSY)	27
EWDS	1	00	00XXXXXX	—	High-Z	11

INSTRUCTION SET FOR 93LC66: ORG = 0 (x 8 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A8 - A0	—	D7 - D0	20
EWEN	1	00	11XXXXXX	—	High-Z	12
ERASE	1	11	A8 - A0	—	(RDY/BSY)	12
ERAL	1	00	10XXXXXX	—	(RDY/BSY)	12
WRITE	1	01	A8 - A0	D7 - D0	(RDY/BSY)	20
WRAL	1	00	01XXXXXX	D7 - D0	(RDY/BSY)	20
EWDS	1	00	00XXXXXX	—	High-Z	12

Basic Serial EEPROM Operation

2 WIRE BUS OPERATION PRIMER

As indicated in the 3 wire bus section, many Serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are unfortunately either vaguely embedded in the data sheet text or not included. This section is a PRIMER for the data sheet to familiarize the system designer with the basic 2 wire Serial EEPROM operation principles.

Basic Principles

The common device nomenclature is 24XXXX and 85XXXX.

Only the SCL and SDA pins are essential for bus operation. The other pins are supplementary:

SCL (Serial clock)

SDA (Serial Data)

WP (Active High WRITE Protection)

A0,A1,and A2 (Chip or block select)

SDA's open-drain requires a pull-up resistor to V_{DD}.

The data is organized as X8.

Signals are level triggered, not edge triggered. Also, there are filters on the inputs that will filter noise glitches <100ns wide.

An Auto-ERASE (logical "1") cycle is performed during each WRITE cycle.

The I²C protocol utilizes master / slave bi-directional communication. A device that sends data onto the bus is defined as the transmitter, and a device that is receiving data is the receiver. Both the master and the slave can operate as the transmitter or receiver. The bus must be controlled by a master device (most often a microcontroller), which generates the serial clock (SCL), controls the bus direction, and generates the START and the STOP conditions.

The Serial EEPROM is the slave. The Serial EEPROM will be the bus transmitter during READ operations and when the Serial EEPROM must acknowledge data transmitted by the master.

START and STOP bits control the bus activity. Operations must begin with a START bit and end with a STOP bit.

A START bit is when SDA transitions LOW while SCL is HIGH while observing the START set-up and hold time specifications.

A STOP bit is when SDA transitions HIGH while SCL is HIGH while observing the STOP set-up and hold time specifications.

Data is recognized as valid while SCL is high. The data on SDA must observe data in set-up and hold specifications before and after SCL is pulsed. There is only one bit of data for each SCL pulse.

Control Byte Requirements

After a START bit, each command begins with an 8 bit control byte sent by the master. This control byte has the following three primary functions before the data and/or word address information is loaded for all commands:

Identify the Serial EEPROM as the slave addressed on the bus.

To select the specific Serial EEPROM or the internal memory block on the bus. There may be up to 8 Serial EEPROMs on the bus)

Select the READ or WRITE function for the next command transmitted by the master.

The diagram of a control byte (not including the START bit) is shown below:

1	0	1	0	A2	A1	A0	X
I ² C Slave Address				Chip or Block Select			Read or Write bit

Control Bits 1-4 are the Slave Address Bits (Must be 1010 for Memory)

Since there is not a chip select pin, the part is selected by a four bit code in the control byte to identify the type of product. The four bit code was established by Philips for the I²C protocol. A 1010 code identifies the slave device as a Serial EEPROM. The Serial EEPROM will remain in stand-by until the 1010 code is transmitted on the bus. Other non Serial EEPROM slave devices will not respond to the 1010 code on the bus.

Control Bits 5-7 are the 1 of 8 Chip or Block Address Select Bits

The next three control bits are utilized for the chip selection or internal block selection. The standard I²C protocol was developed to allow up to 16k bits of memory to be selected. This could be accomplished by accessing a combination of devices or blocks within a device, as shown in the table on the following page:

Basic Serial EEPROM Operation

Device	K bits Density	Internal Blocks	A0	A1	A2	Bus Access Devices
24LC01B, 24C01,85C72	1	1	H or L	H or L	H or L	Up to 8 devices
24LC02B, 24C02,85C82	2	1	H or L	H or L	H or L	Up to 8 devices
24LC04B, 24C04,85C92	4	2	X	H or L	H or L	Up to 4 devices
24LC08B	8	4	X	X	H or L	Up to 2 devices
24LC16B	16	8	X	X	X	Only 1 device

X= NOT USED. This pin must be tied to Vss or Vdd. It is not recommended to FLOAT these pins since there may be test modes accessed to these pins via a high voltage signal.

These three bits for this select must match the hardware conditions (IF ANY ARE USED) of the external A0, A1, and A2 pins or the internal block selects.

With this selection scheme, devices from 2k to 16k are software compatible. For example, four 2k devices or one 8k device could be connected to the bus with the same software.

The A0, A1, and A2 signals are the same for the 1k and 2k products. The A7 bit for the 1k product is a DON'T CARE.

The A0, A1, and A2 pins are not commonly used today in the industry with the advent of the density evolution up to the I²C protocol limit of 16K bits.

Control Bit 8 Operation Code

If this bit is a "1" then the operation will be a READ
If this bit is a "0" then the operation will be a WRITE

After the control byte acknowledge bit is generated by the Serial EEPROM, the master will send the appropriate word address and data information.

Acknowledge Requirements

The Serial EEPROM must generate an acknowledge bit after receiving each byte segment in a command. The Serial EEPROM will generate the acknowledge bit automatically after the master has transmitted all of the data for the segment.

To acknowledge the master, the Serial EEPROM must pull the SDA line LOW during the entire HIGH period of the next clock generated by the master. During the READ operations, the master must acknowledge each data byte or the Serial EEPROM will abort the READ operation and return to a stand-by mode waiting for the next START bit.

The attached 24LC16 timing diagrams illustrate the READ and WRITE operations.

MICROCHIP 2 WIRE DEFAULT CONDITIONS

As stated before, data sheets do not provide adequate information on basic operation. This lack of information forces each reader of the databook to make interpretations of the operating conditions. These readers have included other semiconductor circuit designers, which unfortunately leads to subtle compatibility problems. The part is designed to operate to the default of the circuit designers interpretation. This next section details Microchip's default conditions to help the system engineer to minimize "Trial and Error" prototyping and to increase the awareness of these default conditions.

Also, to improve corporate wide compatibility, Microchip is standardizing their circuits on various product versions. Unless indicated otherwise, all references to default conditions are for the 24LCXX products, not the 24CXXXX products.

Power Up

READ, WRITE, and ERASE operations are valid 5 μ S after Vdd has ramped to the specified operating range.

PAGE WRITE by Product for Multiple BYTE WRITE Operation

The 24C01 and 24C02 have a 2 byte buffer.

The 24C04 has an 8 byte buffer.

The 24LC01 and 24LC02 have an 8 byte page.

The 24LC04, 24LC08, and 24LC16 have a 16 byte page.

The buffer will load bytes identically as the page loads bytes. The difference in the two modes is that the buffer will execute a WRITE of one byte per WRITE cycle in sequence. The page mode will execute all bytes loaded in one WRITE cycle in parallel.

Basic Serial EEPROM Operation

There are pages within blocks. For a 16 byte page product, the most significant 4 bits of the word address point to the page address and the least significant 4 bits point to the byte address within a page. For an 8 byte page product, the most significant 5 bits of the word address point to the page address and the least significant 3 bits point to the byte address within a page.

The number of bytes loaded in to the page is from one byte up to the page size. For example, three bytes can be loaded into the 16 byte page of the 24LC16. If during the loading of the fourth byte a STOP bit is received, the page will WRITE three bytes. The fourth byte will not be written since loading the fourth byte was not complete.

NOTE: New versions released in March 1993 will default to ABORTING the entire operation if a STOP bit is received in the middle of a byte while loading a page.

If more than 16 bytes are loaded in the page of a 16 byte page product, then the 17th byte will override the data loaded into the original first byte (the page data will wrap around WITHIN a page). Therefore, the system designer must take precautions to not WRITE over a page boundary during a multiple byte WRITE operation.

Bytes not changed in the page will NOT result in data corruption in the array. For example, If two bytes are loaded in to the 24LC16 page with the least significant word address bits of 0000 and then a STOP bit is transmitted. Bytes 1 and 2 in the array will have the data changed to the new page contents. Bytes 3 through 16 WILL NOT change.

The WRITE operation will not be executed until a STOP bit is transmitted.

At this point, the Serial EEPROM is free from the bus since the actual WRITE function is self-timed. Therefore, the microcontroller interfacing to the Serial EEPROM may perform other functions not associated to communication with the Serial EEPROM during the self-timed WRITE operations.

Once the part is in the auto-ERASE mode, it will complete the ERASE/WRITE operation unless power is removed. STOP and START bits will be ignored.

READ

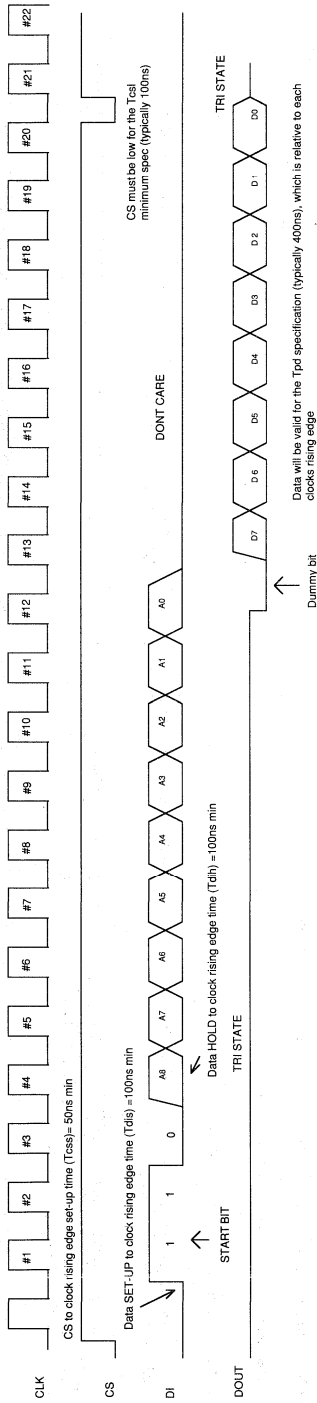
Once the Serial is in a RANDOM READ operation, it can be placed into the sequential READ operation. If the master issues an acknowledge bit instead of a STOP bit, the Serial will READ the next sequential 8 bits. The Serial will wait for the next bit command from the master. The sequential READ will continue as long as the master issues an acknowledge bit on the next clock cycle after the last bit is READ. The READ will continue from block to block and will wrap around if the last bit in the array is addressed. Again, this will continue until the master issues a STOP bit instead of an acknowledge bit.

While reading zeroes the master cannot pull SDA high to generate a STOP bit, since the Serial EEPROM SDA pin is outputting a low. To recover from a fault during a READ, repeat 9 clocks with data floating high. Therefore, the acknowledge bit will not occur and the part will reset and return to stand-by.

A START bit during an operation will cease the current operation and begin the next operation.

*AUTHOR: Steve Drehobl
Memory Products Division*

3 WIRE BUS EXAMPLE MICROCHIP 93LC66 READ CYCLE TIMING DIAGRAM



NOTE: THIS EXAMPLE IS FOR X8 OPERATION OF MICROCHIP'S 93LC66. INSTRUCTION LENGTHS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS

Data must conform to specified set-up and hold times (T_{ds} and T_{dh}) relative to the RISING clock edge. Each parameter is typically 100ns.

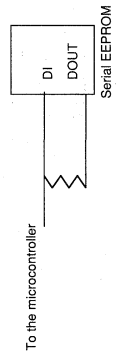
A Read operation is identified by the "1 0" op-code following the start bit.

Next, the address location bits are loaded.

Then the address contents are clocked out on the rising clock pulse edge. Data will become valid on the DOUT pin per the specified T_{pd} time (typically 400ns) relative to the rising edge of the clock on the DOUT pin. Note, the first bit output will be a "dummy bit" with a logical zero state. This event is triggered by the clock rising edge of the last address bit for a duration of one clock pulse.

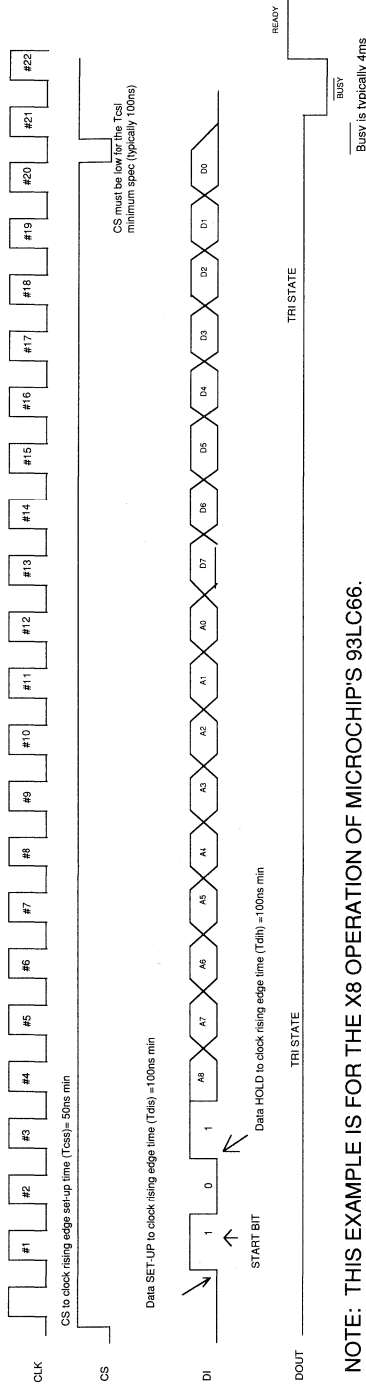
If the data from the current address is complete and the clock pulses continue, the data from the next address will be READ automatically as long as CS remains high. This is the SEQUENTIAL READ FUNCTION. READ operations will continue while clock pulses continue or until CS is brought low.

It is possible to tie the DOUT pin and the DI pin together to save on I/O requirements from the microcontroller. Caution must be exercised to avoid bus contention for an A0 high condition, because of the dummy bit. It is recommended that a resistor between the microcontroller port connected to the DI pin and DOUT pin be added for isolation. This example is shown below:



Basic Serial EEPROM Operation

3 WIRE BUS EXAMPLE MICROCHIP'S 93LC66 WRITE CYCLE TIMING DIAGRAM



NOTE: THIS EXAMPLE IS FOR THE X8 OPERATION OF MICROCHIP'S 93LC66. INSTRUCTION LENGTHS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS.

Data must conform to specific set-up and hold times (T_{dis} and T_{dh}) relative to the clock edge. Each parameter is typically 100ns.

A WRITE operation is identified by a the "0 1" op code following the start bit

Next, the address location bits are loaded on the DI pin.

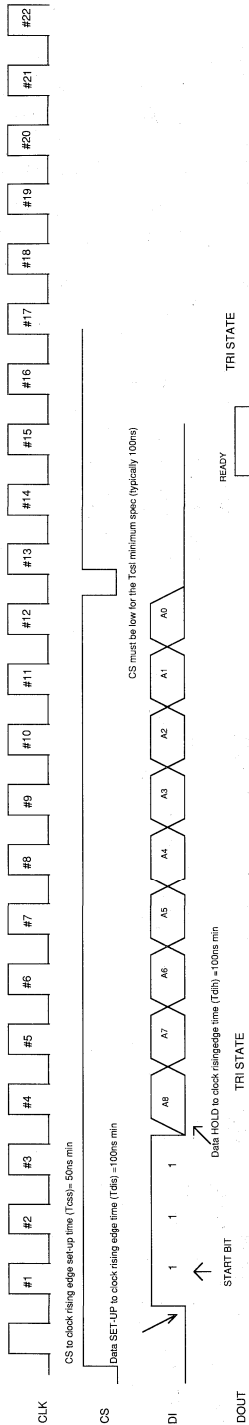
Then, the data bits to be written are loaded on the DI pin.

CS must be brought low after the last DI bit is loaded. When CS is brought low for the T_{cs1} period, a self timed WRITE is executed. If too many bits are loaded during ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

The DOUT pins only function during a WRITE, is to indicate the status of the write with the READY/BUSY function. While DOUT is low, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When DOUT is high, the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the T_{cs1} time is complete to initiate the READY/BUSY function. Microchip's 93LCXX products can be polled multiple times for the same cycle.

Up through clock pulse 20, data for the instruction is being LOADED. When the CS goes low, the instruction is being EXECUTED. If there are not enough bits loaded during ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the Serial EEPROM will return to stand-by.

3 WIRE BUS EXAMPLE MICROCHIP 93LC66 ERASE CYCLE TIMING DIAGRAM



NOTE: This example is for X8 operation of the Microchip's 93LC66. Instruction lengths may vary with array size and data widths.

Data must conform to specified set-up and hold times (T_{dis} and T_{din}) relative to the clock edge. Each parameter is typically 100ns.

AN ERASE operation is identified by a "11" two bit code that follows the start bit

Next, the address location bits are loaded on the DI pin.

THERE ARE NO DATA BITS TO LOAD. THE ADDRESS LOCATION LOADED WILL BE SET TO AN ERASE STATE OF "1".

CS must be brought low after the last DI bit is loaded. When CS is brought low for the T_{csi} period, a self timed ERASE is executed. If too many bits are loaded during the ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

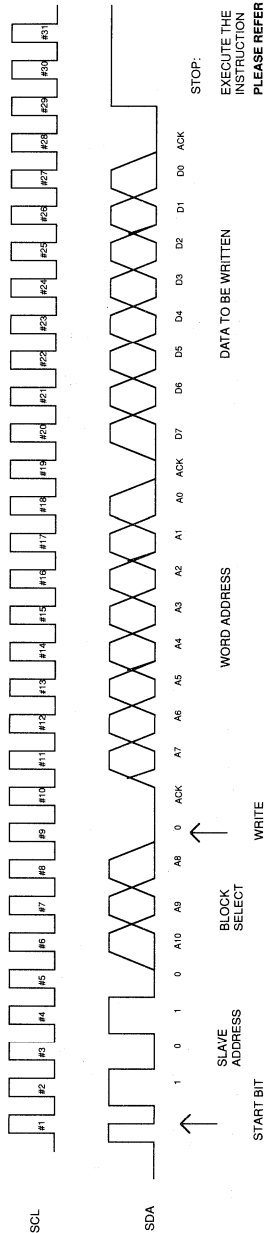
The DOUT pin's only function during an ERASE is to indicate the status of the write with the READY/BUSY function. While DOUT is low, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When Do is high, the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the T_{csi} time is complete to initiate the READY/BUSY function.

Up through clock pulse 12, the address for the instruction is being LOADED. When CS goes low, the instruction is being EXECUTED. If there are not enough bits loaded during the ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the serial EEPROM will return to stand-by.



Basic Serial EEPROM Operation

2 WIRE BUS EXAMPLE MICROCHIP 24LC16 BYTE WRITE CYCLE TIMING DIAGRAM



THE SEQUENCE OF EACH WRITE COMMAND AND THE MASTER/SERIAL DIRECTION OF THE COMMUNICATION IS SHOWN BELOW :

BYTE WRITE

- START BIT FROM THE MASTER
- CONTROL BYTE FROM THE MASTER
- ACKNOWLEDGE BIT FROM THE SERIAL
- WORD ADDRESS FROM THE MASTER
- ACKNOWLEDGE BIT FROM THE SERIAL
- DATA BYTE FROM THE MASTER
- ACKNOWLEDGE BIT FROM THE SERIAL
- DATA BYTE FROM THE MASTER
- ACKNOWLEDGE BIT FROM THE SERIAL
- DATA BYTE FROM THE MASTER
- ACKNOWLEDGE BIT FROM THE SERIAL
- STOP BIT FROM THE MASTER

NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE

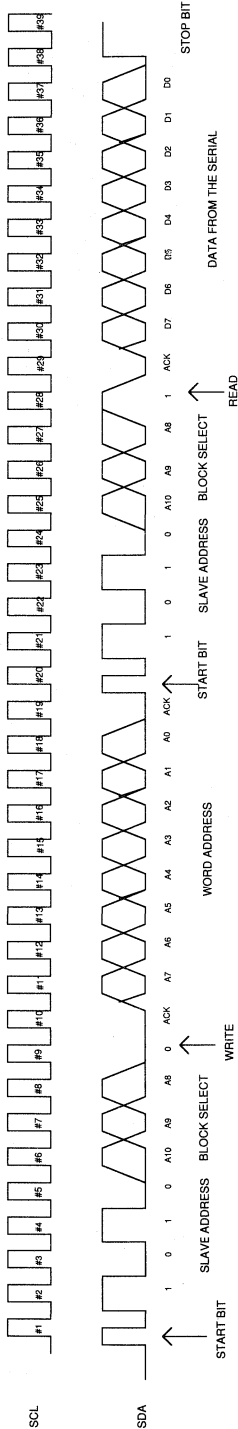
ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL DATA SET-UP TIME OF 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (THD:DAT) FOR THE FALLING CLOCK EDGE.

THE STOP BIT ON CLOCK PULSE #29 WILL INITIATE A SELF TIMED WRITE. THE SERIAL EEPROM CAN NOT EXECUTE ADDITIONAL INSTRUCTIONS UNTIL THE CYCLE IS COMPLETE.

TO EXECUTE A PAGE WRITE, CONTINUE TO LOAD 8 BITS OF DATA AT CYCLE 29 INSTEAD OF ISSUING A STOP BIT (FROM THE MASTER). REMEMBER, A CLOCK PULSE MUST BE ALLOCATED AFTER EACH SUBSEQUENT 8 BITS FOR THE SERIAL EEPROM TO ISSUE AN ACKNOWLEDGE SIGNAL (LOW) AFTER THE DESIRED NUMBER OF BYTES HAVE BEEN LOADED, UP TO THEIR PAGE SIZE, THE MASTER MUST ISSUE A STOP BIT TO EXECUTE THE INSTRUCTION.

2 WIRE BUS EXAMPLE

MICROCHIP 24LC16 READ CYCLE TIMING DIAGRAM



NOTE: THE FIRST 19 CLOCK PULSES OF THE WRITE COMMAND ARE IDENTICAL TO THE FIRST 19 CLOCK PULSES IN THE READ COMMAND. EVEN THE 9TH CLOCK PULSE IS A WRITE BIT '0' TO TRANSMIT TO THE SERIAL EEPROM THE DESIRED WORD ADDRESS. THE READ COMMAND HAS TWO START BITS. THIS IS FOR THE RANDOM READ COMMAND.

AS SHOWN ON THE PREVIOUS PAGES, IF A READ IS DESIRED FROM A CURRENT ADDRESS, THEN THE FIRST 19 CLOCK PULSES ARE NOT REQUIRED. THEREFORE, THE FIRST START BIT IS AT CLOCK PULSE #20. THIS IS ONLY FOR THE CURRENT ADDRESS READ COMMAND

ANOTHER USEFUL READ COMMAND IS THE SEQUENTIAL READ COMMAND. THE SEQUENTIAL READ COMMAND IS THE SAME AS THE RANDOM READ COMMAND; HOWEVER, THE MASTER MUST ISSUE AN ACKNOWLEDGE BIT INSTEAD OF THE STOP BIT AS SHOWN FOR CLOCK PULSE #38. THIS SIGNALS THE SERIAL TO READ THE DATA FROM THE NEXT SEQUENTIAL ADDRESS. THE MASTER MUST CONTINUE TO ACKNOWLEDGE EACH BYTE RECEIVED UNTIL THE MASTER ISSUES A STOP BIT.

NOTE: THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL. DATA SET-UP TIME OF 250NS (TSU; DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (TH0; DAT) FOR THE FALLING CLOCK EDGE.

READ (FROM THE CURRENT ADDRESS)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (FROM A RANDOM ADDRESS)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (Sequential Read at 3 bytes)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (Sequential Read at 3 bytes)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
STOP BIT FROM THE MASTER

Basic Serial EEPROM Operation

NOTES:



Everything a System Engineer Needs to Know About Serial EEPROM Endurance

The term "endurance" has become a confusing parameter for both users and manufacturers of EEPROM products. This is largely because many semiconductor vendors treat this important application-dependent reliability parameter as a vague specmanship topic. As a result, the system engineer often designs without proper reliability information or underutilizes the EEPROM as an effective solution.

Endurance (the number of times an EEPROM cell can be erased and rewritten without corrupting data) is a measure of the device's reliability, not its parametric performance. As such, endurance is not achieved by somehow making EEPROM devices more durable or robust to extend the life of the intrinsic erase/write cycle, but rather by reducing their defect-density failure rates. This has a direct impact on the design engineer characterizing EEPROM memory needs for an application and evaluating components from various manufacturers. The system design engineer needs to understand not only the relationship between the application, expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

This tutorial volume is intended to clarify some of the issues in the industry and provide a tool for the system design engineer, the system reliability engineer, and the component engineer to determine EEPROM reliability and understanding how to apply it to actual application requirements. It will examine four main areas:

- CMOS floating gate memory cell operation and characteristics
- Significant process and design interactions and endurance characterization variables
- Common mis-interpretations of endurance
- Determining some real world application reliability requirements

EEPROM MEMORY CELL OPERATION AND CHARACTERISTICS

In discussing endurance characteristics of EEPROMs, it's important to review how these components operate, and why and how they fail. Figure 1 illustrates a CMOS floating gate EEPROM cell, including voltage conditions for READ, ERASE, and WRITE operations. To erase or write, the row select transistor must have the relatively high potential of 20V. This voltage is internally generated on chip by a charge pump, with the only external voltage required being V_{dd}. The only difference between an ERASE and a WRITE is the direction of the applied field potential relative to the polysilicon floating gate.

When 20V is applied to the polysilicon memory cell gate and 0V is applied to the bit line drain (column), electrons tunnel from the substrate through the 90-angstrom Tunnel Dielectric (TD) oxide to the polysilicon floating gate until the polysilicon floating gate is saturated with charge. The cell is now at an ERASE state of "1". When 0V is applied to the polysilicon memory cell gate and 20V is applied to the bit line drain (column), electrons tunnel from the polysilicon floating gate through the TD oxide to the substrate. The cell then is at a WRITE state of "0". This sequence of the transfer of charge onto the floating gate (ERASE) and the electrical removal of that charge from the floating gate (WRITE) is one ERASE/ WRITE cycle, or "E/W cycle."

The field (applied voltage to an oxide thickness) across the tunneling path created by the 20V potential is extremely high in order to transfer the electrons. Over the cell's "application time," as measured by E/W cycles, the EEPROM cell begins to wear out due to the field stress. The EEPROM cell wears out as the number of cycles increase resulting in the voltage margin between the ERASE and WRITE states decreasing until finally there is not enough margin for the EEPROM sense amp to detect a difference in the two states during a READ. Failure is defined as when the sense amp can no longer reliably differentiate logic state changes.

Figure 2 (single cell EEPROM endurance characteristics) illustrates that the intrinsic wear out point for a normal cell with specified dimensions and electrical characteristics is very acceptable, in excess of 2 million E/W cycles. Failures at lower cycles are due mostly to very small defects or imperfections in the oxide or silicon-to-oxide interface.

Serial EEPROM Endurance

A key point to remember is that most failures occurring at less than 2 million E/W cycles are due to the number of defects per a given area (defect density dependent.) Thus high EEPROM endurance reliability is achieved by reducing the defect density failure rates, not by increasing the number of intrinsic cycles in the cell's operational design.

Error correction circuits are design techniques commonly used by EEPROM manufacturers to increase endurance by reducing the failure rate caused by single bit failures. These circuits are transparent to the user. One typical scheme is using 4 bits of error correction for every 8 "real" bits (one byte). In this scheme, one bit failure in the byte is correctable, while if two bits within the byte fail, the byte is not correctable.

Another error correction scheme is to use one "parity" bit for every "cell." Here both EEPROM cells must fail to result in a bit fail.

FIGURE 1 - CMOS FLOATING GATE EEPROM CELL

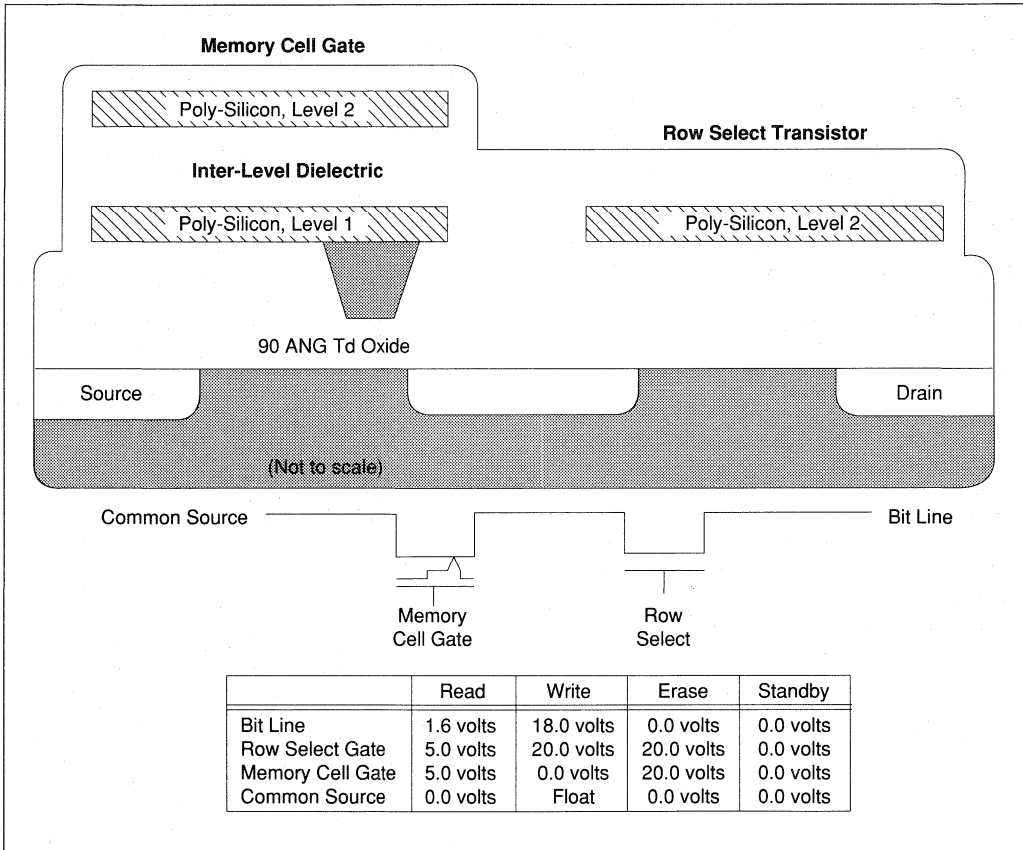
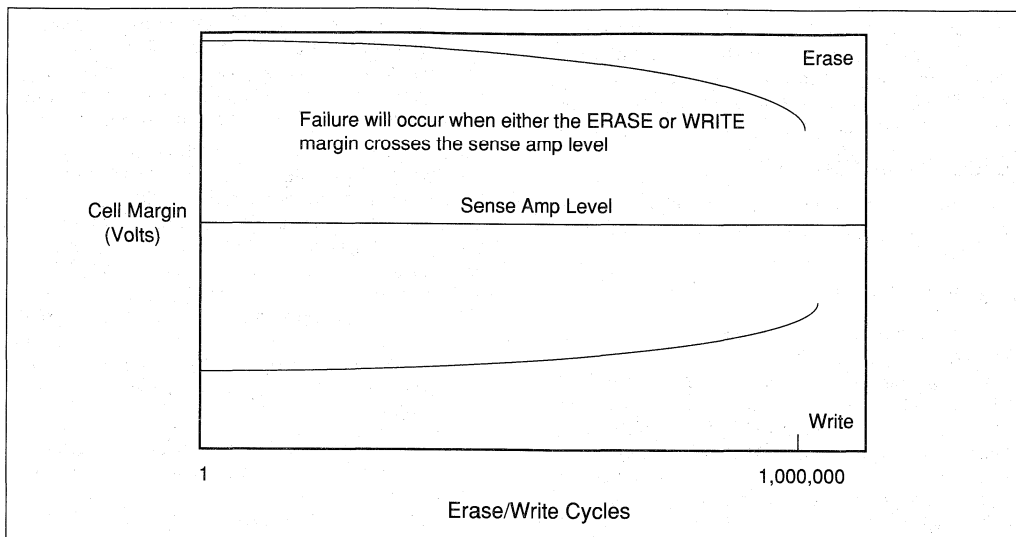


FIGURE 2 - SINGLE EEPROM CELL ENDURANCE CHARACTERISTICS



PROCESS AND DESIGN VARIABLES AFFECTING ENDURANCE

There are many subtle process and design variables that have a strong impact on endurance. These interacting variables will play very different roles depending on the different process technologies of various semiconductor manufacturers.

The primary interaction is the amount of TIME at the HIGH VOLTAGES that is ultimately applied to the cell. A finite amount of time at finite voltages are required to achieve "optimal" ERASE and WRITE thresholds. If the time is too short and the voltage is too low, the EEPROM will not program to the proper threshold. Also, if the programming ramp time is too fast and the voltage is too high, the EEPROM's endurance will be reduced. Unfortunately, there is most often a trade-off between fast reliable programming performance or high endurance reliability. Some of the significant process and design variables are shown below and their impact on programming performance and endurance performance.

PARAMETER	PROGRAMMING	ENDURANCE
Internal High Voltages	HIGHER = Faster Programming	LOWER = Increased Endurance
Internal High Voltage Ramp Rate	FASTER = Faster Programming	SLOWER = Increased Endurance
Programming Time	LONGER = Improved Voltage Margin on the Cell	SHORTER = Less Oxide Stress for Increased Reliability
TD Oxide thickness	THICKER = Slower Programming	THINNER = Reduced Endurance
Temperature	LOWER = Faster Programming	LOWER = Increased Endurance

Serial EEPROM Endurance

COMMON MISINTERPRETATIONS

In examining industry EEPROM literature on the topic of endurance, it's easy to misunderstand or misinterpret endurance concepts due to incomplete databook statements. The following are clarifications to some of the common misinterpretations:

Endurance and Read Cycles

READ operations are unlimited since they impose virtually no stress on the cell. Endurance data apply only to E/W cycles.

Erase/Write Ratings

E/W ratings are based on each byte in the application, not on the number of op codes or control byte commands utilized. For example, if a part is rated to 100K E/W cycles, then each individual byte can be erased and written 100K times. The part is NOT limited to only a total of 100K E/W op-codes or control bytes. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement.

Cycles/Day

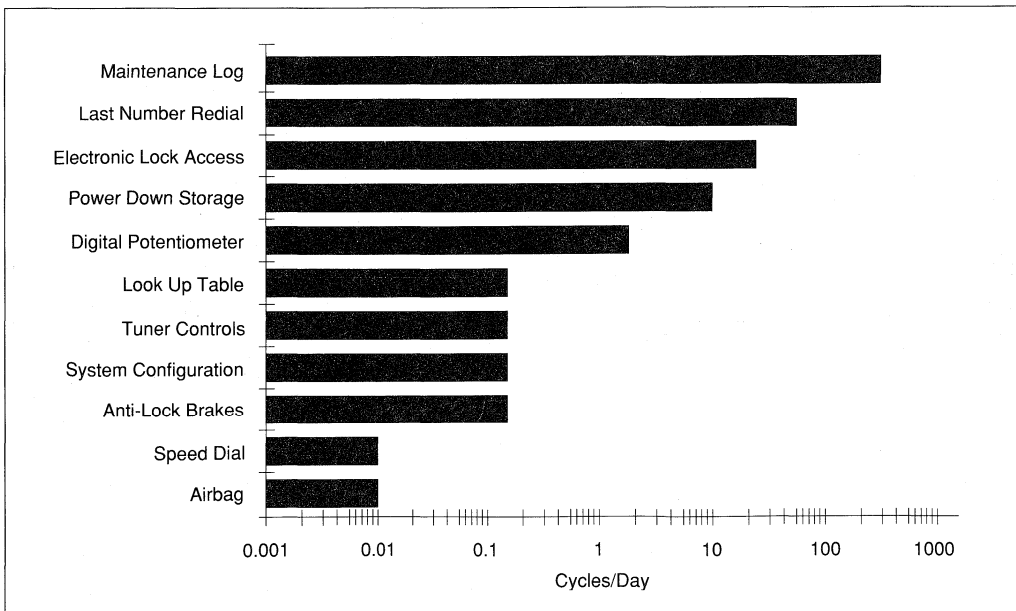
In many cases, a serial EEPROM is used for widely varying functions in an application. These functions have different E/W usage requirements (cycles/day), resulting in different endurance requirements and, usually, different reliability results for each function.

For example, assume a given end-product application will have a 10-year life. For each function within that application, an assumption must be made for the expected E/W cycles per day for a given segment of bytes. If a function has a segment of bytes cycled 1 time per day, then this segment of bytes will have 3,650 cycles in its lifetime (365 days per year for 10 years at 1 cycle per day and 7 days per week operation). Any given segment of bytes would have to cycle 274 times per day everyday to reach 1,000,000 E/W cycles in its 10-year application lifetime. Such a frequency is, of course, very rare in actual applications.

For reference purposes, Figure 3 indicates typical cycles per day for some common applications. Although many manufacturers routinely discuss very high numbers of E/W cycles, the amount of applications actually utilizing 1 million cycles is very small.

A further and very important incorrect assumption often made is that ALL bits in an application need the same number of cycles and endurance ratings. In most applications, however, functions that require a high

FIGURE 3 - TYPICAL SERIAL EEPROM E/W CYCLES/DAY BY APPLICATION



number of E/W cycles per day require only a small number of bits. Last number redial in a telephone, for example, consumes many E/W cycles per day, but utilizes only a few bytes for this function. By contrast, speed dial storage in that same telephone consumes only a fraction of E/W cycles per day, but requires a relatively large segment of bits to accommodate the many speed dial options. In such an application, the same serial EEPROM normally performs both functions at different address locations.

ENDURANCE DATA FROM THE CUSTOMER'S PERSPECTIVE

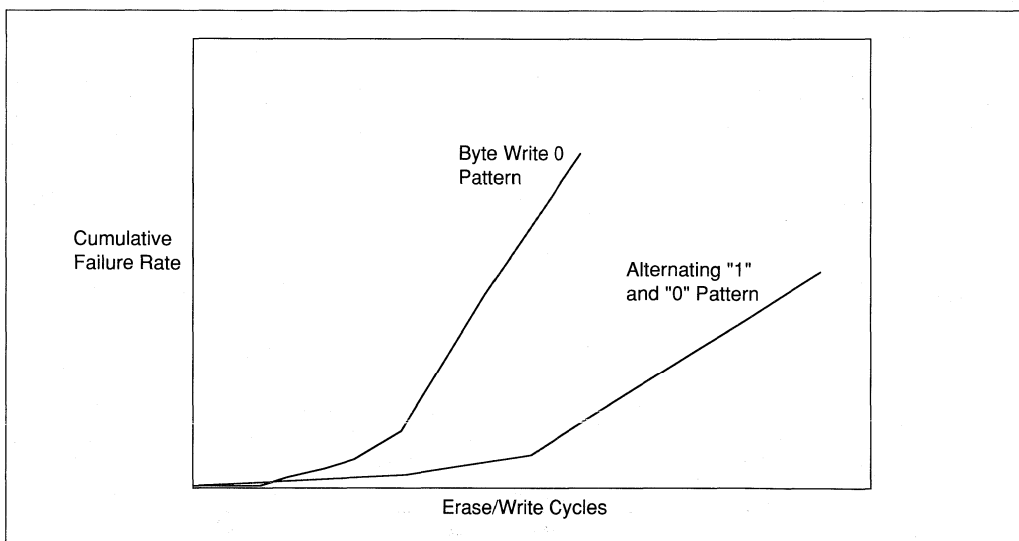
Unfortunately, an industry standard for an endurance test method has yet to be adopted. Since endurance data is not baselined, the process of evaluating endurance becomes that much more complicated for the system designer and reliability engineer.

It is not uncommon for customers to request endurance data from many semiconductor vendors. All vendors would be expected to comment that they experience a low failure rate through 100K E/W cycles. While this can be a true statement, it can also be a very incomplete statement. It is extremely doubtful that all vendors test their components to the same conditions. Yet the variables within endurance testing are extremely significant. Small differences in test protocol can have enormous differences. Pattern, cycling mode, temperature and array size, for example, are the most significant testing variables.

First, memory cell failure rates are defect density driven up to the intrinsic wear out point. Existing defects in a cell, while not causing failure initially, are stressed during every transfer of electrons through the TD oxide until they eventually cause cell failure. Worst case testing would be to erase and write each bit, which is what a write all "0"s pattern with an auto-erase of "1" routine will perform. Indeed, this write all "0"s test pattern will produce very different results than a checkerboard test pattern of alternating "1"s and "0"s within a byte, since cells are changed more often writing all "0"s than in an alternating "1" and "0" write pattern. The resultant failure rate differences are indicated on the pattern effect graph in Figure 4.

In actual use, however, a system will experience a random pattern much more like the alternating "1"s and "0"s pattern than the more stressful all "0"s pattern. The key point for system designers is to determine how accurate a test routine has been used to determine a particular manufacturer's endurance data, and make the appropriate judgement on that part's expected endurance in the application.

FIGURE 4 - PATTERN EFFECT ON ENDURANCE TESTING



Serial EEPROM Endurance

Second, the cycling mode graph in Figure 5 indicates that significantly different results can be achieved in endurance testing using a block cycle mode than using a byte cycle mode. The block mode is commonly used by manufacturers to "speed up" the endurance characterization process. However, endurance results usually will appear much better for the block mode than the byte mode, due to the high voltage variables discussed earlier. The reason is that the voltage ramp rate is significantly SLOWER, the high voltages are slightly lower, thus less stressful for block cycling since the capacitive load of the entire array is on the high voltage charge pump. The capacitive load is much lower with a single row or byte, which thus has a significantly faster ramp rate. Also, there is not a polysilicon to polysilicon stress for adjacent cells since all cells are at the same potential. Most often, these factors combine to yield lower failure rates for block cycling than for byte cycling. Again, the test conditions must match the system conditions.

Finally, increasing temperature also increases stress on the cell. Microchip's endurance characterization data indicates that increasing temperature adds an activation energy (E_a) of 0.12eV on the cell. From a 25°C to 85°C ambient the acceleration factor is approximately 2.1. Therefore, the higher the temperature, the higher the stress. These results will vary significantly with each EEPROM manufacturer.

RECOMMENDED EEPROM ENDURANCE TESTING

Microchip believes that EEPROM components should be endurance tested to reflect system conditions. Therefore, units are cycled to an alternating ONE and ZERO pattern (checkerboard), then to an alternating ZERO and ONE pattern (inverse checkerboard).

Again, since endurance characterization data indicates that a random single bit fail is the primary first order failure mode, endurance is defect density (def/cm² or segment of bit size) dependent. Therefore an expected failure rate range by density can be established.

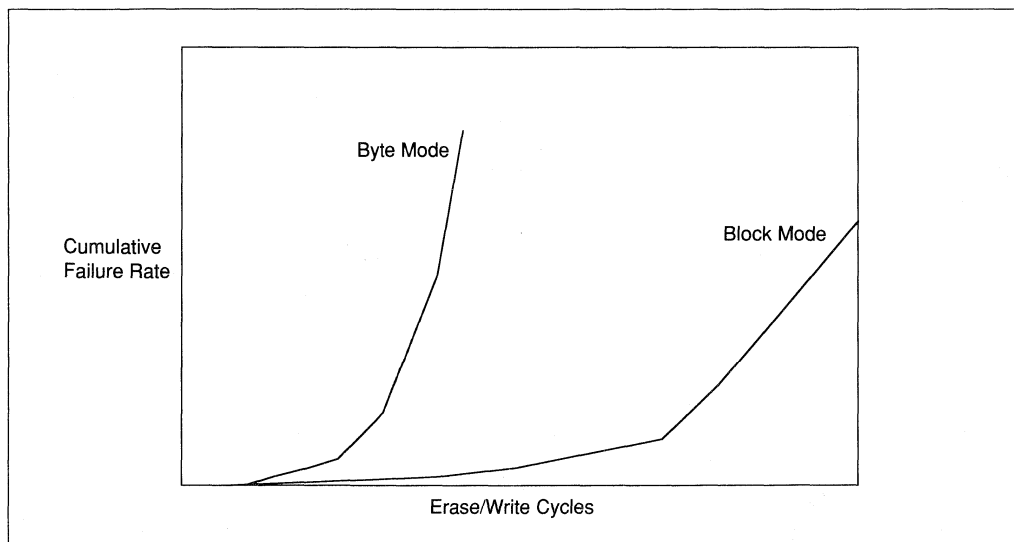
DETERMINING THE RELIABILITY CALCULATIONS

There are three primary components for the system design engineers to use in determining the endurance reliability required for a defect density limited application. These three components are:

- Erase/write cycles/day estimated for the the function.
- The number of bits in the function (or segment size).
- Case operating temperature of the Serial EEPROM.

Let's look at three typical examples utilizing the above information to predict a cumulative failure rate at different points in a system lifetime. Please note that Industry endurance perceptions have improved from a very high (>2%) failure rate expectation to a very low actual PPM level failure rate in the past few years.

FIGURE 5 - CYCLING MODE EFFECT ON ENDURANCE TESTING



EXAMPLE #1 - AUTOMOBILE ELECTRONIC COMPASS

A 16-byte (128-bits) segment from a 2K array stores data every time the power is turned off in an automobile electronic compass. The design engineers expect the system to power down an average of 5 times/day over a 10-year life in an 85°C case temperature environment. The projected Microchip cumulative failure rate under these conditions at the 10 year point is less than 8 PPM or 0.0008% failures in 10 years. The cumulative PPM failure rate graph is referenced on Figure 6.

EXAMPLE #2 FULL-FEATURED TELEPHONE

An 8-byte (64-bit) segment is used to store the last number redial on a stationary full feature phone operating in a room temperature environment. A 12K bit (12,288-bits) segment is used for storage of speed dial numbers on the same phone. This application has two major functions and therefore it will have two separate failure rate calculations.

The last number redial function is expected to be utilized an average of 20 times/day over a 10 year life. Each speed dial is expected to be updated an average of 0.1 times/day over a 10 year life.

Figure 7, the cumulative failure rate graph for these two conditions indicates an extremely low failure rate in the projected 10 year lifetime. The failure rate begins beyond 20 years as shown on the attached cumulative PPM failure rate graph.

EXAMPLE #3 - LASER PRINTER

A serial EEPROM could have many functions in a laser printer. A function that would likely require the most cycles/day is the maintenance log storage of the pages printed (estimated to be 100 cycles/day). Three bytes are utilized to store this number. The case temperature environment is estimated to be between 55°C and 85°C

The high number of cycles at an extreme temperature of 85°C indicate a failure rate of less than 1200 PPM thru the first 5 years and 2600 PPM thru the first 10 years.

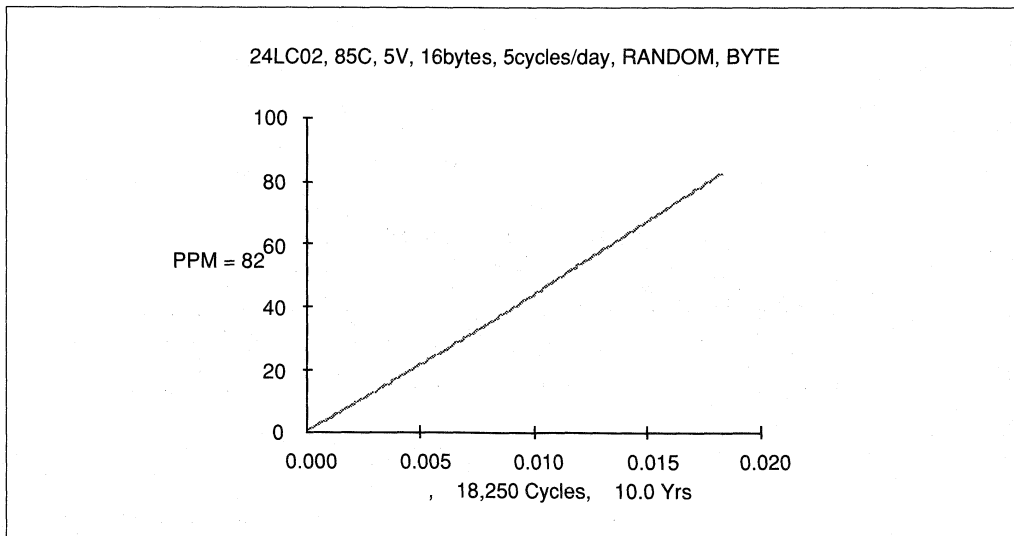
This failure rate can be dramatically reduced if the operating temperature is reduced to 55°C. The same 5 and 10 year PPM levels are reduced to 450 PPM and 1600 PPM at 55°C.

These failure rates are illustrated on Figure 8.

SUMMARY

Microchip Technology Inc. has recognized that increasing reliability in serial EEPROMs through increased endurance is not a function of extending the life of the intrinsic erase/write cycle, but depends on reducing defect-density failure rates. Design engineers characterizing EEPROM memory needs for an application and evaluating EEPROM components from various manufacturers need to understand not only the relationship between the application and expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

FIGURE 6 - EXAMPLE #1: THE AUTOMOBILE COMPASS



Serial EEPROM Endurance

Microchip has developed a program to calculate the cumulative PPM failure rates for specific system conditions, as shown on the previous pages. This program is being transferred onto menu driven DOS floppy disks. These disks will be available in December 1992 to the

system designers, reliability engineers, and component engineers to project endurance failure rates for specific system conditions. These disks will be periodically updated to reflect Microchip's most recent endurance data.

FIGURE 7 - EXAMPLE #2: THE FULL FEATURED PHONE

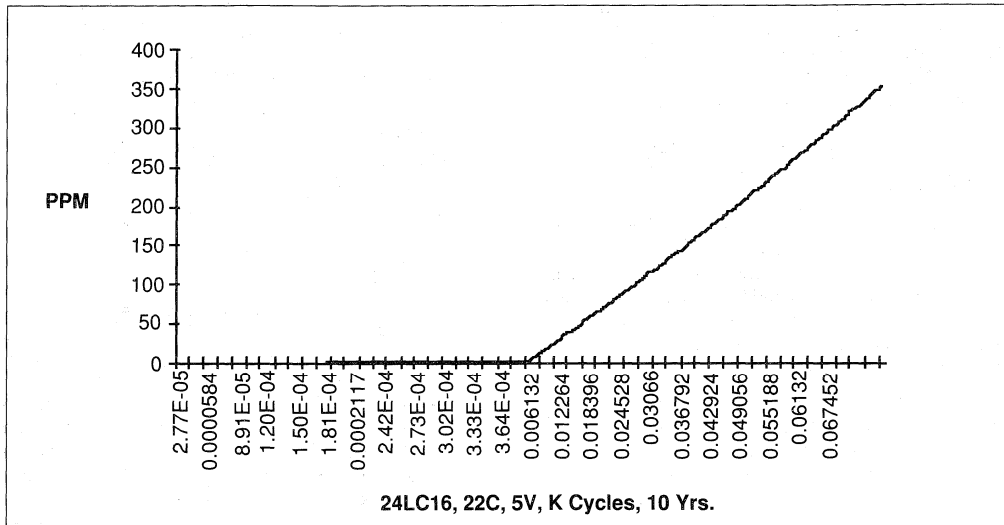
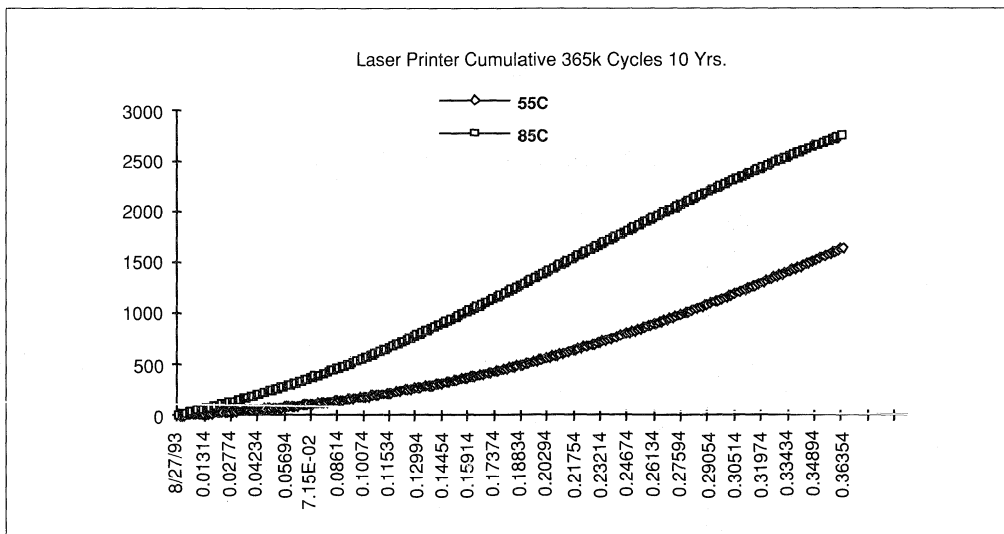


FIGURE 8 - EXAMPLE #3: THE LASER PRINTER



*AUTHOR: Steve Drehobl
Memory Products Division*

Using the Microchip Endurance Predictive Software

INTRODUCTION

Endurance, as it applies to non-volatile memory, refers to the number of times an individual memory cell can be erased and or written (some architectures do not erase before every write). Advances in process technology have made it possible to increase these limits and for Microchip Technology Inc. to offer new concepts - Total Endurance™ and a split architectural design for variable endurance. These concepts lead to more reliable products with more bits per dice, such as the 24C32 and 24C65.

TOTAL ENDURANCE™

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate (for each byte in the application not on the number of op codes or control byte commands) and is never based on any read functions whether they be a data read or configuration read. If a part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. These studies led to the concept of using the computer to predict the theoretical wear out of the floating gate and ultimately to project the point in time of a products life cycle when the first non-volatile memories bit or periphery failure should occur. After many man years of data collecting, predicting and verifying the results, Microchip feels confident in publishing and offering for the general technical community this predictive model in the form of IBM® PC-compatible software. Microchip has a patent pending on this predictive mathematical model.

TOTAL ENDURANCE PREDICTIVE SOFTWARE

The predictive software described here originally was being developed as a tool for determining endurance levels of Microchip non-volatile devices. Upon seeing the potential as a design aid, it was decided to develop a software tool that could be purchased by the engineering community and used as to predict non-volatile endurance parameters for architectural and operating parameter trade-offs before the design was completed and without subjecting the devices to prolonged incoming test for endurance levels. It should be noted that this predictive model applies only to Microchip Technology Inc. non-volatile devices.

The program uses a iterative statistical model developed by Microchip Technology Inc. physicists. The model was first used in a DOS-based text program as a proof of concept and for developing the exhaustive database needed for such a tool (included on the program disk as enddos.exe). This model was then imported to a Windows™-based software package with full GUI capabilities and all the normal cut, paste, print, viewing properties. The model actually operates as a mathematical function which is called from within the Windows Visual Basic shell and is passed all of the pertinent operational, process, and device information. The model then, after calculating the essential data points, returns this information to the main program to formatted and displayed both textually and graphically.

Applying the predictive data to the high endurance block of the 24C65, using the 24LC04 which has similar characteristics, and assuming the following:

- a five-year life
- an expected E/W cycles of 10 times per day
- a function of 11 bytes

Using Endurance Predictive Software

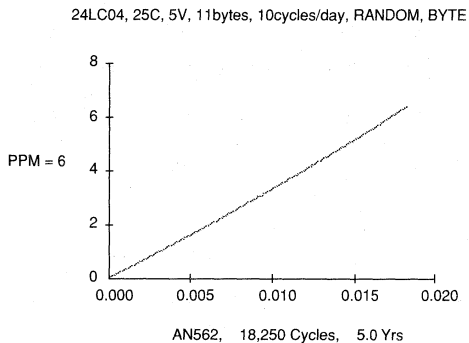
Operational specifications:

Device	24C65 (24LC04B)
Voltage	5
Temperature	25 C
Bytes/Cycle	11
E/W/Day	10
App. Life (Yr.)	5
Cycling Mode	BYTE
Data Pattern	RANDOM

The 4K HE block with 1 M E/W cycles typical, in this application, should yield the following results:

FIT	1.0 PPM	6
Time	5.0 Write cycles	18,250

FIGURE 1 -



The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance, see the related application notes AN536 and AN537 contained elsewhere in this volume. All operation parameters, along with the process technology, effect the effective endurance of a non-volatile device. The voltage, temperature, cycles per, bytes per cycle, and even the number of times written per day (time between write cycles) all have an effect on the oxide breakdown or periphery failure rate of a particular non-volatile process.

Endurance is not a well-defined concept within the semiconductor industry. The number of erase/write cycles which a particular EEPROM can endure is dependent not only upon the design of the device but also upon the application environment in which it is used. Therefore, blanket claims such as "1 million erase/write cycles typical" can only be validated based upon the specific parameters of each application. Yet until now, there has been no tool available for predicting the endurance of a

particular EEPROM device within a set of application parameters. Trade-off analysis can be painfully time-consuming and only marginally accurate without specific knowledge of the behavior of the device under different conditions of use.

The Microchip Total Endurance Software allows the designer to trade off voltage, temperature, write cycles, number of bytes written, number of writes per day, PPM and FIT rates, and years of use in order to optimize the system and accurately predict product lifetime and reliability.

The following is an example using the Endurance Software to aid in the design of an electronic phone book/auto-dialer:

The auto-dialer may have new numbers added or changed several times per day; but how can the manufacturer specify the life of the unit, and at what rate of update of the phone numbers? First, the designer must make some assumptions. If we assume that the average user will change or add 50 phone numbers per day, and the manufacturer is willing to live with a 0.1% failure rate (1,000 PPM) after 10 years of use, then we have almost enough information to verify whether we are in the ball park given the physics of the EEPROM device which will store the numbers. We also need to know the operating voltage and temperature of the application; we will say that a 3.3V lithium button battery is powering the unit and the temperature range is limited to that for which the LCD display will function: 0°C to 70°C. End-of-life voltage for the battery is approximately 2.0V; assuming that the ASIC or microcontroller in the application will operate down to 2.5V, the EEPROM also has a 2.5V requirement. The designer would like to be able to store 100 phone numbers of 16 bytes each, which results in a 1.6K byte requirement for the Serial EEPROM. Because 1.6K bytes is equal to 12.8K bits, a 16K bit 2-wire Serial EEPROM will more than suffice. Specifically, Microchip's 24LC16B will operate down to 2.5V and even includes a write-protect feature which can be used to block inadvertent writes in a noisy environment.

Here is a summary of the application:

Device	24LC16B
Voltage	2.5V - 3.3V
Temperature	0°C to 70°C (55°C typical)
Cycles per day	50
Bytes per cycle	16
Application life	10 years

Using Endurance Predictive Software

Once these values are entered into the Total Endurance program, it outputs the following:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55
Bytes/Cycle	16
E/W	50
App. Life (Yrs)	10
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
FIT	21.0
PPM	1,842
Time	10.0
Write cycles	182,500

Both of the lists above were copied directly from the Total Endurance program output to the Microsoft® Windows clipboard and pasted into this document (the Total Endurance program has a handy menu click to make this easy).

Unfortunately for our designer, the desired 0.1% failure rate has almost doubled to 0.18% (1842 PPM). But fortunately for the designer, the Total Endurance program makes trade off analysis very simple and fast. At this point there are at least three options: (1) live with almost 2000 PPM, or (2) look at the endurance plot and check whether there is a reasonable number of E/W cycles which will provide a 1000 PPM failure rate, or (3) specify a PPM rate to the Total Endurance program and let it crank out the number of cycles it will take.

Below is the endurance plot, again pasted directly from the Total Endurance program:

You can see that by reducing the number of cycles from the 182,500 which resulted from our first trial to about 100,000, we can achieve a PPM rate of about 1000 (0.1%). But how does 100,000 cycles translate into application life or cycles per day?

By switching the Total Endurance program mode to a PPM request mode instead of application life mode, we can query the program for this information. Let's ask it

for the application life of the product given a 1000 PPM failure rate. Here are the results:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55
Bytes/Cycle	16
E/W	50
PPM Level (Yrs)	1000
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
PPM	1,000
Time	5.97
Write cycles	109,000

Now we have some more options: (1) specify the product life at 5 years or (2) trade off other parameters of the application such as voltage or temperature, or (3) decide which is more important - a 10-year product lifetime, or the ability to change 50 numbers every single day. Maybe this analysis has caused our designer to re-evaluate the 50 cycle-per-day requirement. Will the user really change or add that many numbers per day - half of the unit's total capacity? Maybe 20 or even 10 is a more practical figure. Realistically, a user may enter or change quite a few numbers the first week or two of the application, and after that the unit will be used mostly for reading and dialing numbers.

Changing the number of erase/write cycles to 20 per day gives us the following results:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55
Bytes/Cycle	16
E/W	20
PPM Level (Yrs)	1000
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Using Endurance Predictive Software

Device Data: Output Parameters

PPM	1,000
Time	14.93
Write cycles	109,000

Wow! Reducing the number of cycles per day not only brought us back to a 10-year life, it gave us some margin on that, too. Keeping all the other parameters the same and forcing a 10-year lifetime gives us the following final results:

Device Data: Output Parameters

FIT	7.1
PPM	625
Time	10.0
Write cycles	73,000

The new PPM rate of 625 gives our triumphant designer more than 30% margin on his PPM target of 1000.

This example shows the significant reduction in time for design trade off analysis and time-to-market which can be achieved with a useful tool like the Microchip Total Endurance Disk. In addition, it demonstrates the increase in robustness of the system design by providing known quantities and readily accessible handles to modify those quantities in the trade-off analysis. This tool can literally reduce weeks of effort into a few minutes of point and click.

*AUTHORS: Peter Sorrells
Memory Products Division
Richard J. Fisher
Memory Products Division*

Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54

INTRODUCTION

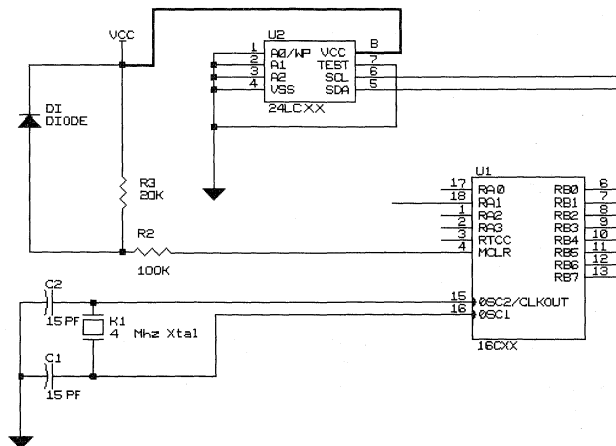
The 24LCXXB Serial EEPROMs from Microchip Technology are I²C™ compatible, will operate in the standard 100 kHz and the 400 kHz Fast Mode. When using any serial protocol with a general microcontroller that does not have a dedicated protocol specific serial port, the designer must generate the specific code routines to accomplish the several memory access functions that the microcontroller will perform. The PIC16CXX products from Microchip are both versatile and efficient to program. This application note is a series of stand-alone programs to perform the basic I²C interface functions on a PIC16C54 running at 4 MHz in XT mode. This configuration will provide a 60 kHz serial bus rate. At this speed, the interface does not meet the 100 kHz specification. If a higher clock rate is desired, the HS crystal oscillator must be used, which has a maximum fre-

quency of 20 MHz. NOTE THAT THE TIMING ROUTINES MUST BE REWRITTEN TO RUN AT THESE FASTER CLOCK RATES. The user must consult all applicable data sheets for design details. These programs have been fully tested and are being made available for general use. Figure 1 demonstrates the tested configuration of PIC16C54 and the 24LCXXB device.

This application note includes the following programs.

- 2-Wire Byte Read
- 2-Wire Byte Write
- 2-Wire Byte Write with Data Polling
- 2-Wire Page Write
- 2-Wire Sequential Read

FIGURE 1 - TESTED CONFIGURATION OF THE PIC16C54 AND THE 24LCXXB DEVICE



*AUTHOR: Bruce Negley
Memory Products Division*

Interfacing the 24LCXXB Serial EEPROMs

```
LIST P=16C54
;*****
;      2-Wire Byte Read Program (118 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24LCXXB Serial EE device
;      and perform a random read operation. This program
;      will read 8 consecutive addresses in the 'random
;      read' mode. This entails setting the address pointer
;      before doing read command for every address.
;
;      Another, more efficient method of reading consecutive
;      addresses is the 'sequential read' mode. This involves
;      sending the control byte and address for the first byte
;      to read, then continuing to provide clocks for the next
;      addresses. The device will automatically increment the
;      address. An example of the sequential read mode is
;      provided in the '2wseqr.asm' file.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 60 Khz for this setup.
;
;      As an option, the user may connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as an
;      acknowledge fail indicator. This LED will come on if
;      the serial EE fails to acknowledge correctly.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      -----
;      Pin 12 (RB6)  ->  SCLK
;      Pin 13 (RB7)  ->  SDATA
;
;      PIN 18 (RA1)  ->  Acknowledge fail LED (Optional)
;
;*****
;      Register Definitions
;*****
port_a equ 5h      ; port 5 (port a) used for LED display
port_b equ 6h      ; port 6 (port b) used for data and
                   ; clock lines
eeprom equ 0ah     ; bit buffer
bycnt  equ 0bh     ; byte counter for read mode
addr   equ 0ch     ; address counter
datai  equ 0dh     ; data input register
datao  equ 0eh     ; data output register
slave  equ 0fh     ; device address 1010xxx0)
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bit counter
bcount equ 12h     ; byte counter
loops  equ 15h     ; delay loop counter
loops2 equ 16h     ; delay loop counter

;*****
;      Bit Definitions
;*****
d      equ 7       ; eeprom input bit
do     equ 6       ; eeprom output bit
sdata  equ 7       ; serial EE data line (port_b, pin 13)
sclk   equ 6       ; serial EE clock line (port_b, pin 12)
ackf   equ 1       ; acknowledge fail LED (port_a)
;*****
;
;      org 01ffh      ; set reset vector
;      goto PWRUP
;      org 000h      ;
;      goto PWRUP
```

Interfacing the 24LCXXB Serial EEPROMs

```
;
;*****
;      Start Bit Subroutine
;      this routine generates a start bit
;      (Low going data line while clock is high)
;*****
;
BSTART
    bcf    port_b,sdata    ; make sure data is high
    movlw  b'00111111'
    tris   port_b          ; set data and clock lines for output
    bcf    port_b,sclk     ; make sure clock is low
    nop
    bsf    port_b,sclk     ; set clock high
    nop
    nop
    nop
    nop
    bcf    port_b,sdata    ; data line goes low during
                          ; high clock for start bit
    nop
    nop
    nop
    nop
    nop
    nop
    bcf    port_b,sclk     ; timing adjustment
                          ; start clock train
    nop
    nop
    retlw  0
;
;*****
;      Stop Bit Subroutine
;      This routine generates a stop bit
;      (High going data line while clock is high)
;*****
BSTOP
    bcf    port_b,sdata    ; make sure data line is low
    movlw  b'00111111'
    tris   port_b          ; set data/clock lines as outputs
    bcf    port_b,sdata    ; make sure data line is low
    nop
    nop
    bsf    port_b,sclk     ; set clock high
    nop
    nop
    nop
    bsf    port_b,sdata    ; data goes high while clock high
                          ; for stopbit
    nop
    nop
    bcf    port_b,sclk     ; set clock low again
    nop
    nop
    nop
    retlw  0
;
;*****
;      BITOUT routine takes the bit of data in 'do' and
;      transmits it to the serial EE device
;*****
BITOUT
    movlw  b'00111111'    ; set data, clock as outputs
    tris   port_b
    btfss  eeprom,do      ; check for state of data bit to xmit
    goto   bitlow         ; low? go set data line low
    bsf    port_b,sdata    ; high? set data line high
    goto   clkout         ; go toggle the clock
```

Interfacing the 24LCXXB Serial EEPROMs

```
bitlow bcf port_b,sdata ; output a low bit
clkout bsf port_b,sclk ; set clock line high
nop
nop
nop
nop
bcf port_b,sclk ; return clock line low
retlw 0
;
;*****
; BITIN routine reads one bit of data from the
; serial EE device and stores it in the bit 'di'
;*****
BITIN
    bsf eeprom,di ; assume input bit is high
    movlw b'10111111' ; make sdata an input line
    tris port_b
    bsf port_b,sdata ; set sdata line for input
    bsf port_b,sclk ; set clock line high
    nop ; just sit here a sec
    nop
    nop
    nop
    btfs port_b,sdata ; read the data bit
    bcf eeprom,di ; input bit was low, set 'di' accordingly
    bcf port_b,sclk ; set clock line low
    ;
    retlw 0
;
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw .8
    movwf count ; set the #bits to 8
    ;
TXLP
    bcf eeprom,do ; assume bit out is low
    btfs txbuf,7 ; is bit out really low?
    bsf eeprom,do ; no, set it high
    call BITOUT ; send the bit to serial EE
    hf txbuf ; rotate txbuf left
    decfsz count ; 8 bits done?
    goto TXLP ; no - go again
    call BITIN ; read ack bit
    btfs eeprom,di ; check ack bit
    bsf port_a,ackf ; set acknowledge fail LED if the
    ; device did not pull data low
    ;
    retlw 0
;
;*****
; Receive data routine
; This routine reads one byte of data from the part
; into the 'data1' register. It then sends a high
; ack bit to indicate that no more data is to be read
;*****
RX
    cldf data1 ; clear input buffer
    movlw .8 ; set #bits to 8
```


Interfacing the 24LCXXB Serial EEPROMs

```

movwf count
RXLP rlf datai ; rotate data i 1 bit left
call BITIN ; read a bit
btfsz eeprom,di
bsf datai,0 ; set bit 0 if necessary
decfsz count ; 8 bits done?
goto RXLP ; no, do another
bsf eeprom,do ; set ack bit = 1
call BITOUT ; to finish transmission
retlw 0
;
;*****
; Power up routine
; This is the program entry point. I/O line status is
; set for port A here.
;*****
PWRUP
movlw b'00000000'
tris port_a ; set port A as all output
clr port_a ; all output lines low
goto READ
;
;*****
; READ (read routine)
; This routine reads 8 consecutive addresses of the
; serial EE device starting at address 00 in the
; random access mode (non-sequential read). Reading
; the device using the random access mode
; requires that the address pointer be set for every
; byte before the read takes place. The address pointer
; is set by sending a 'write mode' control byte with the
; address of the location to read.
;*****
READ
;
bcf port_a,ackf ; clear the ack fail LED if on
movlw .8 ; set number of bytes to read as 8
movwf bcount ;
clr addr ; set starting address to 00
;
rbyte call BSTART ; generate start bit
;
; now send the write control byte and
; address to set the pointer
;
movlw b'10100000' ; get slave address (write mode)
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; get word address
movwf txbuf ; into transmit buffer
call TX ; and send it
;
; now read one byte from the part
;
call BSTART ; generate start bit
movlw b'10100001' ; get slave address and read mode
movwf txbuf ; into transmit buffer
call TX ; and transmit it
call RX ; read 1 byte from serial EE
call BSTOP ; send stop bit to end transmission
incf addr ; add 1 to address counter
decfsz bcount ; yes, are all 8 bytes read?
goto rbyte ; no, do another byte
goto READ ; yes, start all over
;
END
```

Interfacing the 24LCXXB Serial EEPROMs

```
LIST P=16C54
;*****
;      2-Wire Byte Write Program (123 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24LCXX Serial EE device
;      and perform a byte write operation on 8 consecutive
;      addresses.
;
;      This routine waits approximately 10mS for the write
;      cycle time, which is enough time for any of the
;      24LCxx devices to complete a write. A more efficient
;      method of determining when the write cycle is complete
;      is called "data polling." The data polling method is
;      explained in the program "2wcpoll.asm." That
;      particular program uses data polling in a byte write
;      mode but it can be used in exactly the same way for
;      a page mode write.
;
;      As an option, the user may connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as an
;      acknowledge fail indicator. This LED will come on if
;      the serial EE fails to acknowledge correctly.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 60 KHz for this setup.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      - - - - -      - - - - -
;      Pin 12 (RB6)  ->  SCLK
;      Pin 13 (RB7)  ->  SDATA
;
;      PIN 18 (RA1)  ->  Acknowledge fail LED (Optional)
;
;*****
;      Register Definitions
;*****
port_a equ 5h      ; port 5 (port_a) used for LEDs
port_b equ 6h      ; port 6 (port_b) used for data and
                   ; clock lines
eeprom equ 0ah     ; bit buffer
bycnt  equ 0bh     ; byte counter for read mode
addr   equ 0ch     ; address counter
datai  equ 0ch     ; data input register
datao  equ 0eh     ; data output register
slave  equ 0fh     ; device address
                   ; (1010xxx0)
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bit counter
bcount equ 12h     ; byte counter
loops  equ 15h     ; delay loop counter
loops2 equ 16h     ; delay loop counter
;*****
;      Bit Definitions
;*****
d      equ 7      ; eeprom input bit
do     equ 6      ; eeprom output bit
sdata  equ 7      ; serial EE data line (port_b, pin 13)
sclk   equ 6      ; serial EE clock line (port_b, pin 12)
ackf   equ 1      ; acknowledge fail LED (port_a, pin 18)
;*****
;
;      org 01ffh      ; set reset vector
;      goto PWRUP
;      org 000h      ;
;      goto PWRUP
```

Interfacing the 24LCXXB Serial EEPROMs

```
;
;*****
;      DELAY ROUTINE
;      This routine takes the value in 'loops'
;      and multiplies it times 1 millisecond to
;      determine delay time.
;*****
WAIT
;
top    movlw  .110    ; timing adjustment variable
      movwf  loops2
top2   nop          ; sit and wait
      nop
      nop
      nop
      nop
      decfsz loops2 ; inner loops complete?
      goto  top2   ; no, go again
      ;
      decfsz loops ; outer loops complete?
      goto  top    ; no, go again
      retlw  0     ; yes, return from sub
;
;*****
;      Start Bit Subroutine
;      this routine generates a start bit
;      (Low going data line while clock is high)
;*****
BSTART
      bsf    port_b,sdata ; make sure data is high
      movlw b'00111111'
      tris  port_b       ; set data and clock lines for output
      bcf   port_b,sclk  ; make sure clock is low
      nop
      bsf   port_b,sclk  ; set clock high
      nop
      nop
      nop
      nop
      bcf   port_b,sdata ; data line goes low during
                        ; high clock for start bit
      nop
      nop
      nop
      nop
      nop
      bcf   port_b,sclk  ; start clock train
      nop
      nop
      retlw 0
;
;      End of Subroutine
;*****
;      Stop Bit Subroutine
;      This routine generates a stop bit
;      (High going data line while clock is high)
;*****
BSTOP
      movlw b'00111111' ;
      tris  port_b       ; set data/clock lines as outputs
      bcf   port_b,sdata ; make sure data line is low
      nop
      nop
      nop
      bsf   port_b,sclk  ; set clock high
      nop
      nop
```

Interfacing the 24LCXXB Serial EEPROMs

```
        nop
        bsf    port_b,sdata    ; data goes high while clock high
                                ; for stopbit

        nop
        nop
        bcf    port_b,sclk     ; set clock low again
        nop
        nop
        nop
        retlw  0
;
;      End of Subroutine
;*****
;      BITOUT routine takes one bit of data in 'do' and
;      transmits it to the serial EE device
;*****
BITOUT
        movlw  b'00111111'    ; set data, clock as outputs
        tris   port_b
        btfs   eepon, do      ; check for state of data bit to xmit
        goto   bitlow         ;
        bsf    port_b,sdata    ; set data line high
        goto   clkout         ; go toggle the clock

bitlow  bcf    port_b,sdata    ; output a low bit
clkout  bsf    port_b,sclk     ; set clock line high
        nop
        nop
        nop
        nop
        bcf    port_b,sclk     ; return clock line low
        retlw  0
;
;      End of Subroutine
;*****
;      BITIN routine reads one bit of data from the
;      serial EE device and stores it in 'di'
;*****
BITIN
        bsf    eepon,di        ; assume input bit is high
        movlw  b'10111111'    ; make sdata an input line
        tris   port_b
        bsf    port_b,sdata    ; set sdata line for input
        bsf    port_b,sclk     ; set clock line high
        nop    ; just sit here a sec
        nop
        nop
        nop
        nop
        btfs   port_b,sdata    ; read the data bit
        bcf    eepon,di        ; input bit was low
        bcf    port_b,sclk     ; set clock line low
        ;
        retlw  0
;
;*****
;      Transmit Data Subroutine
;      This routine takes the byte of data stored in the
;      'datao' register and transmits it to the serial EE device.
;      It will then send 1 more clock to the serial EE for the
;      acknowledge bit. If the ack bit from the part was low
;      then the transmission was successful. If it is high, then
;      the device did not send a proper ack bit and the ack
;      fail LED will be turned on.
;*****
TX
        movlw  .8
        movwf  count          ; set the #bits to 8
        ;
```

Interfacing the 24LCXXB Serial EEPROMs

```

TXLP      bcf      eeprom,do      ; assume bit out is low
          btfsz   txbuf,7         ; is bit out really low?
          bsf     eeprom,do      ; otherwise data bit =1
          call    BITOUT         ; serial data out
          rlf     txbuf          ; rotate txbuf left
          decfsz  count          ; 8 bits done?
          goto    TXLP          ; no - go again
          call    BITIN         ; read ack bit
          btfsz   eeprom,di      ; check ack bit
          bsf     port_a,ackf    ; set acknowledge fail LED if the
          ;
          retlw   0

;
;*****
;
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; byte write routine.
;*****
PWRUP     movlw   b'00000000'
          tris   port_a         ; set port A as all output
          crrf   port_a         ; all output lines low
          goto   WRBYTE

;*****
;
; Byte Write Routine
; This routine writes the data in "data0" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. This routine waits 10ms after every
; byte to give the device time to do the write. This
; program repeats forever.
;*****
WRBYTE    ;
          crrf   port_a         ; clear all LEDs
          movlw  b'10100000'    ; set slave address and write mode
          movwf  slave
          movlw  b'01010101'    ; set data to write as 55h
          movwf  data0

;
          movlw  .8             ; set number of bytes
          movwf  bcount         ; to write as to 8
          crrf   addr          ; set starting address to 00
          ;
byte      call    BSTART        ; generate start bit
          movf   slave,w        ; move slave address
          movwf  txbuf          ; into transmit buffer
          call   TX             ; and send it
          movf   addr,w         ; move word address
          movwf  txbuf          ; into transmit buffer
          call   TX             ; and send it
          movf   data0,w        ; move data byte
          movwf  txbuf          ; to transmit buffer
          call   TX             ; and transmit it
          call   BSTOP         ; generate stop bit
          ;

          movlw  .10            ; set delay time to give
          movwf  loops          ; 10 ms wait after every byte
          call   WAIT          ; 10 ms wait after every byte
          incf   addr           ; add 1 to address counter
          decfsz bcount         ; all 8 bytes written?
          goto  byte           ; no, do another byte
          ;
          goto  wrbyte         ; start over

;
;
END

```

Interfacing the 24LCXXB Serial EEPROMs

```
LIST P=16C54
;*****
;      2-Wire Byte Write With Data Polling Program (129 bytes)
;
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24LCXX Serial EE device.
;      This program performs a byte write operation on 8
;      consecutive addresses using the data polling method
;      to determine when the write cycle is complete.
;
;
;      When writing to a serial E2 device, there are 2
;      ways to handle the internal timed write cycle
;      time of the device. The simplest method is
;      simply to wait until the maximum cycle time
;      is exceeded before attempting another command.
;      The other, more efficient method is known as "data
;      polling." Data polling is done by sending a start
;      bit and control byte to the part after the write
;      cycle has been initiated by a stop bit. If the ack bit
;      is low, then the device is through writing, otherwise
;      the the sequence is repeated. If no low acknowledge
;      is found within 40 attempts (about 10 milliseconds)
;      then the routine times out and sets the timeout
;      LED (pin 1) high.
;
;
;      As an option, the user can connect a LED to pin 1
;      on the PIC16CXX (use about a 1K resistor in series) as a
;      timeout indicator. This LED will come on if the
;      data polling is unsuccessful and the device being
;      programmed does not respond. This can be tested by
;      simply running the program with no part in the socket.
;
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 40 KHz for this setup.
;
;
;      PIC16CXX to Serial EE Connections:
;
;
;      PIC16CXX      Serial EE
;      - - - - -    - - - - -
;      Pin 12 (RB6) -> SCLK
;      Pin 13 (RB7) -> SDATA
;
;
;      Pin 1  (RA2) -> Write cycle timeout fail LED (optional)
;
;*****
;      Register Definitions
;*****
port_a  equ  5h      ; port 5 (port_a) used LEDs
port_b  equ  6h      ; port 6 (port_b) used for data and
                    ; clock lines
eeprom  equ  0ah     ; bit buffer
bycnt   equ  0bh     ; byte counter for read mode
addr    equ  0ch     ; address counter
datai   equ  0dh     ; data input register
datao   equ  0eh     ; data output register
slave   equ  0fh     ; device address (1010xxx0)
txbuf   equ  10h     ; transmit buffer
count   equ  11h     ; bit counter
bcount  equ  12h     ; byte counter
loops   equ  15h     ; delay loop counter
loops2  equ  16h     ; delay loop counter
pollcnt equ  17h     ; data polling counter
;*****
;      Bit Definitions
;*****
d       equ  7       ; eeprom input bit
do      equ  6       ; eeprom output bit
sdata   equ  7       ; serial EE data line (port_b, pin 13)
```

Interfacing the 24LCXXB Serial EEPROMs

```
sclk equ 6 ; serial EE clock line (port_b, pin 12)
timeout equ 2 ; write cycle timeout fail LED, port_a (pin 1)
ackf equ 1 ; acknowledge fail LED, port_a (pin 18)
;*****
;
org 01ffh ; set reset vector
goto PWRUP
org 000h ;
goto PWRUP
;
;*****
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;*****
WAIT
;
top movlw .110 ; timing adjustment variable
movwf loops2
top2 nop ; sit and wait
nop
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again
;
decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub
;
;*****
; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;*****
BSTART
bsf port_b, sdata ; make sure data is high
movlw b'00111111'
tris port_b ; set data and clock lines for output
bcf port_b, sclk ; make sure clock is low
nop
bsf port_b, sclk ; set clock high
nop
nop
nop
nop
nop
bcf port_b, sdata ; data line goes low during
; high clock for start bit
nop
nop
nop
nop
nop
bcf port_b, sclk ; start clock train
nop
nop
retlw 0
;
;*****
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;*****
BSTOP
```

Interfacing the 24LCXXB Serial EEPROMs

```
        movlw   b'00111111'   ;
        tris    port_b         ; set data/clock lines as outputs
        bcf     port_b,sdata   ; make sure data line is low
        nop
        nop
        nop
        bsf     port_b,sclk    ; set clock high
        nop
        nop
        nop
        bsf     port_b,sdata   ; data goes high while clock high
                                ; for stopbit

        nop
        nop
        bcf     port_b,sclk    ; set clock low again
        nop
        nop
        nop
        retlw   0
;
;*****
;   BITOUT routine takes one bit of data in 'do' and
;   transmits it to the serial EE device
;*****
BITOUT
        movlw   b'00111111'   ; set data, clock as outputs
        tris    port_b
        btfss   eeprom,do     ; check for state of data bit to xmit
        goto    bitlow        ;
        bsf     port_b,sdata   ; set data line high
        goto    clkout        ; go toggle the clock
                                ;
bitlow   bcf     port_b,sdata   ; output a low bit
clkout   bsf     port_b,sclk   ; set clock line high
        nop
        nop
        nop
        nop
        bcf     port_b,sclk    ; return clock line low
        retlw   0
;
;   End of Subroutine
;
;*****
;   BITIN routine reads one bit of data from the
;   serial EE device and stores it in 'di'
;*****
BITIN
        bsf     eeprom,di     ; assume input bit is high
        movlw   b'10111111'   ; make sdata an input line
        tris    port_b
        bsf     port_b,sdata   ; set sdata line for input
        bsf     port_b,sclk    ; set clock line high
        nop
        nop                    ; just sit here a sec
        nop
        nop
        nop
        nop
        btfss   port_b,sdata   ; read the data bit
        bcf     eeprom,di     ; input bit was low
        bcf     port_b,sclk    ; set clock line low
                                ;
        retlw   0
;
;*****
;   Transmit Data Subroutine
;   This routine takes the byte of data stored in the
;   'datao' register and transmits it to the serial EE device.
;   It will then send 1 more clock to the serial EE for the
;   acknowledge bit. If the ack bit from the part was low
```


Interfacing the 24LCXXB Serial EEPROMs

```
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw    .8
    movwf    count        ; set the #bits to 8
;
;
TXLP
    bcf      eeprom,do    ; assume bit out is low
    btfscc  txbuf,7       ; is bit out really low?
    bsf      eeprom,do    ; otherwise data bit =1
    call    BITOUT        ; serial data out
    rlf      txbuf         ; rotate txbuf left
    decfsz  count         ; 8 bits done?
    goto    TXLP          ; no - go again
    call    BITIN         ; read ack bit
;
;
    retlw   0
;
;
;*****
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; write routine.
;*****
PWRUP
    movlw   b'00000000'
    tris    port_a        ; set port A as all output
    clrf    port_a        ; all output lines low
    goto    WRBYTE

;*****
; Byte Write Routine with data polling technique
;
; This routine writes the data in "data0" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. To determine when the write cycle time
; of the device, the 'data polling' method is used. This
; involves sending a start bit and control byte to the part
; and checking for an acknowledge. If the ack bit is low, then
; the device is through writing, otherwise the the sequence
; is repeated. If no low acknowledge is found within 40
; attempts (about 10 milliseconds) then the routine times
; out and sets the timeout LED (pin 18) high. This program
; will repeat forever.
;*****
;
WRBYTE
;
    clrf    port_a        ; all LEDs off
    movlw   b'10100000'   ; set slave address and write mode
    movwf   slave
    movlw   b'10101010'   ; set data to write as AAh
    movwf   data0
    movlw   .8             ; set number of bytes
    movwf   bcount
    clrf    addr          ; set starting address to 00
;
byte
    call    BSTART        ; generate start bit
    movf    slave,w       ; move slave address
    movwf   txbuf         ; into transmit buffer
    call    TX            ; and send it
    movf    addr,w       ; move word address
    movwf   txbuf         ; into transmit buffer
    call    TX            ; and send it
    movf    data0,w      ; move data byte
    movwf   txbuf         ; to transmit buffer
```

Interfacing the 24LCXXB Serial EEPROMs

```
    call    TX           ; and transmit it
    call    BSTOP        ; generate stop bit
                        ;
                        ; now start polling for a low ack bit
                        ;
    movlw   .40          ;
    movwf   pollcnt      ; set max number of times to poll as 40
poll    call    BSTART    ; generate start bit
    movlw   b'10100000' ; move slave address (write mode)
    movwf   txbuf        ; into transmit buffer
    call    TX           ; and send it
    btfss  eeprom,di     ; was the ack bit low?
    goto   exitpoll     ; yes, do another byte
    decfsz pollcnt       ; is poll counter down to zero?
    goto   poll         ; no, poll again. Other wise the part is
    bsf    port_a,timeout ; not responding in time so set timeout
                        ; LED and continue on
    ;
exitpoll ircf   addr     ; add 1 to address counter
    decfsz bcount      ; all 8 bytes written?
    goto   byte        ; no, do another byte
    goto   WRBYTE      ; yes, start over
;
;
END
```

Interfacing the 24LCXXB Serial EEPROMs

```
LIST P=16C54
;*****
;      2-Wire Page Write Program (122 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC 16C54 to a 24LCXX Serial EE device
;      and perform a page write operation on 8 consecutive
;      addresses.
;
;      All of Microchip's 2-wire serial EE devices have a
;      buffer or 'page' that can be used as a more efficient
;      method of writing to consecutive addresses. Some
;      devices have page lengths of 2 bytes, others have
;      page lengths of 4, 8, 16 or 64 bytes. Please consult
;      the databook for the page length of the device you
;      are using. THIS ROUTINE IS WRITTEN FOR A DEVICE
;      WITH A PAGE LENGTH OF 8 OR MORE BYTES.
;
;      When using page mode, the control byte and address
;      are sent for the first address only. After the data
;      byte for the first address is sent, the data for the
;      next consecutive address is clocked in. This is
;      repeated as many times as needed (as long as the page
;      length is not exceeded) and then a stop bit is sent.
;      The device will still acknowledge between every byte of
;      data. After the stop bit is sent, the part will
;      initiate the self timed write cycle. For all of
;      Microchip's 24LCxx devices, the cycle time for
;      a byte write and a page write is the same. Therefore,
;      writing 8 bytes in byte mode with a typical 5ms write
;      cycle consumes 40ms of wait time, while the same data
;      written in page mode would consume only 5ms of wait time.
;
;      This routine waits approximately 10mS for the write
;      cycle time, which is enough time for any of the
;      24LCxx devices to complete a write. A more efficient
;      method of determining when the write cycle is complete
;      is called "data polling." The data polling method is
;      explained in the program "2wdpoll.asm." That
;      particular program uses data polling in a byte write
;      mode but it can be used in exactly the same way for
;      a page mode write.
;
;      As an option, the user can connect a LED to pin 18
;      on the PIC (use about a 1K resistor in series) as a
;      acknowledge fail indicator. This LED will come on
;      if the device being programmed does not send a low
;      acknowledge bit at the proper times. This can be
;      tested by simply running the program with no part
;      in the socket.
;
;      Timing is based on using the PIC in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 40 Khz for this setup.
;
;      PIC to Serial EE Connections:
;
;      PIC          Serial EE
;      - - - - -
;      Pin 12 (RB6) -> SCLK
;      Pin 13 (RB7) -> SDATA
;
;      PIN 18 (RA1) -> Acknowledge fail LED (Optional)
;
;*****
;      Register Definitions
;*****
```



Interfacing the 24LCXXB Serial EEPROMs

```
port_a equ 5h ; port 5 (port a) used for LED outputs
port_b equ 6h ; port 6 (port b) used for data and
; clock lines
eeprom equ 0ah ; bitbuffer
addr equ 0ch ; address counter
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
;*****
; BitDefinitions
;*****
d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b,pin 13)
sclk equ 6 ; serial EE clock line (port_b,pin 12)
ackf equ 1 ; acknowledge fail LED, port_a (pin 18)
;*****
;
; org 01ffh ; set reset vector
; goto PWRUP
; org 000h ;
; goto PWRUP
;
;*****
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;*****
WAIT
;
top movlw .110 ; timing adjustment variable
movwf loops2
top2 nop ; sit and wait
nop
nop
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again
;
decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub
;
;*****
; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;*****
;
BSTART
bsf port_b,sdata ; make sure data is high
movlw b'00111111'
tris port_b ; set data and clock lines for output
bcf port_b,sclk ; make sure clock is low
nop
bsf port_b,sclk ; set clock high
nop
nop
nop
nop
nop
```

Interfacing the 24LCXXB Serial EEPROMs

```

        bcf     port_b,sdata    ; data line goes low during
                                ; high clock for start bit

        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; timing adjustment
                                ; start clock train
        nop
        nop
        retlw  0
    ;
    ;         End of Subroutine
;*****
;         Stop Bit Subroutine
;         This routine generates a stop bit
;         (High going data line while clock is high)
;*****
BSTOP
        movlw  b'00111111'    ;
        tris   port_b         ; set data/clock lines as outputs
        bcf   port_b,sdata    ; make sure data line is low
        nop
        nop
        nop
        bsf   port_b,sclk     ; set clock high
        nop
        nop
        nop
        bsf   port_b,sdata    ; data goes high while clock high
                                ; for stop bit

        nop
        nop
        bcf   port_b,sclk     ; set clock low again
        nop
        nop
        nop
        retlw  0
    ;
    ;         End of Subroutine
;*****
;         BITOUT routine
;         This routine takes one bit of data in 'do' and
;         transmits it to the serial EE device
;*****
BITOUT
        movlw  b'00111111'    ; set data, clock as outputs
        tris   port_b
        btfss  eeprom,do      ; check for state of data bit to xmit
        goto  bitlow         ;
        bsf   port_b,sdata    ; set data line high
        goto  clkout         ; go toggle the clock

bitlow  bcf   port_b,sdata    ; output a low bit
clkout  bsf   port_b,sclk     ; set clock line high
        nop
        nop
        nop
        nop
        bcf   port_b,sclk     ; return clock line low
        retlw  0
    ;
    ;         End of Subroutine
;*****
;         BITIN routine reads one bit of data from the
;         serial EE device and stores it in 'di'
;*****
BITIN

```

Interfacing the 24LCXXB Serial EEPROMs

```

    bsf     eeprom,di       ; assume input bit is high
    movlw  b'10111111'    ; make sdata an input line
    tris   port_b
    bsf    port_b,sdata    ; set sdata line for input
    bsf    port_b,sclk     ; set clock line high
    nop
    nop
    nop
    nop
    nop
    nop
    btfss  port_b,sdata    ; read the data bit
    bcf    eeprom,di      ; input bit was low
    bcf    port_b,sclk     ; set clock line low
    ;
    retlw  0              ;

;*****
;
; Transmit Data Subroutine
;
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw  .8
    movwf  count          ; set the #bits to 8
;
TXLP
    bcf    eeprom,do      ; assume bit out is low
    btfs   txbuf,7        ; is bit out really low?
    bsf    eeprom,do      ; otherwise data bit =1
    call   BITOUT         ; serial data out
    rlf   txbuf           ; rotate txbuf left
    decfsz count         ; 8 bits done?
    goto  TXLP           ; no - go again
    call   BITIN         ; read ack bit
    btfs   eeprom,di     ; check ack bit
    bsf    port_a,ackf    ; set acknowledge fail LED if the
    ;                    ; device did not pull data low
;
    retlw  0
;
;*****
;
; Power up routine
;
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the Page
; write routine.
;*****
PWRUP
    movlw  b'00000000'
    tris   port_a        ; set port A as all output
    clrf  port_a         ; all output lines low
    goto  WRPAGE
;
;*****
;
; Page Write Routine
;
; This routine uses page mode to write the data in "data0" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. This routine waits 10mS after every
; page to give the device time to do the write. This
; routine executes forever
;*****
WRPAGE
;

```

Interfacing the 24LCXXB Serial EEPROMs

```
    clrf    port_a        ; clear all LEDs
    movlw  b'10100000'   ; set slave address and write mode
    movwf  slave
    movlw  b'01010101'   ; set data to write as 55h
    movwf  data0
    ;
    movlw  .8            ; set number of bytes
    movwf  bcount
    clrf   addr          ; set starting address to 00
    ;
    call   BSTART        ; generate start bit
    movf   slave,w       ; move slave address
    movwf  txbuf         ; into transmit buffer
    call   TX            ; and send it

    movf   addr,w        ; move word address
    movwf  txbuf         ; into transmit buffer
    call   TX            ; and send it

byte   movf   data0,w    ; move data byte
       movwf txbuf      ; to transmit buffer
       call   TX        ; and transmit it
       decfsz bcount    ; all 8 bytes written?
       goto  byte       ; no, do another
       call   BSTOP     ; yes, generate stop bit
       ;
       movlw  .10       ; set delay time to give
       movwf  loops     ; set delay time to give
       call   WAIT      ; 10 ms wait after every byte
       ;
       goto  wrpage     ; start over
;
;
END
```

Interfacing the 24LCXXB Serial EEPROMs

```
LIST P=16C54
;*****
;      2-Wire Sequential Read Program (120 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24LCXX Serial EE device
;      and perform a sequential read operation. A sequential
;      read involves setting the address pointer once and then
;      using the auto-increment ability of the part to read
;      consecutive addresses by simply providing more clocks.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 60 Khz for this setup.
;
;      As an option, the user may connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as an
;      acknowledge fail indicator. This LED will come on if
;      the serial EE fails to acknowledge correctly.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      -----
;      Pin 12 (RB6)  ->  SCLK
;      Pin 13 (RB7)  ->  SDATA
;
;      PIN 18 (RA1)  ->  Acknowledge fail LED (Optional)
;
;*****
;      Register Definitions
;*****
port_a  equ  5h      ; port 5 (port_a) used for LEDs
port_b  equ  6h      ; port 6 (port_b) used for data and
                    ; clock lines
eeprom  equ  0ah     ; bit buffer
bycnt   equ  0bh     ; byte counter for read mode
addr    equ  0ch     ; address counter
datai   equ  0dh     ; data input register
datao   equ  0eh     ; data output register
slave   equ  0fh     ; device address (1010xxx0)
txbuf   equ  10h     ; transmit buffer
count   equ  11h     ; bit counter
bcount  equ  12h     ; byte counter
loops   equ  15h     ; delay loop counter
loops2  equ  16h     ; delay loop counter
;*****
;      Bit Definitions
;*****
d       equ  7       ; eeprom input bit
do      equ  6       ; eeprom output bit
sdata   equ  7       ; serial EE data line (port_b, pin 13)
sclk    equ  6       ; serial EE clock line (port_b, pin 12)
ackf    equ  1       ; acknowledge fail LED, port_a
;*****
;
;      org      01ffh  ; set reset vector
;      goto    PWRUP
;      org      000h   ;
;      goto    PWRUP
;
;*****
;      Start Bit Subroutine
;      this routine generates a start bit
;      (Low going data line while clock is high)
;*****
;
;*****
BSTART
```


Interfacing the 24LCXXB Serial EEPROMs

```
    bsf     port_b,sdata      ; make sure data is high
    movlw  b'00111111'
    tris   port_b            ; set data and clock lines for output
    bcf     port_b,sclk      ; make sure clock is low
    nop
    bsf     port_b,sclk      ; set clock high
    nop
    nop
    nop
    nop
    bcf     port_b,sdata      ; data line goes low during
                                ; high clock for start bit

    nop
    nop
    nop
    nop
    nop
    nop
    bcf     port_b,sclk      ; timing adjustment
                                ; start clock train
    nop
    nop
    retlw  0
    ;
    ;           End of Subroutine
;*****
;           Stop Bit Subroutine
;           This routine generates a stop bit
;           (High going data line while clock is high)
;*****
BSTOP
    movlw  b'00111111'      ;
    tris   port_b            ; set data/clock lines as outputs
    bcf     port_b,sdata      ; make sure data line is low
    nop
    nop
    nop
    bsf     port_b,sclk      ; set clock high
    nop
    nop
    nop
    bsf     port_b,sdata      ; data goes high while clock high
                                ; for stopbit

    nop
    nop
    bcf     port_b,sclk      ; set clock low again
    nop
    nop
    nop
    retlw  0
;
;           End of Subroutine
;*****
;           BITOUT routine takes one bit of data in 'do' and
;           transmits it to the serial EE device
;*****
BITOUT
    movlw  b'00111111'      ; set data, clock as outputs
    tris   port_b
    btfss  eeprom,do        ; check for state of data bit to xmit
    goto   bitlow           ;
    bsf     port_b,sdata      ; set data line high
    goto   clkout           ; go toggle the clock

bitlow  bcf     port_b,sdata      ; output a low bit
clkout  bsf     port_b,sclk      ; set clock line high
    nop
    nop
    nop
```

Interfacing the 24LCXXB Serial EEPROMs

```
        bcf     port_b,sclk ; return clock line low
        retlw  0
;
;       End of Subroutine
;
;*****
;       BITIN routine reads one bit of data from the
;       serial EE device and stores it in 'di'
;*****
BITIN
        bsf     eeprom,di    ; assume input bit is high
        movlw  b'10111111'  ; make sdata an input line
        tris   port_b
        bsf     port_b,sdata ; set sdata line for input
        bsf     port_b,sclk  ; set clock line high
        nop    ; just sit here a sec
        nop
        nop
        nop
        btfsz  port_b,sdata  ; read the data bit
        bcf     eeprom,di    ; input bit was low, set 'di' accordingly
        bcf     port_b,sclk  ; set clock line low
        ;
        retlw  0
;
;*****
;       Transmit Data Subroutine
;       This routine takes the byte of data stored in the
;       'data0' register and transmits it to the serial EE device.
;       It will then send 1 more clock to the serial EE for the
;       acknowledge bit. If the ack bit from the part was low
;       then the transmission was successful. If it is high, then
;       the device did not send a proper ack bit and the ack
;       fail LED will be turned on.
;*****
TX
        movlw  .8
        movwf  count        ; set the #bits to 8
        ;
TXLP
        bcf     eeprom,do    ; assume bit out is low
        btfsz  txbuf,7      ; is bit out really low?
        bsf     eeprom,do    ; no, set it high
        call   BITOUT       ; send the bit to serial EE
        rlf   txbuf         ; rotate txbuf left
        decfsz count        ; 8 bits done?
        goto  TXLP         ; no - go again
        call   BITIN        ; read ack bit
        btfsz  eeprom,di    ; check ack bit
        bsf     port_a,ackf  ; set acknowledge fail LED if the
        ; device did not pull data low
        ;
        retlw  0
;
;*****
;       Receive data Routine
;       This routine reads one byte of data from the part
;       into the 'data1' register. It then sends a high
;       ack bit to indicate that no more data is to be read
;*****
RX
        movlw  .8          ; set # bits to 8
        movwf  count
RXLP
        rlf   data1       ; rotate data1 1 bit left
        call  BITIN       ; read a bit
        btfsz eeprom,di
```

Interfacing the 24LCXXB Serial EEPROMs

```
        bsf      data1,0    ; set bit 0 if necessary
        decfsz   count     ; 8 bits done?
        goto    RXLP      ; no, do another
        retlw   0

;
;*****
;      Power up routine
;      This is the program entry point, which in this case simply
;      sets the port_a I/O lines and directs control to the
;      Read routine.
;*****
PWRUP
        movlw   b'00000000'
        tris   port_a    ; set port A as all output
        clrf   port_a    ; all output lines low
        goto   READ

;
;*****
;      READ (sequential read routine)
;
;      This routine reads 8 consecutive addresses of the
;      serial EE device starting at address 00 in the
;      sequential read mode. Reading in this mode is more
;      efficient than the random read mode as the control byte
;      and address have to be sent only once at the beginning
;      of the sequence. As many consecutive addresses as
;      needed can then be read from the part until a stop bit is
;      sent. In the read mode, the PIC16CXX must send the acknowledge
;      bit after every 8 data bits from the device. When the
;      last byte needed has been read, then the controller will
;      send a high acknowledge bit and then a stop bit to halt
;      transmission from the device.
;*****
READ
        ;
        bcf    port_a,ackf ; clear the ack fail LED if on
        movlw  .8          ; set number of bytes to read as 8
        movwf bcount      ; set slave address and write mode
        movlw  b'10100000' ; set slave address and write mode
        movwf slave
        clrf  addr        ; set starting address to 00
        ;
        call  BSTART      ; generate start bit
        movf  slave,w     ; get slave address
        movwf txbuf      ; into transmit buffer
        call  TX          ; and send it
        movf  addr,w     ; get word address
        movwf txbuf      ; into transmit buffer
        call  TX          ; and send it
        call  BSTART      ; generate start bit
        movlw b'10100001' ; get slave address and readmode
        movwf txbuf      ; into transmit buffer
        call  TX          ; and transmit it
        ;
rbyte  call  RX          ; read 1 byte from device
        decfsz bcount    ; are all 8 bytes read?
        goto  lowack     ; no, send low ack and do another
        bsf  eeprom,do   ; yes, send high ack bit
        call BITOUT      ; to stop transmission
        call BSTOP       ; and send a stop bit
        goto READ       ; start all over

lowack bcf  eeprom,do    ; send low ack bit
        call BITOUT      ; to continue transmission
        goto rbyte      ; and read another byte

;
END
```

Interfacing the 24LCXXB Serial EEPROMs

Using the 24C65 and 24C32 with Stand Alone PIC16C54 Code

USING THE SMART SERIAL™ SERIES

With the advent of CMOS silicon devices came the battery powered application. The battery powered application required more functionality and thus more power from the microcontroller which required more power from the batteries which cost more. Just as nature has a definite cycle, so it seems, does the portable application. Nowhere has this been more evident than in the areas of personal communication and data acquisition. Never before in the history of the industrial revolution have the type of applications seen emerging today been possible. Every man-made object that requires human or machine interface, has the potential to be controlled by embedded circuits and to be powered by the advanced technology batteries available today. Along with this increased functionality comes the requirement for MORE MEMORY. More in a smaller package, at a better price requiring less power. The majority of the hand-held embedded applications either use what little non-volatile memory that is available on the controllers or they use external devices. Parallel EEPROMs require too many I/O connects, which is the major source of device active current. Serial EEPROMs are typically the answer for these applications and the industry standard I²C™ is the industry leader (>70 % worldwide). The I²C standard specifies a maximum 16k bit address space. How does the personal communications system designer, who requires more memory, solve this problem?

THE I²C DILEMMA

The I²C serial bus has many advantages over other common serial interfaces for serial embedded devices. The I²C bus with level-triggered inputs offers better noise immunity over edge triggered technology. Op codes are not needed to communicate with storage devices because all interfaces are intuitive and comparable to parallel devices. The I²C protocol assigns a slave address for each unique device or device family. Microchip Technology, and most non-volatile suppliers, use the slave address of 1010 for serial electrical erasable programmable read only memory (Serial E²PROM). The protocol also facilitates up to a maximum of 16K bytes of memory on the bus via the 8-bit address and the three device or memory block select pins A0, A1, and A2. Here lies the dilemma: with the advent of the more sophisticated personal communication devices such as cellular and full featured phones, personal digital assistants and palm-top computers, 16K bytes is not enough!

Smart Serial Products

The Smart Serial concept grew from the industry need for increased memory requirements in I²C embedded applications, smarter endurance performance, security needs, and the need for more functionality at lower power demands. Currently, Microchip Technology Inc. has the 24C65 and 24C32 devices available. All comments in this application note pertain primarily to the 24C65 but the routines and general architectural comments apply to the 24C32 as well.

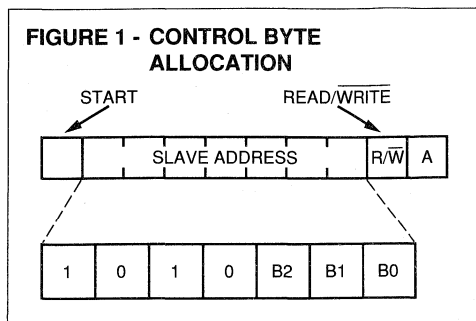
The user should reference the individual data sheets for specific differences.

The Microchip Technology Inc. 24C65 is a serial memory device with 64K bits (8Kx8) capacity and additional patent pending unique features not found anywhere else. First let's look at the current I²C addressing scheme, the cascadable solution, and finally the Microchip total embedded systems solution.

I²C ADDRESSING

The I²C protocol utilizes a master/slave bi-directional communication bus. The master, usually a microcontroller, which controls the bus, generates the serial clock (SCL) originates the start and stop conditions. A Serial EEPROM is considered a slave device and is defined as a transmitter during read operations and generates acknowledges when receiving data from the master. The start and stop bits are utilized to control the bus. Normal operation begins with a start bit and ends with a stop bit. Following a start, commands begin with an 8 bit 'control' byte originated by the master. The control byte identifies the slave device to be addressed and defines the operation to take place. A typical control byte for a Serial EEPROM (slave address = 1010) is shown in figure 1. The control byte, therefore, consists of a start bit, a four-bit slave address, a read/write bit and an acknowledge. The slave address consists of the 1010 identifying address plus the three block or chip select bits.

Using the 24C65 and 24C32



All memory and peripheral devices on the I²C bus conform to this sequence for identification and selection. There are 128 assigned slave addresses in the standard protocol. There is a 10-bit extension to the protocol for 1024 additional slave addresses.

THE CASCADABLE SOLUTION

Cascading is an addressing scheme used in the 24164 (2Kx8 EEPROM) to enable using more than the 16K limit set forth by the standard. In this method the A0, A1, and A2 pins are mapped into bits 2, 3, and 4 of the 8-bit slave address. This approach allows the system designer to use the non-standard 24164 to increase the total memory of a given memory subsystem. This solution is definitely workable, but does not offer the user the system design flexibility needed for the newer and more complex systems. Most system architects would prefer to have a linear address space for program and data memory. Programmers also find the linear solution more attractive. The overhead required for bank switching and chip selection usually requires additional overhead and hardware.

THE ULTIMATE SOLUTION BY MICROCHIP

Microchip Technology Inc. has designed an addressing scheme based on the standard I²C protocol and device addresses but incorporating an additional address byte for enabling the designer to use up to 256K bits per device and add from 1 to 8 devices on the system bus. This flexibility allows for future memory expansion and more advanced features in a smaller, more cost effective, design. This enhanced addressing combined with the many advanced and patent pending features of the 24C65 make the 24C65 an exciting and innovative device. It is the first in a family of sophisticated **Smart Serials™** EEPROMs from Microchip Technology Inc.

24C65 Features

The 24C65 has an advanced architecture with the following features:

- 15-bit, 2-byte address field (14 bit for the 24C32)
- 4K-bit High Endurance Block - 1 Million E/W cycles typical (Fixed at the last 4k bit block in the array)
- Programmable write protect security features with up to a 15 blocks of 4K bits
- 8 byte by 8 line input write cache for I²C Fast Mode, burst mode capability, and use as a capture buffer

24C65 Addressing

For the first byte or control byte, the 24C65 adheres to the I²C protocol (reference figure 2). This is the first byte received following the start condition from the master device. The control byte consists of a four-bit control code for the 24C65 (this is assigned as 1010 binary for read and write operations). The next three bits of the control byte are the device select bits (A2, A1, A0). They are used by the master device to select which of the eight devices are to be accessed. These bits are in effect the three most significant bits of the word address. The last bit of the control byte defines the operation to be performed. When set to a one a read operation is selected, when set to a zero a write operation is selected. The least significant 13 bits of the next 2 bytes define the address of the first byte within the 8K block. The most significant byte is transferred first. Following the start condition, the 24C65 monitors the SDA bus, checking the device type identifier being transmitted, upon receiving a 1010 code and appropriate device select bits, the slave device outputs an acknowledge signal on the SDA line. Depending on the state of the R/W bit, the 24C65 will select a read or write operation.

The addressing scheme uses the standard first byte slave address format of the I²C standard with the Microchip Technology-assigned 1010 slave address. The internal bus controller scans the next byte for bit 7 to be asserted indicating that a security operation is to take place. This will be further explained later. The remainder of the byte is composed of the most significant address bits. The next byte is the least significant address byte. See figure 2 for graphical representation of this sequence. After receiving a valid 2 byte address and a stop bit, the 24C65 will process this address according to bit zero of the control byte and either wait for data to be written, if in a write sequence, or present the requested data if in a read sequence.

ADVANCED FEATURES

Programmable security

The 24C65 has a sophisticated security mechanism by which selected blocks of memory may be write protected by the user. The write sequence includes a bit for enabling the security protection scheme, bit 7 of the first address byte. When this bit is set to a one, the first byte

following the address during a write sequence defines the security block. This includes a pointer to the starting 4K block to be protected (the write address), a write/erase flag, and a non zero four bit code for determining the number of 4K blocks to protect up to the maximum of 15 (60K). The 4K blocks must be contiguous. In a normal application the security block or blocks would be set after all code or look-up table data has been finalized. THIS OPERATION CAN ONLY BE PERFORMED ONCE. (See Figures 2 and 3.)

Total Endurance™

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate, for each byte in the application, not on the number of op codes or control byte commands, and is never based on any read functions whether they be a data read or configuration read. If a

part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. Applying the predictive data, from the 24LC04 which has similar characteristics, to the 4K high endurance block and assuming the following:

- a five-year life for a personal communication device
- an expected E/W cycles of 10 times per day
- a last number redial function of 11 bytes

Operational specifications:

Device	24C65
Voltage	5
Temperature	25 C
Bytes/Cycle	11
E/W/Day	10
App. Life (Yr.)	5
Cycling Mode	BYTE
Data Pattern	RANDOM

The 4k HE block with 1 M E/W cycles typical, in this application, should yield the following results:

FIT	1.0
PPM	6
Time	5.0
Write cycles	18,250

FIGURE 2 - CONTROL BYTE ALLOCATION

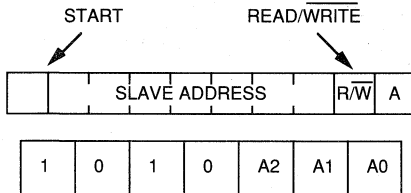
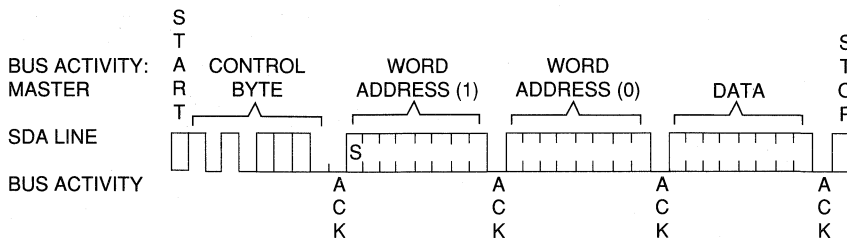
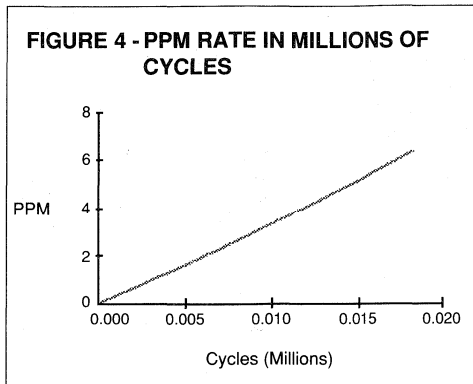


FIGURE3 - BYTE WRITE



Using the 24C65 and 24C32



The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance see the related application notes AN537 and AN562 contained in this volume. (See Figure 4.)

64 Byte Write Cache

The cache is arranged in 8-byte pages by 8 lines each. This yields a total of 64 bytes. The interface to the 24C65 supports both the standard 100 kHz mode and FAST mode at 400 kHz. The input cache can therefore support a burst write option of up to 64 bytes. When using the I²C protocol, an end of a page is defined by the transmission of a stop bit by the master. This sequence in the 24C65 could be used to define pages from 8 to 64 bytes in length. The 8 byte by 8 line cache will roll over if a write is attempted past byte 8 of line 8, thus it can be used as

a 64 byte capture or snapshot buffer. Each line is overwritten during a subsequent write to the same line. Faster memory and controller interfaces will become increasingly important in applications incorporating the ACCESS.bus interface standard.

Power Management

Increasingly, power management is becoming a predominant requirement for hand-held devices where a limited amount of power is available from the total power budget for a particular function. The 24C65 has built-in power saving features such that the entire device is put into standby mode upon receiving a stop bit or an abort when in a read sequence, and after the completion of writing the data in the cache lines to the array when in an erase/write sequence. When the device is in standby mode, the only active circuit is the input circuit for the I²C clock. This yields a standby current that is the current consumption of this lone input and the normal leakage current of the silicon and typically will be less than 2 μ A.

The 24C65 is the first device available in the Smart Serial product line. The features and capabilities initiated with this device will become standard on all future Microchip Technology Inc. serials and some features will be enhanced, such as the security options.

Appendix A of this application note contains the required PIC16C54 assembly routines for setting the security function, addressing the Smart Serial devices and using the most common addressing, read, write, and data polling functions for the I²C bus.

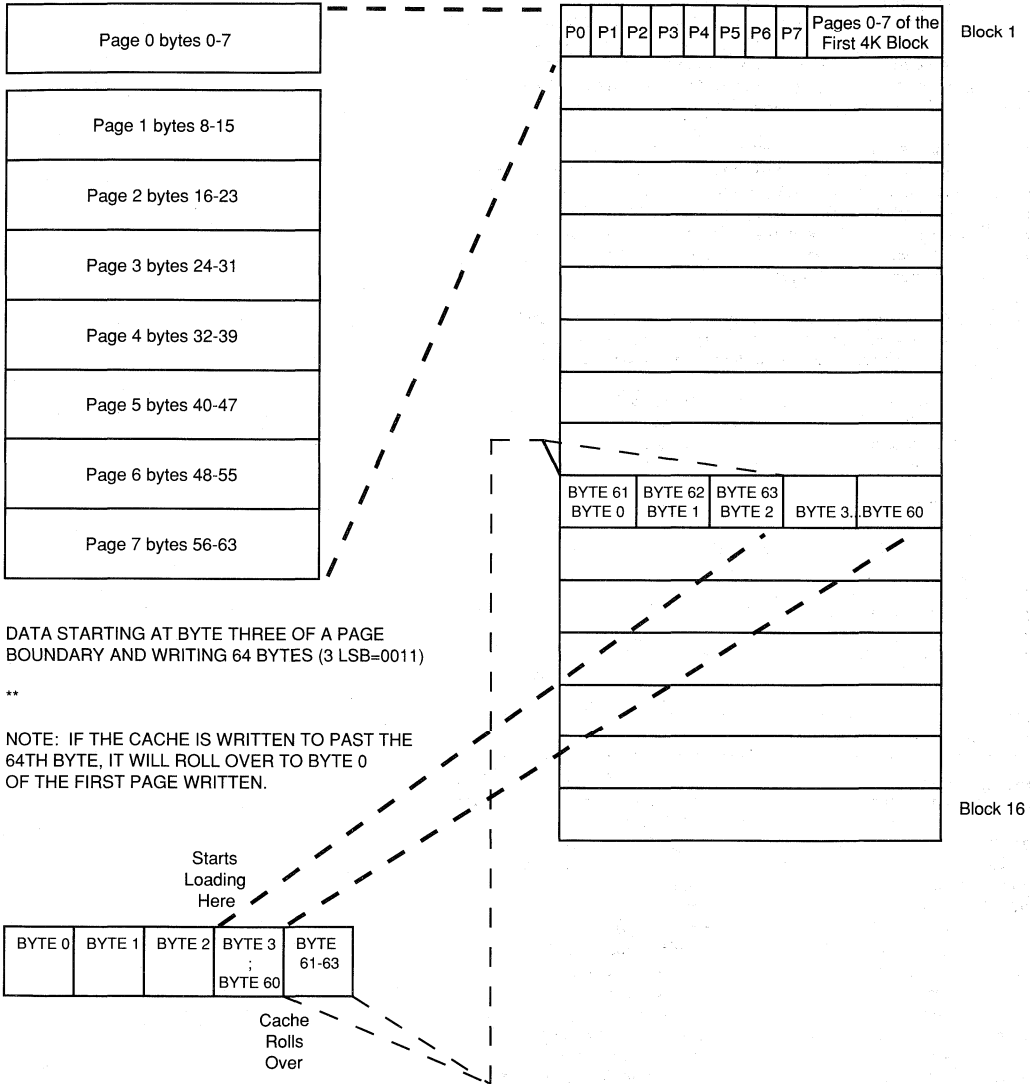
All of the list programs are complete stand-alone programs.

*AUTHORS: Richard J. Fisher
Memory Products Division
Bruce Negley
Memory Products Division*

FIGURE 5 - CACHE WRITE

DATA STARTING AT BYTE ZERO ON A PAGE BOUNDARY AND WRITING 64 BYTES

Page Cache 8 Byte x 8



Using the 24C65 and 24C32

APPENDIX A

```
LIST P=16C54
;*****
;      64K Byte Read Program (138 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24C65 Serial E2 device
;      and perform a random read operation. This program
;      will read 8 consecutive addresses in the 'random
;      read' mode. This entails setting the address pointer
;      before doing the read command for each address.
;
;      Another, more efficient method of reading consecutive
;      addresses is the 'sequential read' mode. This involves
;      sending the control byte and address for the first byte
;      to read, then continuing to provide clocks for the next
;      addresses. The device will automatically increment the
;      address. An example of the sequential read mode is
;      provided in the '64kseqr.asm' file.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 60 Khz for this setup.
;
;      As an option, the user may connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as an
;      acknowledge fail indicator. This LED will come on if
;      the serial EE fails to acknowledge correctly.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      -----
;      Pin 12 (RB6)  ->  SCLK
;      Pin 13 (RB7)  ->  SDATA
;
;      PIN 18 (RA1)  ->  Acknowledge fail LED (Optional)
;
;*****
;      Register Definitions
;*****
port_a equ 5h          ; port 5 (port a) used for LED display
port_b equ 6h          ; port 6 (port b) used for data and
                       ; clock lines
eeprom equ 0ah         ; bit buffer
bycnt  equ 0bh         ; byte counter for read mode
addr   equ 0ch         ; word 0 address counter
datai  equ 0dh         ; data input register
datao  equ 0eh         ; data output register
slave  equ 0fh         ; device address 1010xxx0)
txbuf  equ 10h         ; transmit buffer
count  equ 11h         ; bit counter
bcount equ 12h         ; byte counter
loops  equ 15h         ; delay loop counter
loops2 equ 16h         ; delay loop counter
addr1  equ 17h         ; word 1 address counter
;*****
;      Bit Definitions
;*****
d      equ 7           ; eeprom input bit
do     equ 6           ; eeprom output bit
sdata  equ 7           ; serial EE data line (port_b, pin 13)
sclk   equ 6           ; serial EE clock line (port_b, pin 12)
ackf   equ 1           ; acknowledge fail LED (port_a)
;*****
;
```

Using the 24C65 and 24C32

```
org    01ffh        ; set reset vector
goto  PWRUP
org    000h         ;
goto  PWRUP
;
;*****
;   DELAY ROUTINE
;   This routine takes the value in 'loops'
;   and multiplies it times 1 millisecond to
;   determine delay time.
;*****
WAIT
;
top    movlw    .110        ; timing adjustment variable
      movwf    loops2
top2   nop                ; sit and wait
      nop
      nop
      nop
      nop
      decfsz   loops2      ; inner loops complete?
      goto    top2        ; no, go again
      ;
      decfsz   loops      ; outer loops complete?
      goto    top         ; no, go again
      retlw   0           ; yes, return from sub
;
;*****
;   Start Bit Subroutine
;   this routine generates a start bit
;   (Low going data line while clock is high)
;*****
BSTART
      bsf     port_b,sdata  ; make sure data is high
      movlw  b'00111111'
      tris   port_b        ; set data and clock lines for output
      bcf   port_b,sclk    ; make sure clock is low
      nop
      bsf   port_b,sclk    ; set clock high
      nop
      nop
      nop
      nop
      bcf   port_b,sdata   ; data line goes low during
                          ; high clock for start bit
      nop
      nop
      nop
      nop
      nop
      bcf   port_b,sclk    ; timing adjustment
                          ; start clock train
      nop
      nop
      retlw  0
;
;*****
;   Stop Bit Subroutine
;   This routine generates a stop bit
;   (High going data line while clock is high)
;*****
BSTOP
      bcf   port_b,sdata   ; make sure data line is low
      movlw b'00111111'   ;
      tris  port_b        ; set data/clock lines as outputs
      bcf  port_b,sdata   ; make sure data line is low
      nop
      nop
```

Using the 24C65 and 24C32

```

    nop
    bsf    port_b,sclk           ; set clock high
    nop
    nop
    nop
    bsf    port_b,sdata         ; data goes high while clock high
                                ; for stopbit
    nop
    nop
    bcf    port_b,sclk           ; set clock low again
    nop
    nop
    nop
    retlw  0
;
;*****
; BITOUT routine takes the bit of data in 'do' and
; transmits it to the serial EE device
;*****
BITOUT
    movlw  b'00111111'         ; set data, clock as outputs
    tris   port_b
    btfss  eeprom,do           ; check for state of data bit to xmit
    goto   bitlow              ; low? go set data line low
    bsf    port_b,sdata         ; high? set data line high
    goto   clkout              ; go toggle the clock

bitlow  bcf    port_b,sdata     ; output a low bit
clkout  bsf    port_b,sclk      ; set clock line high
    nop
    nop
    nop
    bcf    port_b,sclk          ; return clock line low
    retlw  0
;
;*****
; BITIN routine reads one bit of data from the
; serial EE device and stores it in the bit 'di'
;*****
BITIN
    bsf    eeprom,di           ; assume input bit is high
    movlw  b'10111111'         ; make sdata an input line
    tris   port_b
    bsf    port_b,sdata         ; set sdata line for input
    bsf    port_b,sclk          ; set clock line high
    nop                                         ; just sit here a sec
    nop
    nop
    nop
    btfss  port_b,sdata         ; read the data bit
    bcf    eeprom,di           ; input bit was low, set 'di' accordingly
    bcf    port_b,sclk          ; set clock line low
    ;
    retlw  0
;
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw  .8
    movwf  count               ; set the #bits to 8
```

```

;
TXLP
    bcf     eeprom,do      ; assume bit out is low
    btfsc  txbuf,7        ; is bit out really low?
    bsf    eeprom,do      ; no, set it high
    call   BITOUT         ; send the bit to serial EE
    rlf    txbuf          ; rotate txbuf left
    decfsz count          ; 8 bits done?
    goto  TXLP           ; no - go again
    call   BITIN          ; read ack bit
    btfsc  eeprom,di      ; check ack bit
    bsf    port_a,ackf    ; set acknowledge fail LED if the
                        ; device did not pull data low

;
    retlw  0
;

;*****
;
;   Receive data routine
;   This routine reads one byte of data from the part
;   into the 'data1' register. It then sends a high
;   ack bit to indicate that no more data is to be read
;*****
RX
    cdf    data1          ; clear input buffer
    movlw  .8             ; set # bits to 8
    movwf  count
RXLP
    rlf    data1          ; rotate data1 1 bit left
    call   BITIN          ; read a bit
    btfsc  eeprom,di      ; set bit 0 if necessary
    bsf    data1,0        ; set bit 0 if necessary
    decfsz count          ; 8 bits done?
    goto  RXLP           ; no, do another
    bsf    eeprom,do      ; set ack bit = 1
    call   BITOUT         ; to finish transmission
    retlw  0
;

;*****
;
;   Power up routine
;   This is the program entry point. I/O line status is
;   set for port A here.
;*****
PWRUP
    movlw  b'00000000'
    tris  port_a          ; set port A as all output
    cdf   port_a          ; all output lines low
    goto  READ
;

;*****
;
;   READ (read routine)
;   This routine reads 8 consecutive addresses of the
;   serial EE device starting at address 00 in the
;   random access mode (non-sequential read). Reading
;   the device using the random access mode
;   requires that the address pointer be set for every
;   byte before the read takes place. The address pointer
;   is set by sending a 'write mode' control byte with the
;   address of the location to read.
;*****
READ
;
    bcf    port_a,ackf    ; clear the ack fail LED if on
    movlw  .8             ; set number of bytes to read as 8
    movwf  bcount        ;
    cdf    addr1          ; set starting high address byte to 00
    cdf    addr          ; set starting low address byte to 00
;
;
rbyte    call   BSTART    ; generate start bit

```

Using the 24C65 and 24C32

```

;
; now send the write control byte and
; address to set the pointer
;
movlw b'10100000' ; get slave address (write mode)
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr1,w ; get word 1 address
movwf txbuf ; into transmit buffer
call TX ; and send it
movf addr,w ; get word 0 address
movwf txbuf ; into transmit buffer
call TX ; and send it
;
; now read one byte from the part
;
call BSTART ; generate start bit
movlw b'10100001' ; get slave address and read mode
movwf txbuf ; into transmit buffer
call TX ; and transmit it
call RX ; read 1 byte from serial EE
call BSTOP ; send stop bit to end transmission
incf addr ; add 1 to address counter
decfsz bcount ; yes, are all 8 bytes read?
goto rbyte ; no, do another byte

movlw .255 ; long delay for scope
movwf loops ; trigger purposes only
call wait
goto READ ; yes, start all over
;
END
```

Using the 24C65 and 24C32

```
LIST P=16C54
;*****
; 64K Byte Write Program (127 bytes)
;
; This program demonstrates how to interface a
; Microchip PIC16C54 to the 24LC65 Serial E2 device
; and perform a byte write operation on 8 consecutive
; addresses.
;
; After each byte is written, time must be given to the
; device for it to complete the write cycle before
; the next command can be sent. The easiest solution
; is to consult the data book for the maximum write
; cycle time and just wait that long before the next
; command is sent (10ms for this device). This program
; demonstrates that solution.
;
; Another, more efficient method of determining when the
; write cycle is complete is called 'data polling.' This
; method is demonstrated in the program "64kdpoll."
;
; As an option, the user may connect a LED to pin 18
; on the PIC16CXX (use about a 1K resistor in series) as an
; acknowledge fail indicator. This LED will come on if
; the serial EE fails to acknowledge correctly.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4Mhz crystal. Clock speeds to the serial EE
; will be approximately 60 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX      Serial EE
; -----
; Pin 12 (RB6)  -> SCLK
; Pin 13 (RB7)  -> SDATA
;
; PIN 18 (RA1)  -> Acknowledge fail LED (Optional)
;
;*****
; RegisterDefinitions
;*****
port_a equ 5h ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port_b) used for data and
; clock lines
eeprom equ 0ah ; bit buffer
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; word 0 address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter
;*****
; BitDefinitions
;*****
d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b, pin 13)
sclk equ 6 ; serial EE clock line (port_b, pin 12)
ackf equ 1 ; acknowledge fail LED (port_a, pin 18)
;*****
;
;
```

Using the 24C65 and 24C32

```
org    01ffh    ; set reset vector
goto   PWRUP
org    000h      ;
goto   PWRUP
;
;*****
;   DELAY ROUTINE
;   This routine takes the value in 'loops'
;   and multiplies it times 1 millisecond to
;   determine delay time.
;*****
WAIT
;
top    movlw    .110    ; timing adjustment variable
      movwf    loops2
top2   nop      ; sit and wait
      nop
      nop
      nop
      nop
      nop
      decfsz   loops2  ; inner loops complete?
      goto    top2    ; no, go again
      ;
      decfsz   loops   ; outer loops complete?
      goto    top     ; no, go again
      retlw   0       ; yes, return from sub
;
;*****
;   Start Bit Subroutine
;   this routine generates a start bit
;   (Low going data line while clock is high)
;*****
;
BSTART
      bsf     port_b,sdata    ; make sure data is high
      movlw  b'00111111'
      tris   port_b          ; set data and clock lines for output
      bcf   port_b,sclk     ; make sure clock is low
      nop
      bsf   port_b,sclk     ; set clock high
      nop
      nop
      nop
      nop
      nop
      bcf   port_b,sdata    ; data line goes low during
                          ; high clock for start bit
      nop
      nop
      nop
      nop
      nop
      bcf   port_b,sclk     ; timing adjustment
                          ; start clock train
      nop
      retlw  0
;
;
```


Using the 24C65 and 24C32

```
*****
;      Stop Bit Subroutine
;      This routine generates a stop bit
;      (High going data line while clock is high)
*****
BSTOP
    movlw    b'00111111'    ;
    tris    port_b          ; set data/clock lines as outputs
    bcf    port_b,sdata    ; make sure data line is low
    nop
    nop
    nop
    bsf    port_b,sclk     ; set clock high
    nop
    nop
    nop
    bsf    port_b,sdata    ; data goes high while clock high
                          ; for stopbit
    nop
    nop
    bcf    port_b,sclk     ; set clock low again
    nop
    nop
    nop
    retlw   0
;
*****
;      BITOUT routine takes one bit of data in 'do' and
;      transmits it to the serial EE device
*****
BITOUT
    movlw    b'00111111'    ; set data,clock as outputs
    tris    port_b
    btfs    eeprom,do      ; check for state of data bit to xmit
    goto    bitlow        ;
    bsf    port_b,sdata    ; set data line high
    goto    clkout        ; go toggle the clock

bitlow    bcf    port_b,sdata    ; output a low bit
clkout    bsf    port_b,sclk     ; set clock line high
    nop
    nop
    nop
    bcf    port_b,sclk     ; return clock line low
    retlw   0
;
*****
;      BITIN routine reads one bit of data from the
;      serial EE device and stores it in 'di'
*****
BITIN
    bsf    eeprom,di        ; assume input bit is high
    movlw  b'10111111'     ; make sdata an input line
    tris    port_b
    bsf    port_b,sdata    ; set sdata line for input
    bsf    port_b,sclk     ; set clock line high
    nop
    nop                    ; just sit here a sec
    nop
    nop
    nop
    btfs    port_b,sdata    ; read the data bit
    bcf    eeprom,di        ; input bit was low
    bcf    port_b,sclk     ; set clock line low
    ;
    retlw  0
;
;
```

Using the 24C65 and 24C32

```
*****
;
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
*****
TX
    movlw    .8
    movwf    count        ; set the #bits to 8
;
TXLP
    bcf      eeprom,do     ; assume bit out is low
    btfscc  txbuf,7       ; is bit out really low?
    bsf      eeprom,do     ; otherwise data bit =1
    call    BITOUT        ; serial data out
    rlf      txbuf        ; rotate txbuf left
    decfsz  count        ; 8 bits done?
    goto    TXLP        ; no - go again
    call    BITIN        ; read ack bit
    btfscc  eeprom,di     ; check ack bit
    bsf      port_a,ackf   ; set acknowledge fail LED if the
;
    retlw   0
;
*****
;
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directions control to the
; byte write routine.
*****
PWRUP
    movlw   b'00000000'
    tris   port_a        ; set port A as all output
    clrf   port_a        ; all output lines low
    goto   WRBYTE
;
*****
;
; Byte Write Routine
; This routine writes the data in "data0" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. This routine waits .0mS after every
; byte to give the device time to do the write. This
; program repeats forever.
*****
WRBYTE
;
    clrf   port_a        ; clear all LEDs
    movlw  b'10100000'   ; set slave address and write mode
    movwf  slave
    movlw  b'01010101'   ; set data to write as 55h
    movwf  data0
;
    movlw  .8            ; set number of bytes
    movwf  bcount       ; to write as to 8
    clrf   addr1        ; set high address byte to 00
    clrf   addr         ; set low address byte to 00
;
byte    call  BSTART     ; generate start bit
        movf  slave,w    ; move slave address
        movwf txbuf     ; into transmit buffer
        call  TX        ; and send it
        movf  addr1,w   ; move word 1 address
```

Using the 24C65 and 24C32

```
movwf txbuf          ; into transmit buffer
call TX              ; and send it
movf  addr,w         ; move word 0 address
movwf txbuf          ; into transmit buffer
call TX              ; and send it
movf  data0,w        ; move data byte
movwf txbuf          ; to transmit buffer
call TX              ; and transmit it
call  BSTOP          ; generate stop bit
                    ;
movlw .10            ; set delay time to give
movwf loops          ; 10 ms wait after every byte
call  WAIT           ; 10 ms wait after every byte
incf  addr            ; add 1 to low address counter
decfsz bcount        ; all 8 bytes written?
goto  byte           ; no, do another byte
                    ;
goto  wrbyte         ; start over
;
;
END
```

Using the 24C65 and 24C32

```
LIST P=16C54
;*****
;      64K Byte Write With Data Polling Program (133 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24LC65 Serial E2 device.
;      This program performs a byte write operation on 8
;      consecutive addresses using the data polling method
;      to determine when the write cycle is complete.
;
;      When writing to a serial E2 device, there are 2
;      ways to handle the internal timed write cycle
;      time of the device. The simplest method is
;      simply to wait until the maximum cycle time
;      is exceeded before attempting another command.
;      The other, more efficient method is known as "data
;      polling." Data polling is done by sending a start
;      bit and control byte to the part after the write
;      cycle has been initiated by a stop bit. If the ack bit
;      is low, then the device is through writing, otherwise
;      the the sequence is repeated. If no low acknowledge
;      is found within 40 attempts (about 10 milliseconds)
;      then the routine times out and sets the timeout
;      LED (pin 1) high.
;
;      As an option, the user can connect a LED to pin 1
;      on the PIC16CXX (use about a 1K resistor in series) as a
;      timeout indicator. This LED will come on if the
;      data polling is unsuccessful and the device being
;      programmed does not respond. This can be tested by
;      simply running the program with no part in the socket.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 40 Khz for this setup.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      - - - - -    - - - - -
;      Pin 12 (RB6) -> SCLK
;      Pin 13 (RB7) -> SDATA
;
;      Pin 1 (RA2) -> Write cycle timeout fail LED (optional)
;
;*****
;      Register Definitions
;*****
port_a  equ    5h      ; port 5 (port_a) used LEDs
port_b  equ    6h      ; port 6 (port_b) used for data and
                       ; clock lines
eeprom equ    0ah      ; bit buffer
bycnt  equ    0bh      ; byte counter for read mode
addr   equ    0ch      ; address counter
data1  equ    0dh      ; data input register
data0  equ    0eh      ; data output register
slave  equ    0fh      ; device address (1010xxx0)
txbuf  equ    10h      ; transmit buffer
count  equ    11h      ; bit counter
bcount equ    12h      ; byte counter
loops  equ    15h      ; delay loop counter
loops2 equ    16h      ; delay loop counter
pollcnt equ    17h      ; data polling counter
addr1  equ    18h      ; word 1 address counter
;*****
;      Bit Definitions
;*****
d      equ    7 ; eeprom input bit
do     equ    6 ; eeprom output bit
```

Using the 24C65 and 24C32

```
sdata equ 7 ; serial EE data line (port_b, pin 13)
sclk equ 6 ; serial EE clock line (port_b, pin 12)
timeout equ 2 ; write cycle timeout fail LED, port_a (pin 1)
ackf equ 1 ; acknowledge fail LED, port_a (pin 18)
;*****
;
org 01ffh ; set reset vector
goto PWRUP
org 000h ;
goto PWRUP
;
;*****
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;*****
WAIT
;
top movlw .110 ; timing adjustment variable
movwf loops2
top2 nop ; sit and wait
nop
nop
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again
;
decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub
;
;*****
; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;*****
BSTART
bsf port_b, sdata ; make sure data is high
movlw b'00111111'
tris port_b ; set data and clock lines for output
bcf port_b, sclk ; make sure clock is low
nop
bsf port_b, sclk ; set clock high
nop
nop
nop
nop
nop
nop
bcf port_b, sdata ; data line goes low during
; high clock for start bit
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
retlw 0
;
;*****
; Stop Bit Subroutine
; This routine generates a stop bit
; (High going data line while clock is high)
;*****
```

Using the 24C65 and 24C32

```
BSTOP
    movlw    b'00111111'    ;
    tris     port_b         ; set data/clock lines as outputs
    bcf     port_b,sdata    ; make sure data line is low
    nop
    nop
    nop
    bsf     port_b,sclk     ; set clock high
    nop
    nop
    nop
    bsf     port_b,sdata    ; data goes high while clock high
                                ; for stop bit
    nop
    nop
    bcf     port_b,sclk     ; set clock low again
    nop
    nop
    nop
    retlw   0
;
;*****
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;*****
BITOUT
    movlw    b'00111111'    ; set data, clock as outputs
    tris     port_b
    btfss   eeprom,do       ; check for state of data bit to xmit
    goto    bitlow          ;
    bsf     port_b,sdata    ; set data line high
    goto    clkout          ; go toggle the clock
                                ;
bitlow    bcf     port_b,sdata ; output a low bit
clkout    bsf     port_b,sclk ; set clock line high
    nop
    nop
    nop
    bcf     port_b,sclk     ; return clock line low
    retlw   0
;
; End of Subroutine
;*****
; BITIN routine reads one bit of data from the
; serial EE device and stores it in 'di'
;*****
BITIN
    bsf     eeprom,di       ; assume input bit is high
    movlw   b'10111111'    ; make sdata an input line
    tris     port_b
    bsf     port_b,sdata    ; set sdata line for input
    bsf     port_b,sclk     ; set clock line high
    nop
    nop                    ; just sit here a sec
    nop
    nop
    nop
    btfss   port_b,sdata    ; read the data bit
    bcf     eeprom,di       ; input bit was low
    bcf     port_b,sclk     ; set clock line low
                                ;
    retlw   0
;
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'datao' register and transmits it to the serial EE device.
```

Using the 24C65 and 24C32

```
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw    .8
    movwf   count        ; set the #bits to 8
;
TXLP
    bcf     eeprom,do    ; assume bit out is low
    btfscc txbuf,7      ; is bit out really low?
    bsf     eeprom,do    ; otherwise data bit =1
    call    BITOUT      ; serial data out
    rlf     txbuf        ; rotate txbuf left
    decfsz count        ; 8 bits done?
    goto   TXLP        ; no - go again
    call    BITIN       ; readack bit
;
    retlw   0
;
;*****
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; write routine.
;*****
PWRUP
    movlw   b'00000000'
    tris   port_a        ; set port A as all output
    clrf   port_a        ; all output lines low
    goto   WRBYTE

;*****
; Byte Write Routine with data polling technique
;
; This routine writes the data in "datao" to
; 8 consecutive bytes in the serial EE device starting
; at address 00. To determine when the write cycle time
; of the device, the 'data polling' method is used. This
; involves sending a start bit and control byte to the part
; and checking for an acknowledge. If the ack bit is low, then
; the device is through writing, otherwise the the sequence
; is repeated. If no low acknowledge is found within 40
; attempts (about 10 milliseconds) then the routine times
; out and sets the timeout LED (pin 18) high. This program
; will repeat forever.
;*****
;
WRBYTE
;
    clrf   port_a        ; all LEDs off
    movlw  b'10100000'   ; set slave address and write mode
    movwf  slave
    movlw  b'10101010'   ; set data to write as AAh
    movwf  datao
    movlw  .8            ; set number of bytes
    movwf  bcount        ; to write as to 8
    clrf   addr1         ; set starting high address to 00
    clrf   addr          ; set starting low address to 00
;
byte     call    BSTART   ; generate start bit
    movf   slave,w       ; move slave address
    movwf  txbuf         ; into transmit buffer
    call   TX            ; and send it
    movf   addr1,w      ; move word 1 address
```

Using the 24C65 and 24C32

```
movwf txbuf          ; into transmit buffer
call TX              ; and send it
movf addr,w          ; move word 0 address
movwf txbuf          ; into transmit buffer
call TX              ; and send it
movf data0,w         ; move data byte
movwf txbuf          ; to transmit buffer
call TX              ; and transmit it
call BSTOP           ; generate stop bit
;
; now start polling for a low ack bit
;
movlw .40             ; set max number of times to poll as 40
movwf pollcnt        ; generate start bit
poll call BSTART      ; move slave address (write mode)
movlw b'10100000'    ; into transmit buffer
movwf txbuf          ; and send it
call TX              ; was the ack bit low?
btfss eeprom,di      ; yes, do another byte
goto exitpoll        ; is poll counter down to zero?
decfsz pollcnt       ; no, poll again. Other wise the part is
goto poll            ; not responding in time so set timeout
bsf port_a,timeout   ; LED and continue on
;
; add 1 to address counter
exitpoll incf addr   ; all 8 bytes written?
decfsz bcount       ; no, do another byte
goto byte           ; yes, start over
goto WRBYTE
;
;
END
```



```

LIST P=16C54
;*****
;      64K Page Write Program (126 bytes)
;
;
;      The 24LC65 has a page length of 8 bytes. This page
;      can be used to write up to 8 bytes of data into the
;      part before initiating the write cycle. Since the
;      write cycle is timed the same for 1 byte or 8 bytes
;      it is more efficient to use the page mode when
;      consecutive addresses are being written to.
;
;
;      When using page mode, the control byte, upper and lower
;      addresses are sent for the first address only. After the
;      data byte for the first address is sent, the data for the
;      next consecutive address is clocked in. This is
;      repeated as many times as needed (as long as the page
;      length is not exceeded) and then a stop bit is sent.
;      The device will still acknowledge between every byte of
;      data. After the stop bit is sent, the part will
;      initiate the self timed write cycle.
;
;
;      This routine waits approximately 10mS for the write
;      cycle to complete for each page. A more efficient
;      method of determining when the write cycle is complete
;      is called "data polling." The data polling method is
;      explained in the program "64kdpoll.asm." That
;      particular program uses data polling in a byte write
;      mode but it can be used in exactly the same way for
;      a page mode write.
;
;
;      As an option, the user can connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as a
;      acknowledge fail indicator. This LED will come on
;      if the device being programmed does not send a low
;      acknowledge bit at the proper times. This can be
;      tested by simply running the program with no part
;      in the socket.
;
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 40 Khz for this setup.
;
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      - - - - -      - - - - -
;      Pin 12 (RB6) -> SCLK
;      Pin 13 (RB7) -> SDATA
;
;
;      PIN 18 (RA1) -> Acknowledge fail LED (Optional)
;
;
;*****
;      Register Definitions
;*****
port_a equ 5h      ; port 5 (port_a) used for LEDs
port_b equ 6h      ; port 6 (port_b) used for data and
                   ; clock lines
eprom  equ 0ah     ; bit buffer
bycnt  equ 0bh     ; byte counter for read mode
addr   equ 0ch     ; word 0 address counter
datai  equ 0dh     ; data input register
datao  equ 0eh     ; data output register
slave  equ 0fh     ; device address (1010xxx0)
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bit counter
bcoun  equ 12h     ; byte counter
loops  equ 15h     ; delay loop counter

```

Using the 24C65 and 24C32

```
loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter
;*****
; Bit Definitions
;*****
d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b, pin 13)
sclk equ 6 ; serial EE clock line (port_b, pin 12)
ackf equ 1 ; acknowledge fail LED (port_a, pin 18)
;*****
;
; org 01ffh ; set reset vector
; goto PWRUP
; org 000h ;
; goto PWRUP
;
;*****
; DELAY ROUTINE
; This routine takes the value in 'loops'
; and multiplies it times 1 millisecond to
; determine delay time.
;*****
WAIT
;
top movlw .110 ; timing adjustment variable
movwf loops2
top2 nop ; sit and wait
nop
nop
nop
nop
nop
decfsz loops2 ; inner loops complete?
goto top2 ; no, go again
;
decfsz loops ; outer loops complete?
goto top ; no, go again
retlw 0 ; yes, return from sub
;
;*****
; Start Bit Subroutine
; this routine generates a start bit
; (Low going data line while clock is high)
;*****
BSTART
bsf port_b, sdata ; make sure data is high
movlw b'00111111'
tris port_b ; set data and clock lines for output
bcf port_b, sclk ; make sure clock is low
nop
bsf port_b, sclk ; set clock high
nop
nop
nop
nop
nop
bcf port_b, sdata ; data line goes low during
; high clock for start bit
nop
nop
nop
nop
nop
nop
bcf port_b, sclk ; start clock train
nop
nop
retlw 0
```

Using the 24C65 and 24C32

```
;
;*****
;   Stop Bit Subroutine
;   This routine generates a stop bit
;   (High going data line while clock is high)
;*****
BSTOP
    movlw    b'00111111'    ;
    tris    port_b          ; set data/clock lines as outputs
    bcf     port_b,sdata    ; make sure data line is low
    nop
    nop
    nop
    bsf     port_b,sclk     ; set clock high
    nop
    nop
    nop
    bsf     port_b,sdata    ; data goes high while clock high
                                ; for stop bit
    nop
    nop
    bcf     port_b,sclk     ; set clock low again
    nop
    nop
    nop
    retlw   0
;
;*****
;   BITOUT routine takes one bit of data in 'do' and
;   transmits it to the serial EE device
;*****
BITOUT
    movlw    b'00111111'    ; set data,clock as outputs
    tris    port_b
    btfs    eeprom,do       ; check for state of data bit to xmit
    goto    bitlow         ;
    bsf     port_b,sdata    ; set data line high
    goto    clkout         ; go toggle the clock

bitlow    bcf     port_b,sdata    ; output a low bit
clkout    bsf     port_b,sclk     ; set clock line high
    nop
    nop
    nop
    bcf     port_b,sclk     ; return clock line low
    retlw   0
;
;*****
;   BITIN routine reads one bit of data from the
;   serial EE device and stores it in 'di'
;*****
BITIN
    bsf     eeprom,di       ; assume input bit is high
    movlw    b'10111111'    ; make sdata an input line
    tris    port_b
    bsf     port_b,sdata    ; set sdata line for input
    bsf     port_b,sclk     ; set clock line high
    nop
                                ; just sit here a sec
    nop
    nop
    nop
    nop
    btfs    port_b,sdata    ; read the data bit
    bcf     eeprom,di       ; input bit was low
    bcf     port_b,sclk     ; set clock line low
                                ;
    retlw   0
;
;
```

Using the 24C65 and 24C32

```

;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw    .8
    movwf   count        ; set the #bits to 8
;
TXLP
    bcf     eeprom,do    ; assume bit out is low
    btfsz  txbuf,7       ; is bit out really low?
    bsf     eeprom,do    ; otherwise data bit =1
    call   BITOUT        ; serial data out
    rlf    txbuf         ; rotate txbuf left
    decfsz count        ; 8 bits done?
    goto  TXLP          ; no - go again
    call   BITIN        ; read ack bit
    btfsz  eeprom,di    ; check ack bit
    bsf    port_a,ackf  ; set acknowledge fail LED if the
;
    retlw  0
;
;*****
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; byte write routine.
;*****
PWRUP
    movlw  b'00000000'
    tris  port_a        ; set port A as all output
    clrf  port_a        ; all output lines low
    goto  pwrite        ; go do the page write
;
;*****
; Page Write Routine
; This routine writes the data in "data0" to 8 consecutive
; bytes using page write mode starting at address 00.
; This routine waits 10mS after every page to give the device
; time to do the write. This program repeats forever.
;*****
;
pwrite
;
    clrf   port_a        ; clear all LEDs
    movlw  b'10100000'   ; set slave address and write mode
    movwf  slave
    movlw  b'01010101'   ; set data to write as 55h
    movwf  data0
;
    movlw  .8            ; set number of bytes
    movwf  bcount       ; to write as to 8
    clrf  addr1         ; set high address byte to 00
    clrf  addr          ; set low address byte to 00
;
    call  BSTART        ; generate start bit
    movf  slave,w       ; move slave address
    movwf txbuf         ; into transmit buffer
    call TX            ; and send it
    movf  addr1,w      ; move word 1 address
    movwf txbuf         ; into transmit buffer
    call TX            ; and send it
    movf  addr,w       ; move word 0 address
    movwf txbuf         ; into transmit buffer

```

Using the 24C65 and 24C32

```
        call    TX                ; and send it
byte   movf    datao,w           ; move data byte
        movwf  txbuf            ; to transmit buffer
        call   TX                ; and transmit it
        decfsz bcount           ; all 8 bytes written?
        goto   byte             ; no, do another byte
        ;
        call   BSTOP            ; generate stop bit
        movlw  .10              ; set delay time to give
        movwf  loops            ; 10 ms wait after every byte
        call   WAIT             ; 10 ms wait after every byte
        goto   pwrite           ; start over
;
;
END
```

Using the 24C65 and 24C32

```
LIST P=16C54
;*****
;      64K Sequential Read Program (142 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 24LC65 Serial EE device
;      and perform a sequential read operation. This program
;      will read 8 consecutive addresses in the 'sequential
;      read' mode. This entails setting the address pointer
;      for the first address only and then clocking out as many
;      bytes of data as needed. The device will automatically
;      increment the address.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 60 Khz for this setup.
;
;      As an option, the user may connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as an
;      acknowledge fail indicator. This LED will come on if
;      the serial EE fails to acknowledge correctly.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      - - - - -      - - - - -
;      Pin 12 (RB6)  ->  SCLK
;      Pin 13 (RB7)  ->  SDATA
;
;      PIN 18 (RA1)  ->  Acknowledge fail LED (Optional)
;
;*****
;      RegisterDefinitions
;*****
port_a equ 5h      ; port 5 (port a) used for LED display
port_b equ 6h      ; port 6 (port b) used for data and
                  ; clock lines
eeprom equ 0ah     ; bit buffer
bycnt  equ 0bh     ; byte counter for read mode
addr   equ 0ch     ; word 0 address counter
data1  equ 0dh     ; data input register
data0  equ 0eh     ; data output register
slave  equ 0fh     ; device address 1010xxx0)
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bit counter
bcount equ 12h     ; byte counter
loops  equ 15h     ; delay loop counter
loops2 equ 16h     ; delay loop counter
addr1  equ 17h     ; word 1 address counter
;*****
;      BitDefinitions
;*****
d      equ 7       ; eeprom input bit
do     equ 6       ; eeprom output bit
sdata  equ 7       ; serial EE data line (port_b, pin 13)
sclk   equ 6       ; serial EE clock line (port_b, pin 12)
ackf   equ 1       ; acknowledge fail LED (port_a)
;*****
;
;      org      01ffh ; set reset vector
;      goto    PWRUP
;      org      000h ;
;      goto    PWRUP
;
;*****
;      DELAY ROUTINE
;      This routine takes the value in 'loops'
;      and multiplies it times 1 millisecond to
;      determine delay time.
```

Using the 24C65 and 24C32

```
*****
;*****
WAIT
;
top    movlw  .110          ; timing adjustment variable
      movwf  loops2
top2   nop                 ; sit and wait
      nop
      nop
      nop
      nop
      decfsz loops2        ; inner loops complete?
      goto  top2          ; no, go again
      ;
      decfsz loops        ; outer loops complete?
      goto  top           ; no, go again
      retlw  0            ; yes, return from sub
;

;*****
;      Start Bit Subroutine
;      this routine generates a start bit
;      (Low going data line while clock is high)
;*****
;*****
BSTART
      bcf   port_b,sdata   ; make sure data is high
      movlw b'00111111'
      tris  port_b        ; set data and clock lines for output
      bcf   port_b,sclk   ; make sure clock is low
      nop
      bsf   port_b,sclk   ; set clock high
      nop
      nop
      nop
      nop
      bcf   port_b,sdata   ; data line goes low during
                          ; high clock for start bit
      nop
      nop
      nop
      nop
      bcf   port_b,sclk   ; timing adjustment
      nop
      bcf   port_b,sclk   ; start clock train
      nop
      retlw 0
;

;*****
;      Stop Bit Subroutine
;      This routine generates a stop bit
;      (High going data line while clock is high)
;*****
;*****
BSTOP
      bcf   port_b,sdata   ; make sure data line is low
      movlw b'00111111'
      tris  port_b        ; set data/clock lines as outputs
      bcf   port_b,sdata   ; make sure data line is low
      nop
      nop
      nop
      bsf   port_b,sclk   ; set clock high
      nop
      nop
```

Using the 24C65 and 24C32

```

    nop
    bsf     port_b,sdata          ; data goes high while clock high
                                           ; for stopbit

    nop
    nop
    bcf     port_b,sclk          ; set clock low again
    nop
    nop
    nop
    retlw  0
;
;*****
; BITOUT routine takes the bit of data in 'do' and
; transmits it to the serial EE device
;*****
BITOUT
    movlw  b'00111111'          ; set data, clock as outputs
    tris   port_b
    btfs   eeprom,do            ; check for state of data bit to xmit
    goto   bitlow               ; low? go set data line low
    bsf    port_b,sdata         ; high? set data line high
    goto   clkout               ; go toggle the clock

bitlow  bcf     port_b,sdata     ; output a low bit
clkout  bsf     port_b,sclk     ; set clock line high
        nop
        nop
        nop
        bcf     port_b,sclk     ; return clock line low
        retlw  0
;
;*****
; BITIN routine reads one bit of data from the
; serial EE device and stores it in the bit 'di'
;*****
BITIN
    bsf    eeprom,di            ; assume input bit is high
    movlw  b'10111111'         ; make sdata an input line
    tris   port_b
    bsf    port_b,sdata         ; set sdata line for input
    bsf    port_b,sclk         ; set clock line high
    nop                                         ; just sit here a sec
    nop
    nop
    nop
    nop
    btfs   port_b,sdata         ; read the data bit
    bcf    eeprom,di            ; input bit was low, set 'di' accordingly
    bcf    port_b,sclk         ; set clock line low
    ;
    retlw  0
;
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw  .8
    movwf  count                ; set the #bits to 8
    ;
TXLP
    bcf    eeprom,do            ; assume bit out is low
```



```

    btfsc    txbuf,7      ; is bit out really low?
    bsf      eeprom,do   ; no, set it high
    call     BITOUT      ; send the bit to serial EE
    rlf      txbuf       ; rotate txbuf left
    decfsz   count       ; 8 bits done?
    goto     TXLP        ; no - go again
    call     BITIN       ; read ack bit
    btfsc    eeprom,di   ; check ack bit
    bsf      port_a,ackf ; set acknowledge fail LED if the
                        ; device did not pull data low
                        ;
    retlw    0

;
;*****
;   Receive data routine
;   This routine reads one byte of data from the part
;   into the 'data1' register. It then sends a high
;   ack bit to indicate that no more data is to be read
;*****
RX
    clrf     data1       ; clear input buffer
    movlw   .8          ; set # bits to 8
    movwf   count
RXLP    rlf      data1   ; rotate data1 1 bit left
        call     BITIN   ; read a bit
    btfsc   eeprom,di   ; set bit 0 if necessary
    bsf     data1,0
    decfsz  count       ; 8 bits done?
    goto    RXLP        ; no, do another
    retlw   0

;
;*****
;   Power up routine
;   This is the program entry point. I/O line status is
;   set for port A here.
;*****
PWRUP
    movlw   b'00000000'
    tris    port_a      ; set port A as all output
    clrf    port_a      ; all output lines low
    goto    READ

;
;*****
;   READ (read routine)
;   This routine reads 8 consecutive addresses of the
;   serial EE device starting at address 00 in the
;   sequential read mode. Reading in this mode is more
;   efficient than the random read mode as the control byte
;   and address have to be sent only once at the beginning
;   of the sequence. As many consecutive addresses as
;   needed can then be read from the part until a stop bit is
;   sent. In the read mode, the PIC16CXX must send the acknowledge
;   bit after every 8 data bits from the device. When the
;   last byte needed has been read, then the controller will
;   send a high acknowledge bit and then a stop bit to halt
;   transmission from the device.
;*****
READ
;
    bcf     port_a,ackf ; clear the ack fail LED if on
    movlw   .8          ; set number of bytes to read as 8
    movwf   bcount
    movlw   b'10100000' ; set slave address and write mode
    movwf   slave
    clrf    addr1       ; set starting high address to 00
    clrf    addr        ; set starting low address to 00
                        ;
    call    BSTART      ; generate start bit
    movf   slave,w      ; get slave address
    movwf  txbuf        ; into transmit buffer

```

Using the 24C65 and 24C32

```
    call    TX                ; and send it
    movwf  addr1,w           ; get word 1 address
    movwf  txbuf             ; into transmit buffer
    call   TX                ; and send it
    movf   addr,w            ; get word 0 address
    movwf  txbuf             ; into transmit buffer
    call   TX                ; and send it
    call   BSTART            ; generate start bit
    movlw  b'10100001'      ; get slave address and read mode
    movwf  txbuf             ; into transmit buffer
    call   TX                ; and transmit it
    ;
rbyte   call   RX            ; read 1 byte from device
        decfsz bcount       ; are all 8 bytes read?
        goto  lowack        ; no, send low ack and do another
        bsf   eeprom,do     ; yes, send high ack bit
        call  BITOUT        ; to stop transmission
        call  BSTOP         ; and send a stop bit

        movlw .255          ; long delay for scope
        movwf loops        ; trigger purposes only
        call  wait
        goto  READ         ; start all over

lowack  bcf   eeprom,do     ; send low ack bit
        call  BITOUT        ; to continue transmission
        goto  rbyte        ; and read another byte

;
END
```

```

LIST P=16C54
;*****
; Program to set the high endurance block on the 64K
; device (24LC65). (122 bytes)
;
; The 24LC65 is a 64K device, 4K of which is error
; corrected to increase the endurance of that portion
; of the array. The location of this high endurance
; 'block' can be set by the user. When the device
; comes from the factory, the high endurance block
; will be set as the uppermost block in the array.
;
; This program sets the high endurance block as the
; first block in the array (address range 00h to 1FFh) .
;
; The high endurance block is set by sending the following
; sequence to the device:
;
; SB 10100000 1XXAAAAX XXXXXXXX 00XX0000 STB
;
; Where SB=start bit
;       1=data high
;       0=data low
; X=don't care
; AAAA=4 bit high endurance block address
; STB=stop bit
;
; As an option, the user may connect a LED to pin 18
; on the PIC16CXX (use about a 1K resistor in series) as an
; acknowledge fail indicator. This LED will come on if
; the serial EE fails to acknowledge correctly.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4Mhz crystal. Clock speeds to the serial EE
; will be approximately 60 KHz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX      Serial EE
; -----
; Pin 12 (RB6) -> SCLK
; Pin 13 (RB7) -> SDATA
;
; PIN 18 (RA1) -> Acknowledge fail LED (Optional);
;*****
; RegisterDefinitions
;*****
port_a equ 5h ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port b) used for data and
; clock lines
eeprom equ 0ah ; bit buffer
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; word 0 address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter
he_blk equ 18h ; high endurance block address
;*****
; Bit Definitions
;*****
d equ 7 ; eeprom input bit
do equ 6 ; eeprom output bit
sdata equ 7 ; serial EE data line (port_b, pin 13)

```

Using the 24C65 and 24C32

```
sclk    equ    6 ; serial EE clock line (port_b, pin 12)
ackf    equ    1 ; acknowledge fail LED (port_a, pin 18)
;*****
;
;
;       org    01ffh ; set reset vector
;       goto   PWRUP
;       org    000h  ;
;       goto   PWRUP
;
;*****
;       DELAY  ROUTINE
;       This routine takes the value in 'loops'
;       and multiplies it times 1 millisecond to
;       determine delay time.
;*****
WAIT
;
top     movlw  .110 ; timing adjustment variable
        movwf  loops2
top2    nop      ; sit and wait
        nop
        nop
        nop
        nop
        nop
        decfsz loops2 ; inner loops complete?
        goto   top2  ; no, go again
        ;
        decfsz loops ; outer loops complete?
        goto   top  ; no, go again
        retlw  0    ; yes, return from sub
;
;*****
;       Start Bit Subroutine
;       this routine generates a start bit
;       (Low going data line while clock is high)
;*****
BSTART
        bsf    port_b, sdata ; make sure data is high
        movlw b'00111111'
        tris  port_b ; set data and clock lines for output
        bcf  port_b, sclk ; make sure clock is low
        nop
        bsf  port_b, sclk ; set clock high
        nop
        nop
        nop
        nop
        nop
        bcf  port_b, sdata ; data line goes low during
        ; high clock for start bit
        nop
        nop
        nop
        nop
        nop
        bcf  port_b, sclk ; timing adjustment
        ; start clock train
        nop
        nop
        retlw 0
;
;*****
;       Stop Bit Subroutine
;       This routine generates a stop bit
;       (High going data line while clock is high)
;*****
BSTOP
        movlw b'00111111' ;
        tris  port_b ; set data/clock lines as outputs
```

```

    bcf    port_b,sdata    ; make sure data line is low
    nop
    nop
    nop
    bsf    port_b,sclk     ; set clock high
    nop
    nop
    nop
    bsf    port_b,sdata    ; data goes high while clock high
                                ; for stop bit

    nop
    nop
    bcf    port_b,sclk     ; set clock low again
    nop
    nop
    nop
    retlw  0
;
;*****
; BITOUT routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;*****
BITOUT
    movlw  b'00111111'     ; set data, clock as outputs
    tris   port_b
    btfss  eeprom,do       ; check for state of data bit to xmit
    goto   bitlow          ;
    bsf    port_b,sdata    ; set data line high
    goto   clkout          ; go toggle the clock

bitlow  bcf    port_b,sdata    ; output a low bit
clkout  bsf    port_b,sclk     ; set clock line high
    nop
    nop
    nop
    bcf    port_b,sclk     ; return clock line low
    retlw  0
;
;*****
; BITIN routine reads one bit of data from the
; serial EE device and stores it in 'di'
;*****
BITIN
    bsf    eeprom,di       ; assume input bit is high
    movlw  b'10111111'     ; make sdata an input line
    tris   port_b
    bsf    port_b,sdata    ; set sdata line for input
    bsf    port_b,sclk     ; set clock line high
    nop    ; just sit here a sec
    nop
    nop
    nop
    nop
    btfss  port_b,sdata    ; read the data bit
    bcf    eeprom,di       ; input bit was low
    bcf    port_b,sclk     ; set clock line low
    ;
    retlw  0
;
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; data0' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX

```

Using the 24C65 and 24C32

```
        movlw    .8
        movwf   count           ; set the #bits to 8
                                   ;
TXLP
        bcf     eeprom,do       ; assume bit out is low
        btfscc txbuf,7         ; is bit out really low?
        bsf     eeprom,do       ; otherwise data bit =1
        call    BITOUT         ; serial data out
        rlf     txbuf           ; rotate txbuf left
        decfsz  count           ; 8 bits done?
        goto    TXLP          ; no - go again
        call    BITIN          ; read ack bit
        btfscc eeprom,di       ; check ack bit
        bsf     port_a,ackf     ; set acknowledge fail LED if the
                                   ;
        retlw   0
;
;*****
;      Power up routine
;      This is the program entry point, which in this case simply
;      sets the port_a I/O lines and directs control to the
;      byte write routine.
;*****
PWRUP
        movlw   b'00000000'
        tris   port_a           ; set port A as all output
        crrf  port_a           ; all output lines low
        goto   setheblk        ; set go set the high endurance block
;*****
;      Set High Endurance Block Routine
;      This routine sets the high endurance block to the first
;      block in the array and then delays about a half second. This
;      routine will repeat forever.
;*****
setheblk
;
        crrf   port_a           ; clear all LEDs
        movwf  slave
        movlw  .0
        movwf  he_blk           ; set the endurance block as first
                                   ; block in array (block 0)
        rlf    he_blk           ; rotate starting block left 1 bit
                                   ; to arrange it correctly
        bsf    he_blk,7         ; set msb bit in block address byte
        call   BSTART          ; generate start bit
        movlw b'10100000'     ; get slave address and write mode
        movwf  txbuf           ; into transmit buffer
        call   TX               ; and send it
        movf   he_blk,w        ; get the block address byte
        movwf  txbuf           ; into transmit buffer
        call   TX               ; and send it
        crrf  txbuf           ; clear buffer
        call   TX               ; send 8 don't care bits
        crrf  txbuf           ; clear buffer
        call   TX               ; send config byte (all 0's)
        call   BSTOP          ; send stop bit

        movlw  .255            ; long delay
        movwf  loops
        call   wait
        movlw  .255
        movwf  loops
        call   wait

        goto   setheblk        ; go again
;
;
END
```

Using the 24C65 and 24C32

```
LIST P=16C54
;*****
; Program to set the security option (write protect)
; on the 64K device (24LC65). (125 bytes)
;
; The security option (or write protect option) allows
; the user to write protect any number of consecutive
; 4K blocks in the device.
;
; This program sets the security option to protect the
; lower 4 blocks (2K bytes) in the array.
;
; ***** CAUTIONS TO THE USER !!!! *****
;
; 1) THE SECURITY OPTION CAN ONLY BE SET ONCE!!
; 2) THE HIGH ENDURANCE BLOCK CANNOT BE CHANGED
; AFTER THE SECURITY OPTION IS SET.
; 3) THE HIGH ENDURANCE BLOCK CANNOT BE PROTECTED.
;
; The security option is set by sending the following
; sequence to the device:
;
; SB 10100000 1XXAAAAX XXXXXXXX 10XXNNNN STB
;
; Where SB=start bit
; 1=data high
; 0=data low
; X=don't care
; AAAA=4 bit number to indicate first secure block (0-15)
; NNNN=4 bit number to indicate how many secure blocks (1-15)
; STB=stop bit
;
; As an option, the user may connect a LED to pin 18
; on the PIC16CXX (use about a 1K resistor in series) as an
; acknowledge fail indicator. This LED will come on if
; the serial EE fails to acknowledge correctly.
;
; Timing is based on using the PIC16CXX in 'XT' mode
; using a 4Mhz crystal. Clock speeds to the serial EE
; will be approximately 60 Khz for this setup.
;
; PIC16CXX to Serial EE Connections:
;
; PIC16CXX      Serial EE
; -----
; Pin 12 (RB6) -> SCLK
; Pin 13 (RB7) -> SDATA
;
; PIN 18 (RA1) -> Acknowledge fail LED (Optional);
;*****
; Register Definitions
;*****
port_a equ 5h ; port 5 (port_a) used for LEDs
port_b equ 6h ; port 6 (port_b) used for data and
; clock lines
eeprom equ 0ah ; bit buffer
bycnt equ 0bh ; byte counter for read mode
addr equ 0ch ; word 0 address counter
datai equ 0dh ; data input register
datao equ 0eh ; data output register
slave equ 0fh ; device address (1010xxx0)
txbuf equ 10h ; transmit buffer
count equ 11h ; bit counter
bcount equ 12h ; byte counter
loops equ 15h ; delay loop counter
loops2 equ 16h ; delay loop counter
addr1 equ 17h ; word 1 address counter
strt_blk equ 18h ; starting block number for secure portion
sec_blks equ 19h ; number of secure blocks
```

Using the 24C65 and 24C32

```
*****
;
;   Bit Definitions
;*****
d      equ 7 ; eeprom input bit
do     equ 6 ; eeprom output bit
sdata  equ 7 ; serial EE data line (port_b, pin 13)
sclk   equ 6 ; serial EE clock line (port_b, pin 12)
ackf   equ 1 ; acknowledge fail LED (port_a, pin 18)
;*****
;
;       org 01ffh ; set reset vector
;       goto PWRUP
;       org 000h ;
;       goto PWRUP
;
;*****
;   DELAY ROUTINE
;   This routine takes the value in 'loops'
;   and multiplies it times 1 millisecond to
;   determine delay time.
;*****
WAIT
;
top     movlw .110 ; timing adjustment variable
        movwf loops2
top2    nop ; sit and wait
        nop
        nop
        nop
        nop
        nop
        decfsz loops2 ; inner loops complete?
        goto top2 ; no, go again
        ;
        decfsz loops ; outer loops complete?
        goto top ; no, go again
        retlw 0 ; yes, return from sub
;
;*****
;   Start Bit Subroutine
;   this routine generates a start bit
;   (Low going data line while clock is high)
;*****
BSTART
        bsf port_b, sdata ; make sure data is high
        movlw b'00111111'
        tris port_b ; set data and clock lines for output
        bcf port_b, sclk ; make sure clock is low
        nop
        bsf port_b, sclk ; set clock high
        nop
        nop
        nop
        nop
        nop
        bcf port_b, sdata ; data line goes low during
                        ; high clock for start bit
        nop
        nop
        nop
        nop
        nop
        bcf port_b, sclk ; timing adjustment
                        ; start clock train
        nop
        nop
        retlw 0
;
;*****
```


Using the 24C65 and 24C32

```
*****
;      Stop Bit Subroutine
;      This routine generates a stop bit
;      (High going data line while clock is high)
*****
BSTOP
    movlw    b'00111111'    ;
    tris    port_b          ; set data/clock lines as outputs
    bcf    port_b, sdata    ; make sure data line is low
    nop
    nop
    bsf    port_b, sclk     ; set clock high
    nop
    nop
    bsf    port_b, sdata    ; data goes high while clock high
                            ; for stop bit
    nop
    nop
    bcf    port_b, sclk     ; set clock low again
    nop
    nop
    nop
    retlw   0
;
*****
;      BITOUT routine takes one bit of data in 'do' and
;      transmits it to the serial EE device
*****
BITOUT
    movlw    b'00111111'    ; set data, clock as outputs
    tris    port_b          ;
    btfs    eeprom, do      ; check for state of data bit to xmit
    goto    bitlow         ;
    bsf    port_b, sdata    ; set data line high
    goto    clkout         ; go toggle the clock

bitlow    bcf    port_b, sdata    ; output a low bit
clkout    bsf    port_b, sclk     ; set clock line high
    nop
    nop
    nop
    bcf    port_b, sclk     ; return clock line low
    retlw   0
;
*****
;      BITIN routine reads one bit of data from the
;      serial EE device and stores it in 'di'
*****
BITIN
    bsf    eeprom, di       ; assume input bit is high
    movlw    b'10111111'    ; make sdata an input line
    tris    port_b          ;
    bsf    port_b, sdata    ; set sdata line for input
    bsf    port_b, sclk     ; set clock line high
    nop
                            ; just sit here a sec
    nop
    nop
    nop
    nop
    btfs    port_b, sdata    ; read the data bit
    bcf    eeprom, di       ; input bit was low
    bcf    port_b, sclk     ; set clock line low
    ;
    retlw   0
;
;
```

Using the 24C65 and 24C32

```
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'datao' register and transmits it to the serial EE device.
; It will then send 1 more clock to the serial EE for the
; acknowledge bit. If the ack bit from the part was low
; then the transmission was successful. If it is high, then
; the device did not send a proper ack bit and the ack
; fail LED will be turned on.
;*****
TX
    movlw    .8
    movwf   count           ; set the #bits to 8
;
TXLP
    bcf     eeprom, do      ; assume bit out is low
    btfsc  txbuf, 7        ; is bit out really low?
    bsf     eeprom, do      ; otherwise data bit =1
    call   BITOUT          ; serial data out
    rlf     txbuf          ; rotate txbuf left
    decfsz count          ; 8 bits done?
    goto   TXLP           ; no - go again
    call   BITIN          ; read ack bit
    btfsc  eeprom, di      ; check ack bit
    bsf     port_a, ackf    ; set acknowledge fail LED if the
;
    retlw   0
;
; *****
; Power up routine
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; byte write routine.
; *****
PWRUP
    movlw   b'00000000'
    tris   port_a          ; set port A as all output
    clrf   port_a          ; all output lines low
    goto   set_sec        ; set go set the high endurance block
;
; *****
; Set Security Routine
; This routine sets the secure portion of the array to the
; lower 4 blocks (2K bytes) in the array and then delays
; about a half second. This routine will repeat forever.
; *****
set_sec
;
    clrf   port_a          ; clear all LEDs
    movlw  .0
    movwf  strt_blk        ; set the first protected block as
; block 0
    movlw  .4
    movwf  sec_blks        ; set the number of protected blocks
; as 4 (2048 bytes)
    rlf    strt_blk        ; rotate starting block left 1 bit
; to arrange it correctly
    bsf    strt_blk, 7      ; set msb bit in starting block address
    bsf    sec_blks, 7     ; set msb bit block count byte
    call   BSTART          ; generate start bit
    movlw  b'10100000'    ; get slave address and write mode
    movwf  txbuf          ; into transmit buffer
    call   TX              ; and send it
;
    movf   strt_blk, w     ; get the starting block address
    movwf  txbuf          ; into transmit buffer
    call   TX              ; and send it
    clrf   txbuf          ; clear buffer
```

Using the 24C65 and 24C32

```
call    TX                ; send 8 don't care bits

movf    sec_blks,w        ; get secure blocks byte
movwf   txbuf             ; into transmit buffer
call    TX                ; and send it
call    BSTOP             ; send stop bit

movlw   .255              ; long delay
movwf   loops
call    wait
movlw   .255
movwf   loops
call    wait

goto    set_sec           ; go again
;
;
END
```

Using the 24C65 and 24C32

NOTES

24C01A Compatibility Issue and Its Mobility for Memory Upgrade

INTRODUCTION

The 24C01 is a 1K (128 x 8) serial EEPROM which is currently offered by Microchip and Xicor. There are several important differences between the two devices which are discussed in this report. This report refers to the Microchip part as the 24C01A and the Xicor part as the X24C01. It is intended to assist in designing a memory subsystem which is compatible with either device.

COMPATIBILITY ISSUES

There are three major differences between Microchip's 24C01A and Xicor's X24C01 as detailed below.

2.1 PAGE MODE DIFFERENCES

The 24C01A was originally designed to work in the same socket as the PCD8572 which has a two-byte page mode. Therefore, its page buffer is two bytes deep. The X24C01 has a page mode of four bytes depth.

If more than two bytes are transmitted to the 24C01A during a page programming cycle, the 24C01A will terminate the write cycle.

In many applications where serial EEPROMs are used and speed is not a key issue, the byte write mode can be used without any loss of system performance. If only the byte write mode is used, there is no compatibility problem (other than the slave address software differences discussed in 2.2).

If the page write feature must be used, two different page mode algorithms can be transmitted by the master depending upon whose device is being used. The master will have to first do a polling routine to determine if it is interfacing with a 24C01A or X24C01. This polling technique is discussed in 2.2.

Interestingly, the 24C01A actually updates faster in the page mode even though it has one-half the page depth of the X24C01. This is due to the faster write cycle time of the 24C01A. The two devices are compared in Figure 1.

2.2 SOFTWARE DIFFERENCES

Microchip's 24C01A is designed to share a 2-wire bus on which it resides with other devices. To support this, the first byte of each command sequence from the master to the 24C01A must be a slave address. The 24C01A monitors the 2-wire bus for its slave address and "wake-up" from standby mode if the address transmitted matches its address as defined by the voltage level (V_{ss} or V_{cc}) on pins 1, 2 and 3. X24C01 does not support a multiple device bus and will always "wake-up" if a start condition is detected.

A slave address must be transmitted to the 24C01A at certain points during reading and writing. This slave address is not required by the X24C01. Transmitting a slave address to X24C01 will result in erroneous operation. This problem can be solved by having the master transmit the proper serial bit pattern to the slave, but first the master has to ascertain with which 24C01 it is communicating.

The master can do a simple polling routine before beginning serial communication with 24C01A or X24C01 to determine with which device it is working. The proper serial protocol for both devices must be contained in the master controller's firmware. Once the master knows which 24C01 is on the bus, it can execute the proper serial commands.

FIGURE 1 - PAGE MODE DIFFERENCES

	<i>Microchip</i>	<i>Xicor</i>
Max byte program time	1 ms	10 ms
Max page program time	2 ms (2 bytes)	10 ms (4 bytes)
Max time to program 4 bytes	4 ms	10 ms
Max time to rewrite device	128 ms	320 ms

24C01A Compatibility Issue

The polling consists of the pattern like the one shown below:

SDA LINE: | START BIT | 00000001 | ACKNOWLEDGE BIT | DATA 7...0 |

If an X24C01 is used on the 2-wire bus, an acknowledge bit and eight data bits will be returned whereas 24C01A will issue no response and will ignore the command.

2.3 HARDWARE DIFFERENCES

Unlike the X24C01, the 24C01A is designed to share a 2-wire bus with other devices. Chip address bits are included in the slave address for the 24C01A, and are incorporated into pins 1, 2 and 3 of the device. They must be connected to Vcc or Vss for proper operation. Since pins 1, 2 and 3 of the Xicor part are NC (no connect) pins and they are not internally connected, they can be tied high or low.

Another hardware difference involves pin 7 which *MUST* be connected to Vss on the X24C01. The 24C01A can have pin 7 connected to Vss or Vcc.

If only one device is planned for the 2-wire bus, the board can be made compatible for either device by connecting pins 1, 2 and 3 to either Vss or Vcc and tying pin 7 to Vss.

Mobility For Memory Upgrade And Expansion

In system applications where the master device needs to address more than one serial EEPROM on a 2-wire bus, the Microchip 24C01A offers the flexibility. Up to eight 24C01A's can be connected to the 2-wire bus. More than one Xicor X24C01 connected to the bus may result in bus contention.

If memory upgrade is required, the Microchip 24C01A can be upgraded to the 24C02A (256 x 8) or the 24C04A (512 x 8) in the same IC socket with *NO change* in hardware. Using the Xicor X24C01, both software and hardware would have to be reconfigured to accommodate the changes.

*CONTACT: Bruce Negley
Memory Products Division*

Optimizing Serial Bus Operations with Proper Write Cycle Times

SERIAL EEPROM WRITE TIME REQUIREMENTS

Elements of the Write Cycle Time

The total write operation time for a Serial EEPROM is determined by three main elements:

- Number of bytes to load for each write operation
- Bus clock speed at which the write operation is loaded
- Fixed internal write cycle timer required for the programming operation

The load component of the write command consists of the control byte, address, and the data of up to 16 bytes. The time required to load the operation depends on the number of bytes to load at one time and the bus clock speed. After this load is complete, the part commences the internally controlled write cycle and the bus and system are free to perform other tasks. The internal write cycle timer is a fixed time delay which is required to program the EEPROM memory cells. Table 1 gives examples of total write time for 1 and 16 bytes for various parts at fast and normal clock speeds.

- **Load time (time at bus free)** is the time the part is being loaded with the instruction, address, and data. The bus is free after this time interval, and the part commences the internally controlled write cycle sequence.

- **Write timer (worse case)** indicates the time the part is in the internally controlled write cycle allowing for the maximum specified datasheet requirements.
- **Write timer (typical)** indicates the time the part is in the internally controlled write cycle assuming nominal conditions and utilizing write cycle polling.
- **Total write time** is the combined load time and typical internal write cycle time.

MINIMIZING SERIAL BUS COMMUNICATION TIME IN A SYSTEM

Utilizing the Page Write Option

The original Microchip Serial EEPROM products, though utilizing a page buffer, only write bytes sequentially. This means the time required to write 8 bytes is 8 ms, worse case. The new 24LCXX products incorporate a page mode that allows simultaneous writes of up to 16 bytes. This allows the programming of up to 16 bytes in one write cycle (10 ms) compared to 16 write cycles (16 ms) for the original 24CXX products. Microchip uses an 8 byte page in the 24LC01 and 24LC02, and a 16 byte page the 24LC04 and higher densities. What this means to a system designer is a write of 8 bytes in a 24LC01 or 24LC02 would take 10 ms worse case versus 8 ms

TABLE 1 - WRITE OPERATION TIME COMPARISON

Product	Page Width	# Bytes to load	Speed (KHz)	Load time (ms) time at bus free	Write timer (ms) Worse case	Write timer (ms) Typical (25C)	Typical total write time (ms)
24C01	2	1	100	0.28	1	0.4	0.68
		8 (4 x 2)	100	1.12	1	0.4	4.32
24LC01	8	1	100	0.28	10	3	3.28
		8	100	0.91	10	3	3.91
		1	400	0.07	10	3	3.07
		8	400	0.23	10	3	3.23
24C04	8	1	100	0.28	1	0.4	0.68
		16 (2 x 8)	100	1.82	1	0.4	8.22
24LC04	16	1	100	0.28	10	3	3.28
		16	100	1.63	10	3	4.63
		1	400	0.07	10	3	3.07
		16	400	0.41	10	3	3.41

Minimizing Serial Bus Communication Time

worse case in either a 24C01 or 24C02. Loading the entire memory of an original 24C04 takes 512 ms, but the same operation on a new 24LC04 is reduced to only 320 ms, assuming worse case. As table 1 shows, the difference is even greater using typical numbers.

The 93LCXX products do not utilize a page mode. However, in 16 bit mode, all 16 bits are written simultaneously. The original 93CXX products write the 16 bits in 2 write cycles, so the write cycle time increase of the new products is only 5 X (3.5 X typical).

Utilizing Write Cycle Polling

One powerful method of increasing programming efficiency is by periodically polling the part to determine if the write cycle has completed. To poll the 24LCXX series products, a control byte is sent and the acknowledge bit from the part is read. If the part acknowledges (pulls SDA low), it is ready to accept a new command. The part will not acknowledge while in the internally timed write cycle.

To poll the 93LCXX series products, the chip select is pulled high after the write cycle commences, and the data out line is read for the ready/busy status. If the part is still busy, it will pull the data line low. When the internally timed write cycle is complete, the part will pull the data line high, indicating it is ready for a new command.

Serial EEPROM System Optimization

Serial EEPROMs are used in systems for two purposes: storing data and reading back data. Read operations are at full clock speed, so the only methods for optimization are to run the clock at maximum frequency and to utilize sequential read whenever possible. Sequential read allows a continuous output data stream on one command.

The write operation, with its internal write timer component, needs special consideration when designing the control software. Efficient operation can be accomplished using both the page mode and write cycle polling. The following example shows a typical fetch-store operation in a system and how optimization can be incorporated. In the example system, the microcontroller must fetch bytes of data from a sensor and send the bytes to a EEPROM for storage. Two cases are shown in Figure 2: case 1 uses a 24C01A with no optimization; case 2 uses a 24LC01B with the available page mode and write cycle polling.

By utilizing the available page write mode and by polling for the write cycle completion, nearly four times as many bytes can be initially loaded to the serial EEPROM in the same time interval. Continuous operation for the optimized case 2 takes only 0.49 ms per byte compared to 1.28 ms per byte for the non-optimized case 1.

Minimizing Serial Bus Communication Time

FIGURE 2 - CASE COMPARISON

CASE 1 (NO OPTIMIZATION):		CASE 2 (USING PAGE MODE AND POLLING):	
Operation	Serial Bus Time	Operation	Serial Bus Time
fetch byte		fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		load 8 bytes to serial	0.91 ms
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		poll for write timer	<3 ms typical
wait for write timer	1 ms	load 8 bytes to serial	0.91 ms
load to serial	0.28 ms		
fetch byte			
wait for write timer	1 ms		
load to serial	0.28 ms		
First 8 bytes loaded in 9.24 ms		First 16 bytes loaded in 4.82 ms	

*AUTHOR: Lenny French
Memory Products Division*

Minimizing Serial Bus Communication Time

Notes:



Using the 93LC56 and 93LC66

INTRODUCTION

The Microchip Technology Inc. 93LC56/66 low-power 3-/4-wire non-volatile memories and are suitable for many embedded system code and data storage. These devices are easily interfaced to most microcontrollers in today's market place, but Microchip's 8-bit RISC series PIC16CXX offer the best code density of any microcontroller on the market today. Using the PIC16C54, the assembly programs contained in this application note have been fully tested and provide the correct timing and 3-wire sequences to fully operate the 93LC56/66 in a PIC16CXX-based embedded application. The PIC16C54 was clocked at a 10 MHz frequency. This application note is intended to provide the engineer with readily available stand alone code modules to accomplish all of the necessary functions to utilize these devices in a low power application using the efficient PIC16C54 microcontroller.

The 93 series of devices have essentially four I/O pins:

CS	Chip Select
CLK	Clock
DI	Data In
DO	Data Out

This series of devices use a series of commands to accomplish the normal memory functions. These are READ, WRITE, EWEN, ERASE, ERAL, WRAL, EWDS. For a more detailed discussion of the function of these devices reference the appropriate data sheet and AN536, also published by Microchip Technology.

The following programs are included in this application note and are fully functional stand alone modules.

3-wire Byte Read Program

- Start Bit Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

3-wire Byte Write Program

- Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine (EWEN)
- Byte Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Byte Write with Data Polling Program

- Data Polling Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine
- Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Sequential Read Program

- Delay Routine
- Start Bit Routine
- Bit In Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

*AUTHOR: Bruce Negley
Memory Products Division*

Using the 93LC56 and 93LC66

```
LIST P=16C54
;*****
;      3-Wire Byte Read Program (80 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
;      device. This program will read 8 consecutive addresses
;      in the 'random read' mode. This means that the opcode
;      and address for each byte will be sent to the device.
;      This program will repeat forever.
;
;      Another, more efficient method of reading consecutive
;      addresses is called the 'sequential read' mode. This
;      involves sending the opcode and address for the first
;      byte to read, then continuing to provide clocks for the
;      next addresses. The device will automatically increment
;      the address. An example of the sequential read mode is
;      provided in the '3wseqr.asm' file.
;
;      This program communicates to the serial EE in the
;      x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
;      ON THE DEVICE TO Vcc.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 50 KHz for this setup.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      - - - - -
;      Pin 10 (RB4) -> Chip Select
;      Pin 11 (RB5) -> Clock
;      Pin 12 (RB6) -> Data In
;      Pin 13 (RB7) -> Data Out
;      ORG = Vcc
;
;*****
;      Register Assignments
;*****
port_a equ 5h      ; port 5 (port_a)
port_b equ 6h      ; port 6 (port_b) conn lines to serial EE
eeprom equ 0ah     ; bit buffer
addr   equ 0ch     ; address register
datai  equ 0dh     ; stored data input reg.
datao  equ 0eh     ; stored data output reg.
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bits transmitted so far
bits   equ 12h     ; bits to transmit
bytcnt equ 13h     ; byte counter for read routine
loops  equ 15h     ; delay loop counter
loops2 equ 16h     ; delay loop counter
hbyte  equ 17h     ; high byte for input data
lbyte  equ 18h     ; low byte for input data
;*****
;      Bit Assignments
;*****
d      equ 7 ; eeprom input
do     equ 6 ; eeprom output
datout equ 7 ; data out line (port_b)
datin  equ 6 ; data in line (port_b)
sclk   equ 5 ; clock line (port_b)
chpsel equ 4 ; chip select line (port_b)
;
;*****
org    01ffh
begin  goto  PWRUP ; set the reset vector
org    000h
goto  PWRUP
;
```

Using the 93LC56 and 93LC66

```
*****
;
;   Start Bit Subroutine
;   this routine generates a start bit
;   (Chip select and DI high when clock goes high)
;*****
BSTART
    bcf    port_b, datin    ; set datain and chipselect lines
    bcf    port_b, chpsel   ; low just to check operation
    bcf    port_b, sclk     ; make sure clock starts low too.
    nop

;
    bcf    port_b, chpsel   ; set chip select line high
    bcf    port_b, datin    ; set data in line high
    nop
    bcf    port_b, sclk     ; set the clock line high to
                            ; generate the start bit

    nop
    nop
    bcf    port_b, sclk     ; set clock low again
    retlw  0

;
;*****
;   BITIN routine reads one bit of data from the
;   serial EE device and stores it in 'di'
;*****
BITIN
    bcf    eeprom, di       ; assume input bit is high
    bcf    port_b, sclk     ; set clock line high
    nop
    btfs   port_b, datout   ; read the data bit
    bcf    eeprom, di       ; input bit was low
    bcf    port_b, sclk     ; set clock line low
    ;
    retlw  0

;
;*****
;   Receive data routine
;   This routine reads one byte of data from the part
;   into the 'datai' register.
;*****
RX
    clrf   datai           ; clear input buffer
    movlw  .8              ; set # bits to 8
    movwf  count

RXLP   rlf    datai         ; rotate the buffer left 1 bit
       call  BITIN         ; read 1 bit
       bcf   datai, 0       ; assume the input bit was low
       btfs  eeprom, di     ; check the bit
       bsf   datai, 0       ; set high if necessary
       decfsz count        ; 8 bits done?
       goto RXLP           ; no, do another
       retlw 0

;
;*****
;   BITOUT routine
;   This routine takes one bit of data in 'do' and
;   transmits it to the serial EE device
;*****
BITOUT
    btfs   eeprom, do       ; check state of data bit
    goto   bitlow           ; low, goto bitlow
    bsf    port_b, datin    ; high, set datain high
    goto   clkout           ; and clock it
    ;
bitlow bcf    port_b, datin  ; output a logic low
clkout bsf    port_b, sclk   ; set clock line high
       nop
       bcf    port_b, sclk   ; return clock line low
       retlw  0

;
```

Using the 93LC56 and 93LC66

```
*****
;
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; data0' register and transmits it to the serial EE device.
*****
TX
    movf    bits,w    ; set the number of bits to xmit
    movwf   count
;
TXLP
    bcf     eeprom,do ; assume bit 7 is low
    btfsc  txbuf,7    ; is bit 7 clear?
    bsf     eeprom,do ; no, set data bit =1
    call   BITOUT     ; transmit 1 bit to serial EE
    rlf    txbuf      ; rotate txbuf left
    decfsz count      ; all bits done?
    goto   TXLP       ; no, do another bit
    retlw  0          ; yes, jump out
;
*****
; POWER-UP ROUTINE
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; read routine.
*****
PWRUP
;
    movlw  b'00000000'
    tris  port_a    ; set port_a as all output
    chf   port_a    ; all lines low

    movlw  b'10000000'
    tris  port_b    ; set RB7 as input, rest output
;
; Fall through and do the read
;
*****
; READ ROUTINE
; This routine reads 8 consecutive addresses in
; random mode starting at address 0. This is done in
; x16 mode and will repeat forever.
*****
READ
;
    movlw  .0        ; set starting address to 00
    movwf  addr      ;
    movlw  .8        ; set number of addresses to
    movwf  bytcnt    ; read as 8
;
rbyte    call   BSTART ; generate the start bit
    movlw  .2        ; set # bits to 2
    movwf  bits      ;
    movlw  b'10000000' ; get opcode (10b)
    movwf  txbuf     ; into output buffer
    call   TX        ; and transmit it
    movlw  .8        ;
    movwf  bits      ; set number of bits to 8
    movf   addr,w    ; get the address
    movwf  txbuf     ; into the output buffer
    call   TX        ; and transmit it

    call   RX        ; read the high byte
    movf   data1,w   ; move input data to w
    movwf  hbyte     ; xfer it to high byte

    call   RX        ; read the low byte
    movf   data1,w   ; move input data to w
    movwf  hbyte     ; xfer it to low byte
```

Using the 93LC56 and 93LC66

```
bcf    port_b, chpsel    ; clear the chip select line

incf   addr              ; add 1 to the address
decfsz bytcnt            ; have all bytes been read?
goto   rbyte             ; no, read another byte
goto   READ              ; yes, start over

END
```

Using the 93LC56 and 93LC66

```
LIST P=16C54
;*****
;
;       3-Wire Byte Write Program (106 bytes)
;
;
;       This program demonstrates how to interface a
;       Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
;       device. This program will execute the erase/write enable
;       command, write to 8 consecutive addresses, and then
;       execute the erase/write disable command. This
;       sequence will repeat forever.
;
;
;       After each byte is written, time must be given to the
;       device for it to complete the write cycle before
;       the next command can be sent. The easiest solution
;       is to consult the data book for the maximum write
;       cycle time and just wait that long before the next
;       command is sent. This program demonstrates that
;       solution.
;
;
;       Another, more efficient method of determining when the
;       write cycle is complete is called 'data polling.' This
;       method is demonstrated in the program "3wdpoll."
;
;
;       This program communicates to the serial EE in the
;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
;       ON THE DEVICE TO Vcc.
;
;
;       Timing is based on using the PIC16CXX in 'XT' mode
;       using a 4Mhz crystal. Clock speeds to the serial EE
;       will be approximately 40 Khz for this setup.
;
;
;       PIC16CXX to Serial EE Connections:
;
;
;       PIC16CXX      Serial EE
;       - - - - -    - - - - -
;
;       Pin 10 (RB4) -> Chip Select
;       Pin 11 (RB5) -> Clock
;       Pin 12 (RB6) -> Data In
;       Pin 13 (RB7) -> Data Out
;
;       ORG=Vcc
;
;
;*****
;
;       Register Assignments
;*****
port_a  equ  5h      ; port 5 (port_a)
port_b  equ  6h      ; port 6 (port b) conn lines to serial EE
eeprom  equ  0ah     ; bit buffer
addr    equ  0ch     ; address register
datai   equ  0dh     ; stored data input reg.
datao   equ  0eh     ; stored data output reg.
txbuf   equ  10h     ; transmit buffer
count   equ  11h     ; bits transmitted so far
bits    equ  12h     ; bits to transmit
bytcnt  equ  13h     ; byte counter for write routine
loops   equ  15h     ; delay loop counter
loops2  equ  16h     ; delay loop counter
;*****
;
;       Bit Assignments
;*****
d       equ  7       ; eeprom input
do      equ  6       ; eeprom output
datout  equ  7       ; data out line (port_b)
datin   equ  6       ; data in line (port_b)
sckl    equ  5       ; clock line (port_b)
chpsel  equ  4       ; chip select line (port_b)
;
;*****
```


Using the 93LC56 and 93LC66

```
begin    org    01ffh
        goto   PWRUP ; set the reset vector
        org    000h
        goto   PWRUP
;
;*****
;      DELAY ROUTINE
;      This routine takes the value in 'loops'
;      and multiplies it times 1 millisecond to
;      determine delay time.
;*****
WAIT
;
top      movlw  .110 ; timing adjustment variable
        movwf  loops2
top2     nop        ; sit and wait
        nop
        nop
        nop
        nop
        nop
        decfsz loops2 ; inner loops complete?
        goto   top2  ; no, go again
        ;
        decfsz loops ; outer loops complete?
        goto   top   ; no, go again
        retlw  0     ; yes, return from sub
;
;*****
;      Start Bit Subroutine
;      this routine generates a start bit
;      (Chip select and DI high when clock goes high)
;*****
BSTART
        movlw  b'10001111'
        tris   port_b ; set port b for output
                    ; except for the data out line
        bcf   port_b, datin ; set datain and chipselect lines
        bcf   port_b, chpsel ; low just to check operation
        bcf   port_b, sclk  ; make sure clock starts low too.
        nop
;
        bsf   port_b, chpsel ; set chip select line high
        bsf   port_b, datin  ; set data in line high
        nop
        bsf   port_b, sclk   ; set the clock line high to
                    ; generate the start bit
        nop
        nop
        bcf   port_b, sclk   ; set clock low again
        retlw  0
;
;*****
;      BITOUT routine
;      This routine takes one bit of data in 'do' and
;      transmits it to the serial EE device
;*****
BITOUT
        btfs  eeprom, do ; check state of data bit
        goto  bitlow ; low, goto bitlow
        bsf   port_b, datin ; high, set datain high
        goto  clkout ; and clock it
;
bitlow  bcf   port_b, datin ; output a logic low
clkout  bsf   port_b, sclk  ; set clock line high
        nop
        bcf   port_b, sclk  ; return clock line low
        retlw  0
;
```

Using the 93LC56 and 93LC66

```
*****
;
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'data0' register and transmits it to the serial EE device.
*****
TX
    movf    bits,w           ; set the number of bits to xmit
    movwf   count

;
TXLP
    bcf     eeprom,do       ; assume bit 7 is low
    btfsz  txbuf,7         ; is bit 7 clear?
    bsf     eeprom,do       ; no, set data bit =1
    call   BITOUT          ; transmit 1 bit to serial EE
    rlf    txbuf            ; rotate txbuf left
    decfsz count           ; all bits done?
    goto  TXLP             ; no, do another bit
    retlw  0               ; yes, jump out
;
*****
; POWER-UP ROUTINE
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; erase/write enable routine.
*****
PWRUP
;
    movlw  b'00000000'
    tris   port_a          ; set port_a as all output
    chf    port_a          ; all lines low

    movlw  b'10000000'
    tris   port_b          ; set RB7 as input, rest output
;
; Fall through and do erase/write enable
;
*****
; EWEN (Erase/Write ENable Routine)
; this routine enables the dut for erasing and
; writing. This must be done prior to any erase, write
; eral,wral instructions.
*****
EWEN
;
    call   BSTART          ; generate a start bit
;
    movlw  .2              ; set # bits to 2
    movwf  bits            ;
    movlw  b'00000000'     ; get the opcode (00b)
    movwf  txbuf           ; into the output buffer
    call   TX              ; and transmit it
    movlw  .8              ; set # bits to 8
    movwf  bits            ;
    movlw  b'11000000'     ; get opcode and address
;                          ; (11XXXXXX)
    movwf  txbuf           ; into output buffer
    call   TX              ; and transmit it
    bcf    port_b,chpsel   ; set chip select line low
    nop
;
; Now continue on to the write command
;
*****
; Byte Write Routine
; This routine writes an AA55h pattern into
; 8 consecutive addresses starting at address 00.
; A delay of about 10ms is given after each byte
; for the write cycle to complete.
; The write is done in the xl6 mode: the user must
; have the ORG pin tied to Vcc on the device
```

Using the 93LC56 and 93LC66

```
*****
WRITE
;
;
;   movlw   .0           ; set starting address to 00
;   movwf   addr        ;
;   movlw   .8           ; set number of bytes to write as 8
;   movwf   bytcnt     ;
;
topwr  call   BSTART    ; generate the start bit
;
;   movlw   .2           ; set # bits to 2 for the opcode
;   movwf   bits        ;
;   movlw   b'01000000' ; get opcode (01b)
;   movwf   txbuf       ; into the transmit buffer
;   call    TX          ; and send it
;
;   movlw   .8           ; set # of bits to 8 for the
;   movwf   bits        ; address
;   movf    addr,w      ; get address counter
;   movwf   txbuf       ; into output buffer
;   call    TX          ; and send it
;   movlw   b'10101010' ; get upper byte of data (AAh)
;   movwf   txbuf       ; into the transmit buffer
;   call    TX          ; and send it
;   movlw   b'01010101' ; get lower byte of data (55h)
;   movwf   txbuf       ; into transmit buffer
;   call    TX          ; and send it
;
;   bcf     port_b, chpsel ; clear the chip select line
;                               ; to initiate write cycle
;
;   movlw   .10          ;
;   movwf   loops       ; set delay time to 10mS
;   call    WAIT        ; and wait
;
;   incf   addr         ; increment address counter
;   decfsz bytcnt       ; all bytes written?
;   goto   topwr        ; no, do another
;                               ; yes, go on
;
;   Now continue on to the erase/write disable command
;
;*****
;   EWDS (Erase/Write Disable Routine)
;   This routine executes the erase/write disable command
;   which prevents the contents of the array from being
;   written to.
;
;*****
EWDS
;
;   call    BSTART      ; generate a start bit
;
;   movlw   .2           ; set # bits to 2
;   movwf   bits        ;
;   movlw   b'00000000' ; get the opcode (00b)
;   movwf   txbuf       ; into the output buffer
;   call    TX          ; and transmit it
;   movlw   .8           ; set # bits to 8
;   movwf   bits        ;
;   movlw   b'00000000' ; get opcode and address
;                               ; (00XXXXXX)
;   movwf   txbuf       ; into output buffer
;   call    TX          ; and transmit it
;   bcf     port_b, chpsel ; set chip select line low
;   nop
;   goto   EWEN        ; start all over
;
END
```

Using the 93LC56 and 93LC66

```
LIST P=16C54
;*****
;      3-Wire Byte Write with Data Poll Program (107 bytes)
;
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
;      device. This program will execute the erase/write enable
;      command, write to 8 consecutive addresses, and then
;      use the 'data polling' method to determine when the
;      write cycle is complete. The program then executes
;      the erase/write disable command. This sequence will
;      repeat forever.
;
;
;      When writing to a 3-wire serial EE device, the
;      internally timed write cycle will begin after
;      the opcode, address and data are sent to the
;      device. There are two ways to make sure that the
;      cycle is complete before you send it another command.
;      The simplest method is to simply wait for the maximum
;      write cycle time, which can be obtained from the data book.
;      A more efficient method is "Data polling." This is
;      done by toggling the chip select line low and then
;      back high after the opcode, address and data are sent.
;      The chip select line must be low for at least 250ns
;      before bringing it high again. The device will pull
;      the data out line low at that point, and set it high
;      when the write cycle is complete. The user can then check
;      or "poll" the dataout line until it goes high before
;      sending the next command.
;
;
;      As an option, the user can connect a LED to pin 18
;      on the PIC16CXX (use about a 1K resistor in series) as a
;      timeout indicator. This LED will come on if the
;      data polling is unsuccessful and the device being
;      programmed does not respond. This can be tested by
;      simply running the program with no part in the socket.
;
;
;      This program communicates to the serial EE in the
;      x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
;      ON THE DEVICE TO Vcc.
;
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 50 KHz for this setup.
;
;
;      PIC16CXX to Serial EE Connections:
;
;
;      PIC16CXX      Serial EE
;      - - - - -      - - - - -
;      Pin 10 (RB4) -> Chip Select
;      Pin 11 (RB5) -> Clock
;      Pin 12 (RB6) -> Data In
;      Pin 13 (RB7) -> Data Out
;      ORG=Vcc
;
;
;*****
;      Register Assignments
;*****
port_a equ 5h      ; port 5 (port_a)
port_b equ 6h      ; port 6 (port b) used comm lines to serial EE
eeprom equ 0ah     ; bit buffer
addr   equ 0ch     ; address register
data1  equ 0dh     ; stored data input reg.
data0  equ 0eh     ; stored data output reg.
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bits transmitted so far
bits   equ 12h     ; bits to transmit
bytcnt equ 13h     ; byte counter for write routine
loops  equ 15h     ; delay loop counter
loops2 equ 16h     ; delay loop counter
```

Using the 93LC56 and 93LC66

```
*****
;
;   Bit Assignments
;*****
d      equ 7 ; eeprom input
do     equ 6 ; eeprom output
datout equ 7 ; data out line (port_b)
datin  equ 6 ; data in line (port_b)
sclk   equ 5 ; clock line (port_b)
chpsel equ 4 ; chipselect line (port_b)
timeout equ 1 ; write cycle timeout warning (port_a)
;
;*****
org    01ffh
begin  goto  PWRUP ; set the reset vector
org    000h
goto  PWRUP
;
;*****
;   DATA POLL DELAY ROUTINE
;   This routine delays 100 us is
;   used for delay between polls
;*****
dpdelay
;
;   movlw .25 ; timing adjustment variable
movwf  loops2
top    nop
decfsz loops2 ; loops complete?
goto  top ; no, go again
;
retlw  0 ; yes, return from sub
;
;*****
;   Start Bit Subroutine
;   this routine generates a start bit
;   (Chip select and DI high when clock goes high)
;*****
BSTART
bcf    port_b,datin ; set datain and chipselect lines
bcf    port_b,chpsel ; low just to check operation
bcf    port_b,sclk ; make sure clock starts low too.
nop
;
bsf    port_b,chpsel ; set chip select line high
bsf    port_b,datin ; set data in line high
nop
bsf    port_b,sclk ; set the clock line high to
; generate the start bit
nop
nop
bcf    port_b,sclk ; set clock low again
retlw  0
;
;*****
;   BITOUT routine
;   This routine takes one bit of data in 'do' and
;   transmits it to the serial EE device
;*****
BITOUT
btfs  eeprom,do ; check state of data bit
goto  bitlow ; low, goto bitlow
bsf    port_b,datin ; high, set datain high
goto  clkout ; and clock it
;
bitlow bcf    port_b,datin ; output a logic low
clkout bsf    port_b,sclk ; set clock line high
nop
bcf    port_b,sclk ; return clock line low
retlw  0
;
```

Using the 93LC56 and 93LC66

```
*****
;
;   Transmit Data Subroutine
;   This routine takes the byte of data stored in the
;   'datao' register and transmits it to the serial EE device.
*****
TX
    movf    bits,w           ; set the number of bits to xmit
    movwf   count
;
TXLP
    bcf     eeprom,do       ; assume bit 7 is low
    btfsz  txbuf,7         ; is bit 7 clear?
    bsf     eeprom,do       ; no, set data bit =1
    call   BITOUT          ; transmit 1 bit to serial EE
    rlf    txbuf            ; rotate txbuf left
    decfsz count           ; all bits done?
    goto  TXLP             ; no, do another bit
    retlw  0                ; yes, jump out
;
*****
;
;   POWER-UP ROUTINE
;   This is the program entry point, which in this case simply
;   sets the port_a I/O lines and directs control to the
;   erase/write enable routine.
*****
PWRUP
;
    movlw  b'00000000'
    tris  port_a           ; set port_a as all output
    cdfc  port_a           ; all lines low
;
    movlw  b'10000000'
    tris  port_b           ; set RB7 as input, rest output
;
;   Fall through and do erase/write enable
;
*****
;
;   EWEN (Erase/Write ENable Routine)
;   this routine enables the dut for erasing and
;   writing. This must be done prior to any erase, write
;   eral,wral instructions.
*****
EWEN
;
    call   BSTART          ; generate a start bit
;
    movlw  .2              ; set # bits to 2
    movwf  bits
;
    movlw  b'00000000'     ; get the opcode (00b)
    movwf  txbuf           ; into the output buffer
    call   TX              ; and transmit it
    movlw  .8              ; set # bits to 8
    movwf  bits
;
    movlw  b'11000000'     ; get opcode and address
;                               ; (11XXXXXX)
    movwf  txbuf           ; into output buffer
    call   TX              ; and transmit it
    bcf   port_b,chpsel    ; set chip select line low
    nop
;
;   Now continue on to the write command
;
*****
;
;   WRITE
;   This routine writes a AA55h pattern into
;   8 consecutive addresses starting at address 00.
;   It will then 'poll' the data out line to determine
;   when the write cycle is complete. If the cycle has
;   not completed within 20 ms, then it will continue
```

Using the 93LC56 and 93LC66

```
; anyway and turn the timeout fail LED on.
; The write is done in the x16 mode: the user must
; have the ORG pin tied to Vcc on the device. This
; program will repeat forever.
;*****
WRITE
;
    movlw    .0            ; set starting address to 00
    movwf   addr          ;
    movlw   .8            ; set number of bytes to write as 8
    movwf   bytcnt        ;
;
topwr    call   BSTART      ; generate the start bit
;
    movlw   .2            ; set # bits to 2 for the opcode
    movwf   bits          ;
    movlw   b'01000000'   ; get opcode (01b)
    movwf   txbuf         ; into the transmit buffer
    call    TX            ; and send it
;
    movlw   .8            ; set # of bits to 8 for the
    movwf   bits          ; address
    movf    addr,w        ; get address counter
    movwf   txbuf         ; into output buffer
    call    TX            ; and send it
    movlw   b'10101010'   ; get first half of data (AAh)
    movwf   txbuf         ; into the transmit buffer
    call    TX            ; and send it
    movlw   b'01010101'   ; get second half of data (55h)
    movwf   txbuf         ; into transmit buffer
    call    TX            ; and send it
;
    bcf     port_b,chpsel  ; clear the chip select line
; to initiate write cycle
;
    nop
    bsf     port_b,chpsel  ; set the chip sel line high
; to begin data polling
;
    movlw   .200          ;
    movwf   loops         ; poll 100 times before timeout (about 20ms)
polltop  call   dpdelay      ; wait 100us
    btfsc   port_b,datout  ; is the data out line high?
    goto    okpoll        ; yes-cycle complete: do another
    decfsz  loops         ; has it timed out?
    goto    polltop       ; no, check again
    bsf     port_a,timeout ; yes, set timeout LED and go on
    goto    badpoll       ;
;
okpoll   bcf     port_a,timeout ; clear the timeout LED
badpoll  bcf     port_b,chpsel  ; chip sel line back low
    incf   addr          ; increment address counter
    decfsz bytcnt        ; all bytes written?
    goto   topwr         ; no, do another
; yes, go on
;
; Now continue on to the erase/write disable command
;
;*****
; EWDS (Erase/Write Disable Routine)
; This routine executes the erase/write disable command
; which prevents the contents of the array from being
; written to.
;*****
```

Using the 93LC56 and 93LC66

```
EWDS
;
;   call    BSTART          ; generate a start bit
;
movlw  .2                ; set # bits to 2
movwf  bits              ;
movlw  b'00000000'      ; get the opcode (00b)
movwf  txbuf            ; into the output buffer
call   TX               ; and transmit it
movlw  .8                ; set # bits to 8
movwf  bits              ;
movlw  b'00000000'      ; get opcode and address
;   (00XXXXXX)
movwf  txbuf            ; into output buffer
call   TX               ; and transmit it
bcf    port_b,chpsel    ; set chip select line low
nop
goto  EWEN              ; start all over
;
;
END
```



```

LIST P=16C54
;*****
;      3-Wire Sequential Read Program (93 bytes)
;
;      This program demonstrates how to interface a
;      Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
;      device. This program will read 8 consecutive addresses
;      in the 'sequential read' mode. This means that the opcode
;      and address are sent for the first address only. After
;      the first address is read, the Chip Select line is left
;      high and clocks for the remaining 7 bytes are sent to the
;      device. The device will automatically increment the address
;      pointer in this mode, allowing the user to read as many
;      consecutive addresses as needed. This program will repeat
;      forever.
;
;      This program communicates to the serial EE in the
;      x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
;      ON THE DEVICE TO Vcc.
;
;      Timing is based on using the PIC16CXX in 'XT' mode
;      using a 4Mhz crystal. Clock speeds to the serial EE
;      will be approximately 50 Khz for this setup.
;
;      PIC16CXX to Serial EE Connections:
;
;      PIC16CXX      Serial EE
;      - - - - -    - - - - -
;      Pin 10 (RB4) -> Chip Select
;      Pin 11 (RB5) -> Clock
;      Pin 12 (RB6) -> Data In
;      Pin 13 (RB7) -> Data Out
;      ORG = Vcc
;
;*****
;      Register Assignments
;*****
port_a equ 5h      ; port 5 (port_a)
port_b equ 6h      ; port 6 (port_b) used comm lines to serial EE
eeprom equ 0ah     ; bit buffer
addr   equ 0ch     ; address register
datai  equ 0dh     ; stored data input reg.
datao  equ 0eh     ; stored data output reg.
txbuf  equ 10h     ; transmit buffer
count  equ 11h     ; bits transmitted so far
bits   equ 12h     ; bits to transmit
bytcnt equ 13h     ; byte counter for read routine
loops  equ 15h     ; delay loop counter
loops2 equ 16h     ; delay loop counter
hbyte  equ 17h     ; high byte for input data
lbyte  equ 18h     ; low byte for input data
;*****
;      Bit Assignments
;*****
d      equ 7       ; eeprom input
do     equ 6       ; eeprom output
datout equ 7       ; data out line (port_b)
datin  equ 6       ; data in line (port_b)
sclk   equ 5       ; clock line (port_b)
chpsel equ 4       ; chip select line (port_b)
;
;*****
org    01ffh
begin  goto PWRUP ; set the reset vector
      org 000h
      goto PWRUP
;*****

```

Using the 93LC56 and 93LC66

```
;      DELAY  ROUTINE
;      This routine takes the value in 'loops'
;      and multiplies it times 1 millisecond to
;      determine delay time.
;*****
WAIT
;
top    movlw  .110          ; timing adjustment variable
      movwf  loops2
top2   nop                 ; sit and wait
      nop
      nop
      nop
      nop
      decfsz loops2        ; inner loops complete?
      goto  top2          ; no, go again
      ;
      decfsz loops        ; outer loops complete?
      goto  top           ; no, go again
      retlw  0            ; yes, return from sub
;
;*****
;      Start Bit Subroutine
;      this routine generates a start bit
;      (Chip select and DI high when clock goes high)
;*****
BSTART
      bcf   port_b, datin   ; set datain and chipselect lines
      bcf   port_b, chpsel  ; low just to check operation
      bcf   port_b, sclk    ; make sure clock starts low too.
      nop
;
      bsf   port_b, chpsel  ; set chip select line high
      bsf   port_b, datin   ; set data in line high
      nop
      bsf   port_b, sclk    ; set the clock line high to
                          ; generate the start bit
      nop
      nop
      bcf   port_b, sclk    ; set clock low again
      retlw  0
;
;*****
;      BITIN routine reads one bit of data from the
;      serial EE device and stores it in 'di'
;*****
BITIN
      bsf   eeprom, di      ; assume input bit is high
      bsf   port_b, sclk    ; set clock line high
      nop
      btfs  port_b, datout  ; read the data bit
      bcf   eeprom, di      ; input bit was low
      bcf   port_b, sclk    ; set clock line low
      ;
      retlw  0
;
;*****
;      Receive data routine
;      This routine reads one byte of data from the part
;      into the 'datai' register.
;*****
RX
      clrf  datai           ; clear input buffer
      movlw .8              ; set #bits to 8
      movwf count
RXLP   rlf   datai           ; rotate the buffer left 1 bit
      call  BITIN           ; read 1 bit
      bcf  datai, 0         ; assume the input bit was low
```

Using the 93LC56 and 93LC66

```

    btfsc    eeprom,di        ; check the bit
    bsf      datai,0          ; set high if necessary
    decfsz   count           ; 8 bits done?
    goto     RXLP            ; no, do another
    retlw    0

;
;*****
; BITOUT routine
; This routine takes one bit of data in 'do' and
; transmits it to the serial EE device
;*****
BITOUT
    btfss    eeprom,do        ; check state of data bit
    goto     bitlow           ; low, goto bitlow
    bsf      port_b,datin     ; high, set datain high
    goto     clkout           ; and clock it
                                ;
bitlow    bcf      port_b,datin ; output a logic low
clkout    bsf      port_b,sclk ; set clock line high
          nop
          bcf      port_b,sclk ; return clock line low
          retlw    0
;
;*****
; Transmit Data Subroutine
; This routine takes the byte of data stored in the
; 'datao' register and transmits it to the serial EE device.
;*****
TX
    movf     bits,w           ; set the number of bits to xmit
    movwf    count
                                ;
TXLP
    bcf      eeprom,do        ; assume bit 7 is low
    btfsc    txbuf,7          ; is bit 7 clear?
    bsf      eeprom,do        ; no, set data bit =1
    call     BITOUT           ; transmit 1 bit to serial EE
    rlf      txbuf             ; rotate txbuf left
    decfsz   count           ; all bits done?
    goto     TXLP            ; no, do another bit
    retlw    0                ; yes, jump out
;
;*****
; POWER-UP ROUTINE
; This is the program entry point, which in this case simply
; sets the port_a I/O lines and directs control to the
; erase/write enable routine.
;*****
PWRUP
;
    movlw    b'00010001'
    tris    port_a            ; set RA0 as input, rest output
    clrf    port_a            ; all lines low
    movlw    b'10000000'
    tris    port_b            ; set RB7 as input, rest output
;
; Fall through and do the read
;
;*****
; READ ROUTINE (Sequential read mode)
; This routine reads 8 consecutive addresses in
; sequential mode starting at address 0. This
; program will repeat forever
;*****
READ
;
    movlw    .0                ; set starting address to 00
    movwf    addr              ;
    movlw    .8                ; set number of addresses to

```

Using the 93LC56 and 93LC66

```
movwf bytcnt          ; read as 8
                       ;
call   BSTART         ; generate the start bit
movlw  .2             ; set #bits to 2
movwf  bits           ;
movlw  b'10000000'   ; get opcode (10b)
movwf  txbuf          ; into output buffer
call   TX             ; and transmit it
movlw  .8             ;
movwf  bits           ; set number of bits to 8
movf   addr,w         ; get the address
movwf  txbuf          ; into the output buffer
call   TX             ; and transmit it

rbyte call   RX             ; read the high byte
      movf  data1,w        ; move input data to w
      movwf hbyte         ; xfer it to high byte

      call   RX             ; read the low byte
      movf  data1,w        ; move input data to w
      movwf hbyte         ; xfer it to low byte

      incf  addr           ; add 1 to the address
      decfsz bytcnt        ; have all bytes been read?
      goto  rbyte         ; no, read another byte

      bcf   port_b, chpsel ; yes, clear the chip select line
      goto  READ          ; and start over

END
```


ER59256/93C06 and NMC9306/NMC93C06

HARDWARE DIFFERENCES

PARAMETER	Microchip		National		
	ER59256	93C06	NMC9306	NMC93C06	
PIN 6	NC	NC	NC	NC	
PIN 7	TEST	NC	NC	NC	
CLOCK FREQ	250 kHz	1 MHz	250 kHz	1 MHz	
CLK DUTY CYCLE	25% - 75%	not specified	not specified	not specified	
CLK HIGH TIME min.	not specified	500 ns	1 us	250 ns	
CLK LOW TIME min.	not specified	500 ns	1 us	250 ns	
CS LOW TIME min.	not specified	100 ns	1 us	250 ns	
ENDURANCE	10K	100K	40K	40K	
ESD RATING	1.0 KV	4.0 KV	2.0 KV	2.0 KV	
E/W TIME	10 - 30 ms	2 ms	10 - 30 ms	10 ms	
ACTIVE CURRENT	10 mA	4 mA	10 mA	3 mA	
STANDBY CURRENT	3 mA	100AA	3 mA	50 AA	

Pin 6 and 7 configuration -

Microchip uses pins 6 and 7 of the ER59256 for factory internal test purposes. Pin 6 is used to monitor the on-chip charge pump which generates the required internal high programming voltage (>20 volts) during the ERASE and WRITE cycles. Any circuitry connected to this pin will reduce the data retention of the device or even inhibit programming. Signals on pin 7 can force the device into its internal test modes resulting in overprogramming all memory locations. It is, therefore, recommended that pin 6 be left open and pin 7 tied to Vss for normal operation.

Microchip's 93C06 and National's NMC9306/NMC93C06 do not use pins 6 and 7. To make the ER59256 compatible, the TEST pin (pin 7) can be left floating but must be noise-free.

Clock high time, clock low time -

For a clock frequency of 250 kHz, both the ER59256 and the NMC9306 have the same electrical parameters:

250 kHz or 4 μ s clock cycle time
 25% of the clock period (4 μ s) = 1 μ s (clock high time)
 75% of the clock period (4 μ s) = 3 μ s (clock low time =
 4 - 3 μ s = 1 μ s)

For slower clock frequencies, the ER59256 specification would restrict the user regarding clock low and high times:

A clock frequency of 100 kHz or 10 μ s cycle time would result in a clock low of 75% of 10 μ s or 7.5 μ s, and a clock high of 25% of 10 μ s or 2.5 μ s.

In reality, Microchip's ER59256 can meet National's NMC9306 specification regarding SK low and SK high of 1 μ sec over all operating frequencies as 250 kHz is the maximum allowable frequency and is, therefore, the worst case.

Chip select low time -

All parts require a chip select (CS) input low between any two instructions. Programming of the parts (ER59256, NMC9306, NMC93C06) begins at the falling edge of the CS. Microchip's 93C06, however, starts self-programming at the rising edge of the last data bit. CS going low and high with a minimum of chip select low time (Tcs) during programming can be used for polling purposes as described in 2.

CONTACT: Bruce Negley
 Memory Products Division



1.8 Volt Technology - Benefits

Portability is and will continue to be one of the key features driving the design and development of new electronic equipment. Existing products such as palmtop computers, cellular phones and data acquisition devices are becoming increasingly smaller and more powerful. Handheld markets will demand that product feature sets be enhanced and that battery operating life be increased. These market requirements, in turn, will put more demands on battery and integrated circuit technology.

In order to optimize battery technology ICs must be able to operate at voltages well below the 5V range that most ICs operate at today. In fact, even integrated circuits operating at 3 volts do not make the most of the available battery technology. Thus the need for devices that operate below 3 volts.

BATTERY TECHNOLOGY OVERVIEW

Primary versus Secondary

In portable or handheld equipment there are two major battery types employed; primary and secondary. Primary cells are disposable, and alkaline and lithium cells are the main primary batteries in use today. Alkaline batteries are extremely common in consumer applications such as electronic games, cordless phones and palmtop computers. While lithium batteries are used as a primary power source in other consumer products such as cameras and as a back-up supply in palmtops.

Secondary cells are rechargeable batteries. Among the many secondary cells available are lead-acid, Ni-cad and Nickel Metal Hydride (NMH) are the most popular.

Voltage Ratings

Batteries for portable applications, whether primary or secondary, have three different voltage ratings; Rated or nominal, Operating, and End-of-life. The rated voltage is usually the open circuit voltage, which in the case of alkaline "AA" cells is 1.5V. Under normal load the "operating voltage" is realized and in all cases is less than the nominal voltage. The final battery voltage rating

is the end-of-life voltage, which is defined as the voltage at which 100% of the usable power of the battery is consumed or as 75% of the operating voltage. In the case of the same alkaline battery the end-of-life voltage is 0.9 volts. Table 1 lists the operating and end-of-life voltages for a single "AA" or equivalent cell of the five battery types mentioned above.

TABLE 1 - BATTERY VOLTAGE SUMMARY

BATTERY TYPE	OPERATING VOLTAGE	END-OF-LIFE VOLTAGE
ALKALINE	1.2V	0.9V
LITHIUM	2.7V	2.0V
LEAD-ACID	2.0V	1.75V
NI-CAD	1.2V	0.9V
NMH	1.2V	1.0V

With applications employing primary cells this means that batteries are replaced less often, thus reducing operating cost as well as having a positive effect on the environment due to the use and disposal of fewer batteries. In systems with secondary cells time between recharging increased, thus enhancing the performance of the product.

LOW VOLTAGE APPLICATIONS

As mentioned previously, the need for increased portability will drive the employment of low voltage technology. Portable and handheld applications requiring low voltage (sub 3V) technology will be numerous and will in many cases be high volume. These include a variety of personal communications devices such as cellular and cordless phones, computing devices like palmtops and portable PCs, and a number of data acquisition devices for a variety of medical, industrial and commercial applications.

1.8 Volt Technology - Benefits

MICROCHIP'S LOW VOLTAGE PRODUCT POSITION

Prior to the announcement of a family of 1.8V Serial EEPROMs in November of 1992, Microchip had already established itself as a leader in low voltage semiconductor products. Microchip's current product portfolio consists of: 1) Serial EEPROMs that operate, both READ and WRITE, down to the following voltage levels; 1.8V, 2.0V, 2.5V and 5V; 2) ROM based microcontrollers that operate down to 2V, and 3) 3V One Time Programmable EPROMs.

The 1.8V Serial EEPROM family operates down to 1.8V without the relaxation of any specification. This includes all AC, DC retention and endurance parameters. Currently the 1.8V family consists of 3 three-wire devices, in densities ranging from 1K to 4K bits. Packaging options include 8 pin SOIC and 8 pin PDIP devices. Typical

operating currents for these devices, at 1.8V, are in the 70uA range, resulting in power consumption levels of less than 150uW. This compares to 1 to 5 mW for comparable devices operating at 5V. In the near future, the 1.8V family of Serial EEPROMs will be expanded to include all 2-wire devices.

SUMMARY

With the introduction of 1.8V Serial EEPROMs, Microchip has made available silicon technology that optimizes existing battery technology. This over time will result in the development, production, marketing and application of more portable and higher performance handheld and portable computing, telecommunications, and data acquisition products.

*Author: Tom Tyson
Memory Products Division*

Serial EEPROM Solutions vs. Parallel Solutions

In searching for solutions to their system non-volatile memory requirements, equipment, systems and product designers are faced with a plethora of design related issues and trade-offs. The non-volatile memory options available to them offer a variety of different device features including performance, ease of design, power consumption, operating voltage, programmability, density, and physical size. For the most part these non-volatile memory options can be grouped into two major categories: serial solutions and parallel solutions. This paper discusses the attributes of each, conducts a comparative analysis and in the process identifies the benefits and advantages of Serial EEPROMs.

SERIAL EEPROMS

The main feature that makes a device a "Serial" and sets it apart from parallel devices is, as its name implies, the ability to communicate through a serial interface. This ability has numerous benefits. First, serial communication is accomplished with a minimum number of I/O's. Serial EEPROMs require only two to four lines (depending on the hardware and software protocol) for complete communication; memory addressing, data input and output, and device control. Thus, the hardware interface requirements for Serial EEPROMs are kept at a minimum. The most common Serial EEPROMs in use today are devices that utilize a 2-wire protocol.

Another benefit of serial communication is package size. Ranging from densities of 256 to 16K bits, most Serial EEPROMs today are available in space-saving 8 pin PDIP and 150mil wide SOIC packaging. This obviously is very beneficial for applications where product size and weight is a key design factor. The final benefit is low current consumption. Due to a limited number of I/O ports and therefore on-chip support requirements, operating currents for Serial EEPROMs are usually well below 3 milli-amperes.

Other features of Serial EEPROMs include: 1) Byte programmability—The ability to erase and program one byte at a time without affecting the contents of the other memory locations in the array; 2) Clock rates of up to 6 Mhz—2-wire devices are rated at 100K hz and 400K hz per the standard I2C* protocol, while 3-wire devices can be operated at 6M hz rates; 3) Low voltage operation—Microchip has introduced a family of devices that operate, both read and write, down to 1.8V. This family complements other 2V and 2.5V low voltage Serial EEPROM families available.

PARALLEL NON-VOLATILE MEMORIES

There are a number of memory devices that fall into this category. The major ones include Parallel EEPROMs, Flash memory products, EPROMs, and SRAMs with battery back-up.

The main common feature of all of these devices is that communication with the device is done through a parallel interface, which results in a high system clock rate. Each type of device has separate data, address and control lines. Thus pin counts are in the 24 to 40 pin range. This also results in relatively large and costly packages and large footprints, even with the most advanced surface mount packages like TSOP. SRAMs with on-board batteries require DIP package heights that are significantly higher than those of standard DIP packages, adding to its package size and cost disadvantage.

Parallel EEPROM and battery backed-up SRAMs are the only two of the four major types of parallel non-volatile memories that have the capability to erase and program one byte at a time. EPROM and Flash devices require the whole array or at least large sectors to be erased prior to reprogramming.

SERIAL VERSUS PARALLEL

Serial EEPROMs have five major advantages over parallel non-volatile memories.

- 1) Lower Current Consumption - The maximum operating current (at 5 volts operating voltage) of a 16K serial EEPROM device is approximately an order of magnitude less than that of an equivalent density parallel EEPROM. Operating currents for 16K Serials are specified at 3mA, while 16K parallel devices are specified at 30mA and above. This relationship will continue as 64K serial devices are introduced. Since power consumption is directly proportional to current consumption, the lower the current the lower the power consumption.
- 2) Lower Voltage - Serial EEPROMs have been available in single supply low voltage options for some time. As mentioned above, Microchip has low voltage Serial EEPROMs that operate down to 1.8V, as well as other low voltage Serials that function down to 2.0V or 2.5V volts. 3V EPROMs

Serial EEPROM Solutions vs. Parallel Solutions

and parallel EEPROMs and single voltage 5V flash devices are just being introduced to the market. (Most flash devices on the market today require 12V for programming in addition to the 5 volts required for normal operation). Low voltage operation also has a positive effect on power consumption. A reduction in the operating voltage from 5 volts to 1.8 volts will result in a power consumption reduction of almost 90% and almost a 65% reduction in power if the operating voltage is reduced from 3V to 1.8V.

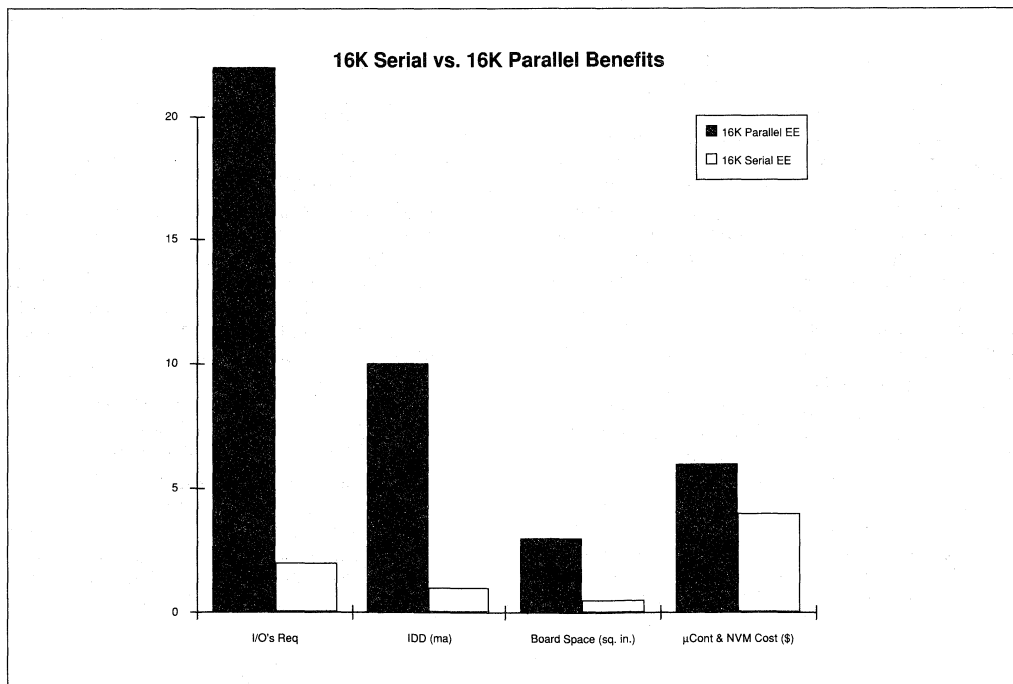
- 3) Programmability - Neither currently available Flash devices nor EPROMs have the ability to program one byte at a time. Erasing is an array or sector function. Therefore, whenever one byte needs to be reprogrammed the entire array or sector must be reprogrammed. This procedure takes a relatively long amount of time to complete, time which may not be available, as is the case when storing critical parameters or data during inadvertent and unexpected system power loss. This procedure also requires software overhead to manage the retrieval and reprogramming operation.
- 4) Physical Size - Again, when comparing a 16K Serial EEPROM to a 16K parallel device the serial has a significant advantage. The area of the 150mil 8 pin SOIC footprint is less than 50K square mils. This

compares to an area of more than 250K square mils for a 24 pin SOIC and almost 800K square mils for a 24 pin 500mil DIP package footprint.

- 5) I/O Requirements - Serials only require 2 to 4 input or output lines for complete communications. Most parallel devices require at least 22 lines, depending on the memory density. This results in increased microcontroller/microprocessor overhead and additional real estate to accommodate the numerous hardware lines.

The advantages that parallel devices currently have over serial EEPROMs is memory density and AC performance. However, in most microcontroller based applications for which Serial EEPROMs are intended, high density and AC are not the most critical design issues or most needed product features.

The key benefits of Serial EEPROM solutions as a result of the advantages outlined above, are reduced system costs, enhanced feature sets, and improved system performance. System size and weight is reduced and power sourcing requirements are kept at a minimum. The following graph compares some of the main attributes of a 16K Serial EEPROM device to a 16K Parallel device.



Serial EEPROM Solutions vs. Parallel Solutions

USES AND APPLICATIONS OF SERIAL EEPROMS

Uses of Serial EEPROMs

The days of simply being a DIP switch replacement for Serial EEPROMs is over. Here is a list of the functions that Serial EEPROMs perform in a variety of computer, industrial, telecommunication, automotive and consumer applications:

- 1) Memory storage of channel selectors or analog controls (volume, tone, etc.).
- 2) Power down storage and retrieval of events such as fault detection or error diagnostics.
- 3) Electronic real time event or maintenance log such as page counting.
- 4) Configuration storage.
- 5) Last number redial and speed dial storage.
- 6) User in-circuit look-up tables.

Serial EEPROM Applications

Serial EEPROMs have found homes in hundreds of embedded control applications in all major application markets. The following list demonstrates the number and variety of applications for serial EEPROMs.

Market

CONSUMER

Applications

TV tuners, VCRs, CD players, cameras, radios, and remote controls

COMPUTER/OA

Printers, copiers, PCs, palmtop and portable computers, disk drives and organizers

INDUSTRIAL

Bar code readers, point of sale terminals, smart cards, lock boxes, garage door openers, test measurement equipment and medical equipment

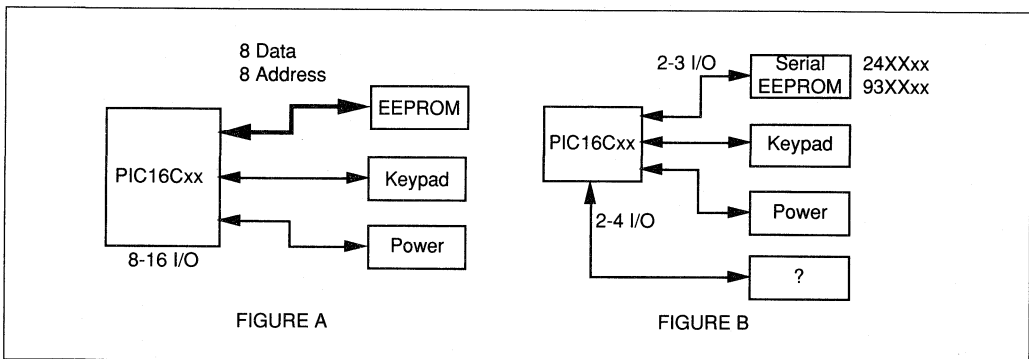
TELECOMM

Cellular, cordless and full feature phones, faxes, modems, pagers, and satellite receivers

AUTOMOTIVE

Air bags, anti-lock brakes, odometers, radios and keyless entry

Using Serial EEPROMs for critical data and configuration storage has only recently become a reality. The current offerings of 2 and 3 wire serial devices offers the systems designer interesting alternatives to the standard parallel EEPROM devices. The Serial EEPROM is basically a standard EEPROM array without the normal parallel data and address I/O. These functions are handled via serial I/O ports coupled with internal self-timed state machines. Not only will the serial device



Serial EEPROM Solutions vs. Parallel Solutions

save power, board space, and cost, but they also offer the advantage of fewer I/O and consequently power in the embedded microcontroller because less I/O are needed to control the same functions. A typical embedded application is shown in FIGURE A, depicting a controller and several functions used in a personal communications device, such as a mobile or portable phone. The EEPROM stores speed dial and last number redial numbers, credit card numbers, ID numbers, and configuration parameters.

FIGURE B shows these same functions using a controller with fewer I/O and a Serial EEPROM. There is no loss of functionality but a significant savings in current, board space, I/O pads, and cost. The serial solution employs 8 to 16 less I/O on the microcontroller, freeing up much needed functionality, and possibly allowing for a much smaller device package and downsized circuit boards.

SUMMARY

Serial EEPROMs are ideal cost effective solutions to all non-volatile memory embedded control applications that require: 1) A small footprint space saving format, 2) The ability and ease of programming one byte at a time, 3) Low current consumption and low operating voltage, 4) Low microcontroller overhead and support and, 5) The best price performance non-volatile memory solution available.

Their size, ease of programmability, low power consumption, and low cost make Serial EEPROMs extremely suitable for all the fast growing handheld and portable battery powered computer, personal communications, medical and industrial markets.

*Author: Tom Tyson
Memory Products Division*

SECTION 7 DEVELOPMENT TOOLS

Introduction to Microchip Development Tools	7-	1
Application-Specific Standard Products Division:		
PICSEE™ Tools Product Brief	7-	3
Logic Product Division:		
MPALC Cross Assembler Product Brief	7-	5
MPASM Universal Assembler/Linker Product Brief	7-	7
MPSIM Simulator Product Brief	7-	9
PICMASTER™-16X System Product Brief	7-	11
PICMASTER™-17 System Product Brief	7-	15
PRO MASTER™ Product Brief	7-	19
PICSTART™-16B Product Brief	7-	21
Memory Products Division:		
Total Endurance™ Predictive Software Model	7-	23
Serial EEPROM Evaluation Board	7-	25
Microchip Bulletin Board Service (BBS)	7-	27



Embedded Control Handbook

Introduction to Development Tools

PIC16/17 MICROCONTROLLER DEVELOPMENT SUPPORT

Look to Microchip to bring you the most complete set of development tools available for the entire PIC16/17 Microcontroller product line. There are a host of configurations available to suit all of your development needs. All of our products are PC-hosted systems allowing ease of use and maximum flexibility. Our products are available as bundled systems or individual components, which ever fits your development needs.

PIC16C5X SUPPORT PRODUCTS

PICMASTER-16X

PIC16CXX PC-Based High Performance In-Circuit Emulator System

PICMASTER Universal Emulator Pod, PIC16CXX Emulator Probe, PC (ISA) Interface Board, Emulation Control Software, Demonstration Board, PRO MASTER, MPALC Assembler, MPSIM Simulator, Power Supply, Cables, and Documentation.

PICSTART-16B

Development Programmer for the PIC16C5X, PIC16C71, and PIC16C84

Includes MPALC Assembler, MPSIM Simulator, and Development Programmer Board.

MPALC

PIC16C5X/PIC16CXX PC-Based Assembler Program

Assembler Software on 3.5" diskette and User's Manual.

MPASM

PIC16C5X/XX/17CXX PC-Based Assembler Program

Assembler Software on 3.5" diskette and User's Manual.

MPSIM

PIC16C5X PC-Based Simulator Program

Simulator Software on 5.25" diskette and User's Manual.

PIC17CXX SUPPORT PRODUCTS

PICMASTER-17

PIC17CXX PC-Based High Performance In-Circuit Emulator System

PICMASTER Universal Emulator Pod, PIC17CXX Emulator Probe, PC (ISA) Interface Board, Emulation Control Software, PRO MASTER Programmer, Power Supply, Cables, and Documentation.

UNIVERSAL SUPPORT PRODUCTS

PRO MASTER Universal Programmer

Unit for Microchip programmable devices including PIC16CXX, PIC17CXX, SE², and foreseeable new products. Includes Programmer Unit, Universal AC Power Supply, Manual, MS/DOS Programmer Software and Cables.

Introduction

NOTES:



PICSEE Product Development Systems

INTRODUCTION

The PICSEE Development Systems provide the product development engineer with cost effective and timely design tool solutions for the MTA8XXXX family of multichip modules. They are designed specifically for the MTA8XXXX family. These tools work in conjunction with existing hardware and software design tools for the PIC16/17 microcontroller family. This allows the development engineer to efficiently implement systems utilizing these multichip modules with a minimal learning curve and capital investment.

PICSEKIT — P/N AC812001

- Supports MTA81010
- Programming Adapters for PDIP and SOIC packages
- Daughter card for PICPROBE-16A
- I²C™ bus Serial Communication Application Software

This kit supports the MTA81010 multichip module. It contains programming adapters, a PICMASTER™ emulator daughter board and MTA81010 product samples in 28-lead PDIP. Also included is an MSDOS, PC-compatible 3.5-inch software diskette that contains example source code for implementing the I²C serial bus protocol to communicate with a Serial EEPROM. Documentation is provided for all of the included hardware and software.

Programming Support

Two programming adapters are provided to allow the MTA81010's internal program EPROM as well as its data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip's PIC16C54 can program the MTA81010's internal EPROM. Also, any programmer that supports Microchip's 24LC01B Serial EEPROM can program the MTA81010's internal Serial EEPROM. There is one adapter for MTA81010's in DIP packages and another for SOIC packages. Both DIP and SOIC programming adapters interface to programmers via a 300 mil DIP header.

Emulation Support

The emulator daughter board allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA81010 Microcontroller with Serial EEPROM. This daughter board replaces Microchip's PIC16C5X Emulator Probe Header (P/N AC162009) emulator probe to support the MTA81010. The daughter board provides the required translation from a PIC16C54 pin out to the MTA81010 pin out. It also contains a discrete 24LC01B Serial EEPROM to provide the same functions as the MTA81010's internal EEPROM. This provides a cost-effective emulation solution to customers who may wish to purchase a PICMASTER in-circuit emulator or those that already have a PICMASTER.

Software Support

Example source code for I²C Bus communication with a serial EEPROM is included in the PICSEE. This pre-tested code can be used directly or modified by the developer to meet their specific needs. This example code is provided royalty free and license free.

PICSEESTART — P/N DV813001

- Complete Low Cost Development Solution for MTA81010
- Combines PICSEKIT AC812001 and PICSTART DV163001
- MPALC Assembler
- MPSIM Simulator
- Low-Cost Programmer
- Programming Adapter Sockets
- I²C Bus Applications Software

This kit combines the PICSEKIT (P/N AC812001) with a PICSTART™ (P/N DV163001) to form a complete low-cost development system for the MTA81010 multichip module. It is designed to support the MTA81010 during the software development and initial prototype phases of new product development. It contains tools for software development and debugging, as well as programmer for programming the MTA81010's internal EPROM program memory. For a more detailed description, please refer to the PICSEKIT P/N AC812001 and PICSTART P/N DV163001 product descriptions.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
AC812001	PICSEKIT FOR MTA81010
DV813001	PICSEESTART FOR MTA81010



Microchip

MPALC Cross Assembler Product Brief

PIC16CXX Microcontroller Cross Assembler Software

This product brief describes the technical aspects of the Microchip Assembler (MPALC) developed by Microchip Technology Incorporated.

The MPALC Cross Assembler is a PC hosted symbolic assembler. It supports the PIC16C5X and PIC16CXX CMOS microcontroller series.

MPALC offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPALC REQUIREMENTS

MPALC will run on any IBM PC/XT™, AT™ or compatible computer, running DOS 3.31 or later. The distribution media is 3 1/2", low density (720k) floppies. It is distributed at the root level, and may be executed directly from the floppy.

No special display or ancillary devices are required.

MPALC ASSEMBLER FEATURES

MPALC supports the 12 bit PIC16C5X and the 14 bit PIC16CXX cores. The 12 bit core has 33 instructions, the 14 bit core has 35.

All instructions in both instruction sets are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPALC has the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for PIC16C5X and PIC16CXX Microchip micro-controllers.
- Macro Assembly capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Output formats: INHX8S, INHX8M, INHX16, and relocatable objects.
- Supports Hex (default), Decimal and Octal source and listing formats.

MPALC DIRECTIVE LANGUAGE

MPALC provides a full featured directive language represented by the following four classes of directives:

- **Data Directives** are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- **Listing Directives** control the MPALC listing format. They allow the specification of titles, subtitles, page ejects and other listing controls.
- **Control Directives** permit sections of conditionally assembled code.
- **Macro Directives** control the execution and data allocation within macro body definitions.

MPALC INSTRUCTION SET

MPALC supports the entire instruction set of the PIC16C5X and PIC16CXX micro-controllers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Control Operations

The Microchip microcontroller instruction set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- Two data transfer operations
- Six arithmetic operations (the PIC16CXX series provides two more)
- Six logical operations
- Three rotate operation

MPALC provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The PIC16C5X and PIC16CXX instruction sets allow read and write of special function registers such as the PC and status registers.

NOTES:



Microchip MPASM Universal Assembler Product Brief

Microchip Universal Microcontroller Assembler Software

This product brief describes the technical aspects of the PIC16/17 Assembler developed by Byte Craft Limited and distributed by Microchip Technology. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC16C5X CMOS, PIC16CXX, and PIC17CXX families.

MPASM offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from the Microchip Universal Emulator System (PICMASTER).

MPASM REQUIREMENTS

MPASM will run on any IBM PC/XT™, AT™ or compatible computer running DOS 4.1 or later. The distribution media is 3 1/2", low density (720K) floppies. It is distributed at the root level, and may be executed directly from the floppy.

No special display or ancillary devices are required.

MPASM ASSEMBLER FEATURES

MPASM supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions in both instruction sets are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPASM have the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro Assembly Capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.

MPASM DIRECTIVE LANGUAGE

MPASM provides a full featured directive language represented by four basic classes of directives:

- **Data Directives** are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- **Listing Directives** control the MPASM listing display. They allow the specification of titles and subtitles, page ejects and other listing control.
- **Control Directives** permit sections of conditionally assembled code.
- **Macro Directives** control the execution and data allocation within macro body definitions.

MPASM INSTRUCTION SET

MPASM supports the entire instruction set of the PIC16C5X, PIC16CXX and PIC17CXX microcontrollers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Special Control Operations

The Microchip microcontroller set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- Data Transfer Operations
- Logical Operations
- Rotate Operations

MPASM provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The Microchip microcontroller instruction sets allow read and write of special function registers such as the PC and status registers.

MPASM Universal Assembler Product Brief

MPLINK

MPLINK supports the linking of multiple relocatable objects created by MPASM into a single absolute hex file, suitable for simulating, emulating, and programming. MPLINK supports the hex outputs supported by MPASM and absolute listing file.

MPLIB

MPLIB provides the ability to group several relocatable objects into a logical collection, in one file. Libraries created by MPLIB can be referenced by MPASM. Only those objects required by the linking phase are included in the resulting hex output.



Microchip

MPSIM Simulator Product Brief

PIC16C5X and PIC16CXX Microcontroller Simulator

MPSIM is a discrete event simulator software application designed to imitate operation of the PIC16C5X and PIC16CXX microcontrollers. It allows the user to debug software that will use any of these micro-controllers.

At any instruction boundary, you may examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. MPSIM gives you a solid, low cost, source-level debug tool to help you through the early design verification stages of your project.

MPSIM REQUIREMENTS

MPSIM requires an IBM PC/XT™, AT™ or compatible computer running DOS version 3.31 or later. The PC needs a 3 1/2" floppy disk drive and at least 256K of main memory; MPSIM.EXE occupies roughly 150K. Recommended is a hard disk drive with 5 Mb of available storage.

MPSIM SIMULATOR

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging Microchip microcontroller assembler code.

The simulator is meant for use with smaller projects not requiring precise, more extensive development equipment. Since the PIC16C5X architecture is essentially a single tasking micro-controller without interrupts, many applications can be developed by using a simulator program alone.

The PIC16CXX family supports various peripherals and interrupt strategies. These interrupts can be simulated but certain peripheral functions (such as A/D conversions) are not.

The MPSIM Simulator has the following features to assist in the debugging of software / firmware for the user:

Program Load / Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display and Alter

Provisions are made to display and alter Program Memory, Register Files and status register bits. Also, simulator information such as cycle times, elapsed time, and step count can be displayed.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use and ASCII data.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, a number of items can be traced. Address ranges, registers and register contents and others.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

Assembler Support

MPSIM supports both the Microchip MPALC and the MPASM Universal Assembler.

MPSIM Simulator Product Brief

NOTES:



PICMASTER™ -16X System

PICMASTER Universal In-Circuit Emulator System



SYSTEM FEATURES

General:

- Complete Hi-Performance PC-based Microcontroller Development System for the PIC16CXX family.
- For use on PC-compatible 286, 386, and 486 machines under Microsoft Windows® 3.X environment.
- Assembler Software, Emulator System, and EPROM Programmer unit, sample kit, and demonstration hardware and software provide a complete microcontroller product development environment.

Emulator System:

- Hi-Performance In-Circuit Emulation of Microchip Microcontrollers.
- Real-time instruction emulation.
- Single and Multiple instruction step execution.
- Program Memory emulation and memory mapping capability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory.

- Real-time trace memory capture of 40 bits of information for each instruction cycle in an 8Kx40 trace buffer. Trace region can range from 0 to 64K in any address combinations.
- Real-time trace data can be captured and displayed without halting emulation.
- Unlimited number of hardware breakpoints can be set anywhere in the program memory.
- External Break with "AND"/"OR" capability with internal breakpoints.
- Multiprocessor emulation capability. Up to eight PICMASTER emulators can be synchronized on a single PC, for multi-processor development.
- Extended 48-bit cycle counter.
- Trigger Output available on any range of addresses.
- Full Symbolic Debug Capability. Symbolic display and alter of all register files, special purpose registers, stack registers, and bank registers.
- Selectable Internal Emulator Clock or User Target (Prototype) System Clock.
- User selectable internal or external Power Supply (provided).

PICMASTER-16X Development System

EPROM Programmer System:

- PRO MASTER™ Device Programmer unit for all current PIC16C5X and PIC16CXX products.
- Operates as a Stand-alone Unit or in Conjunction with a PC-compatible host system.
- Performs READ, PROGRAM, and VERIFY functions in Stand-alone mode.
- PC Host Software provides file display and editing, file transfer to and from programmer unit, device serialization, and program voltage calibration.

Macro Assembler:

- Provides translation of Assembler source code to object code for the PIC16C5X and PIC16CXX family of microcontrollers.
- Macro-assembly and conditional assembly capability.
- Produces Object files, Listing files, Symbol files, and special files required for symbolic debug with the PICMASTER Emulator System.
- Binary / Hex output formats: INHX8S, INHX8M, INHX16, and PICMASTER.

SYSTEM DESCRIPTION

The PICMASTER Universal In-Circuit Emulator System is intended to provide the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PIC16CXX. The PICMASTER system currently supports the PIC16C54, PIC16C55, PIC16C56 and PIC16C57 at clock frequencies of 4 MHz and PIC16C71, PIC16C84 to 10 MHz.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

The Emulator System is designed to operate on low-cost PC-compatible machines ranging from 286-AT class ISA-bus systems through the new 486 EISA-bus machines. The development software runs in the Microsoft Windows 3.X environment, allowing the operator access to a wide range of supporting software and accessories.

Provided with the PICMASTER System is a high performance real-time In-Circuit Emulator, a microcontroller programmer unit, a macro assembler program, and a simulator program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC16CXX microcontroller line.

Coupled with the user's choice of text editor, the system is ready for development of products containing any of Microchip's microcontroller products.

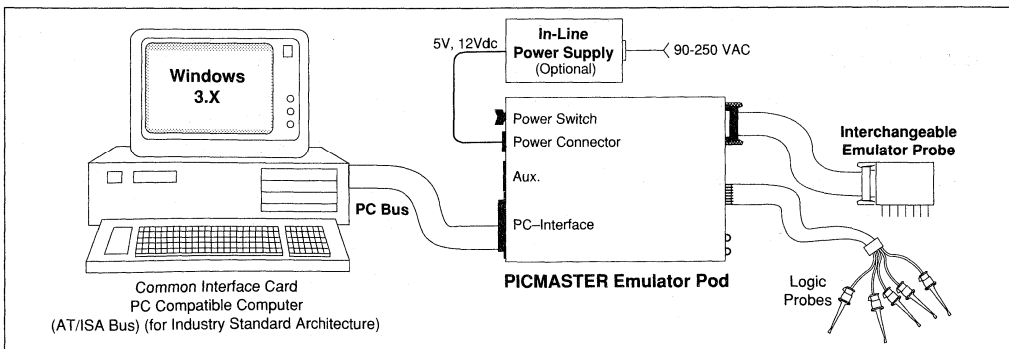
A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is included for additional convenience.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS system.

Host System Requirements:

The PICMASTER has been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The AT platform and Windows 3.X environment was chosen to best make these features available to you the end user. To properly take advantages of these features, PICMASTER requires installation on a system having the following minimum configuration:

- PC/AT-compatible machine: 286, 386SX, 386DX, or 486 with ISA or EISA Bus.
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MSDOS / PCDOS version 3.1 or greater.
- Microsoft Windows version 3.0 or greater operating in either standard or 386 enhanced mode).
- 1 Mbyte RAM (2 Mbytes recommended).
- One 5.25" floppy disk drive.
- Approximately 10 Mbytes of hard disk (1 Mbyte required for PICMASTER, remainder for Windows 3.X system).
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)
- Microsoft mouse or compatible (highly recommended).



Emulator System Components:

The PICMASTER Emulator Universal System consists primarily of four major components:

- Host-Interface Card:** The PC Host Interface Card connects the emulator system to a PC compatible system. This high-speed parallel interface requires a single half-size standard AT / ISA slot in the host system. A 37-conductor cable connects the interface card to the external Emulator Control Pod.
- Emulator Control Pod:** The Emulator Control Pod contains all emulation and control logic common to all microcontroller devices. Emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic are contained here. The Pod controls and interfaces to an interchangeable target-specific emulator probe via a 14" precision ribbon cable.
- Target-specific Emulator Probe:** A probe specific to microcontroller family to be emulated is installed on the ribbon cable coming from the control pod. This probe configures the universal system for emulation of a specific microcontroller. Currently, the PIC16C5X family, PIC16C71, PIC16C84, and the PIC17C42 microcontrollers are supported. Future microcontroller probes will be available as they are released.
- PC Host Emulation Control Software:** Host software necessary to control and provide a working user interface is the last major component of the system. The emulation software runs in the Windows 3.X environment, and provides the user with full display, alter, and control of the system under emulation. The Control Software is also universal to all microcontroller families.

The Windows 3.X System is a multitasking operating system which will allow the developer to take full advantage of the many powerful features and functions of the PICMASTER system.

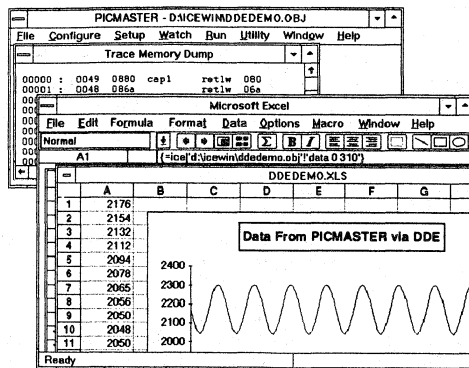
PICMASTER emulation can operate in one window, while a text editor is running in a second window. Dynamic Data Exchange (DDE), a feature of Windows 3.X, will be available in this and future versions of the software. DDE allows data to be dynamically transferred between two or more Windows programs. With this feature, data collected with PICMASTER can be automatically transferred to a spreadsheet or database program for further analysis.

Under Windows 3.X, up to eight PICMASTER emulators can run simultaneously on the same PC making development of multi-microcontroller systems possible (e.g., a system containing a PIC16CXX processor and a PIC17CXX processor).

PRO MASTER Device Programmer:

The PRO MASTER Programmer system included in the PICMASTER Development System provides the product developer with the ability to program (transfer) the developer's software into PIC16CXX microcontrollers.

The programmer unit comes complete with accessories for use with a PC host computer. Supplied are interface cables and connectors to a standard PC serial port, a power supply unit, and host operating software.



The PRO MASTER Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.

STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

When the PRO MASTER is connected to a host PC system, many more options and conveniences are available to the user. Host mode allows full interactive control over the PRO MASTER unit. A full screen, user-friendly software program is provided to fully assist the user.

As in stand-alone mode, parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified. In addition, other features available in host-mode are:

PICMASTER-16X Development System

Editing

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program and fuse data can be loaded and saved to DOS disk files. Files generated by the Assembler program are directly loadable into programmer memory.

VDD and VPP Adjust

The programming environment voltage settings of VDD max, VDD min, and VPP can be set and altered only on PC host mode. The voltage settings allow the user to program the part in the environment that the part will be used. The part will be programmed at VDD max and verified at VDD min. VPP is the programming voltage.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the factory or the listed sales offices.

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
EM167007	Complete PICMASTER-16A System for PIC16C5X
EM167010	Complete PICMASTER-16A System for PIC16C5X without Programmer
EM167011	Complete PICMASTER-16B System for PIC16C71
EM167012	Complete PICMASTER-16B System for PIC16C71 without Programmer
EM167013	Complete PICMASTER-16C System for PIC16C84
EM167014	Complete PICMASTER-16C System for PIC16C84 without Programmer
EM167017	Complete PICMASTER-16E System for PIC16C64
EM167018	Complete PICMASTER-16E System for PIC16C64 without Programmer



PICMASTER™ -17 System

PICMASTER PIC17CXX In-Circuit Emulator System

SYSTEM FEATURES

General:

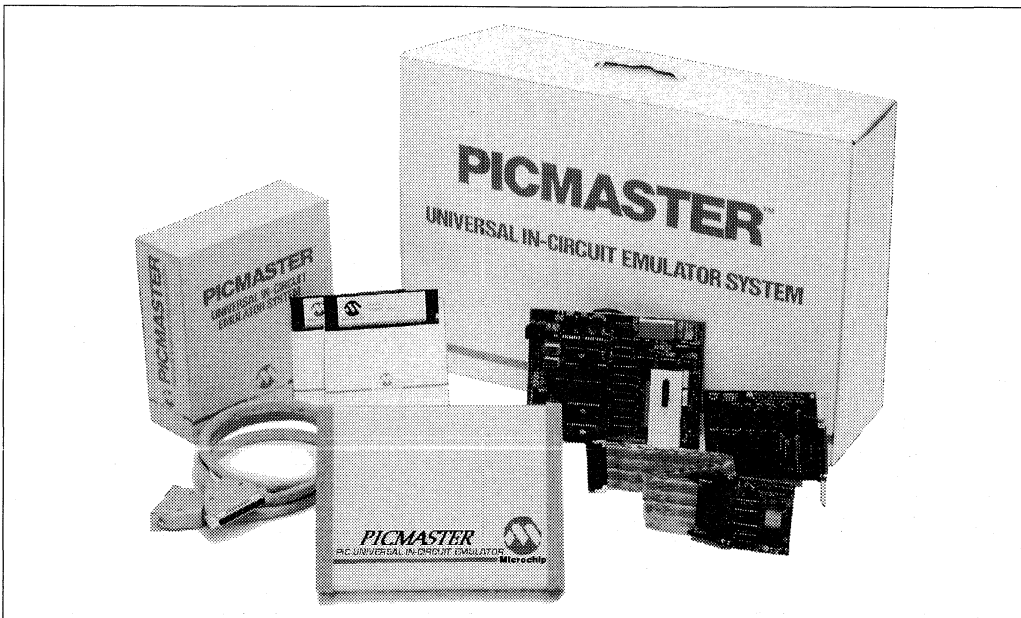
- The PICMASTER-17 Development System is designed by Microchip Technology Inc. and manufactured in the U.S.A.
- Complete Hi-Performance PC-based Microcontroller Development System for the PIC17CXX family.
- For use on PC-compatible 286, 386, and 486 machines under the Windows® 3.X environment.
- Assembler Software, Emulator System, and Programmer unit, sample kit, and software provide a complete microcontroller product development environment.

Emulator System:

- Universal In-Circuit Emulation pod supports emulation of PIC17CXX family. It can easily support other Microchip microcontroller products with the purchase of a low cost personality probe kit.
- Real-time emulation to 16 MHz.
- Single and Multiple instruction step execution.
- Program Memory emulation and memory mapping capability up to 64K words. Instruction execution can

be mapped into either emulation memory or user prototype memory.

- Real-time trace memory capture of 40 bits of information for each instruction cycle in an 8Kx40 trace buffer. Trace region can range from 0 to 64K in any address combinations.
- Real-time trace data can be captured and displayed without halting emulation.
- Unlimited number of hardware breakpoints can be set anywhere in the program memory.
- External Break with "AND"/"OR" capability with internal breakpoints.
- Multiprocessor emulation capability. Two or more PICMASTER emulators can be synchronized on a single PC for multi-processor development.
- Extended 48-bit cycle counter.
- Trigger Output available on any range of addresses.
- Full Symbolic Debug Capability. Symbolic display and alter of all register files, special purpose registers, stack, and bank registers.
- Selectable Internal Emulator Clock or User Target (Prototype) System Clock.
- User selectable internal or external Power Supply (provided).



PICMASTER-17 PIC17CXX In-Circuit Emulator

EPROM Programmer System:

- PRO MASTER™ EPROM Programmer unit for all Microchip PIC17CXX CMOS microcontrollers.
- Operates as a Stand-alone Unit or in Conjunction with a PC-compatible host system.
- Performs READ, PROGRAM, and VERIFY functions in Stand-alone mode.
- PC Host Software provides file display and editing, file transfer to and from programmer unit, device serialization, and program voltage calibration.

Macro Assembler:

- ASM-17 provides macro-assembly and conditional assembly capability.
- Provides translation of Assembler source code to object code for the PIC17CXX family of microcontrollers.
- Produces Object files, Listing files, Symbol files, and special files required for symbolic debug with the PICMASTER Emulator System.
- Binary / Hex output formats: INHX8S, INHX8M, INHX32.

SYSTEM DESCRIPTION

The PICMASTER Universal In-Circuit Emulator System provides the product development engineer with a complete microcontroller design toolset for all microcontrollers in the PIC16CXX and PIC17CXX families.

The PICMASTER-17 System is configured to support the PIC17C42 and related PIC17CXX family members.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

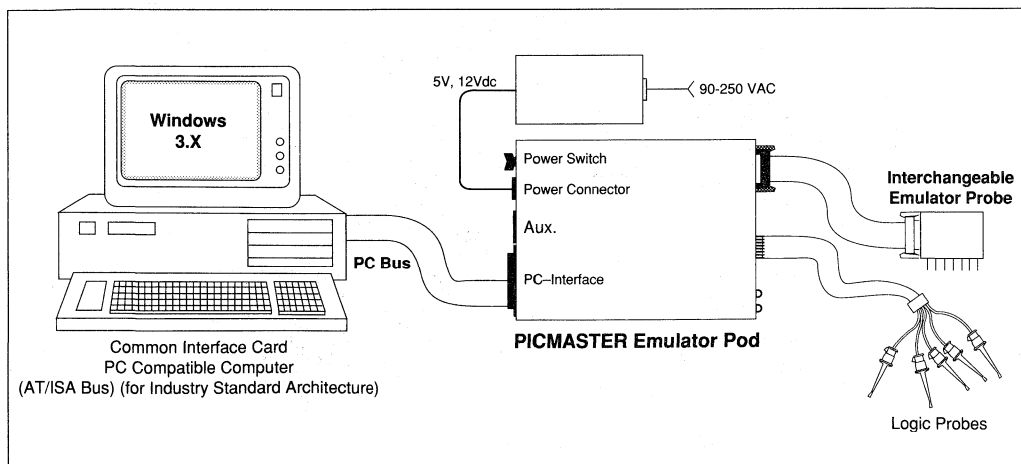
The Emulator System is designed to operate on low-cost PC-compatible machines ranging from 286-AT class ISA-bus systems through the new 486 EISA-bus machines. The development software runs in the Microsoft Windows 3.X environment, allowing the operator access to a wide range of supporting software and accessories.

Provided with the PICMASTER System is a high performance real-time In-Circuit Emulator, a microcontroller programmer unit, and a macro assembler program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC17CXX microcontroller line.

Coupled with the user's choice of text editor, the system is ready for development of products containing any of Microchip's microcontroller products.

A "Quick Start" PIC17CXX Product Sample Pak containing user programmable parts is included for additional convenience.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing Microchip Technology's Bulletin Board System (BBS).



PICMASTER-17 PIC17CXX In-Circuit Emulator

Host System Requirements:

The PICMASTER has been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The AT® platform and Windows 3.X environment was chosen to best make these features available to you the end user. To properly take advantages of these features, PICMASTER requires installation on a system having the following minimum configuration:

- PC/AT-compatible machine: 286, 386SX, 386DX, or 486 with ISA or EISA Bus
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MSDOS / PCDOS version 3.1 or greater.
- Microsoft Windows version 3.0 or greater operating in either standard or 386 enhanced mode).
- 1 Mbyte RAM (2 Mbytes recommended).
- One 5.25" floppy disk drive.
- Approximately 10 Mbytes of hard disk (1 Mbyte required for PICMASTER, remainder for Windows 3.0 system)
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)
- Microsoft mouse or compatible (highly recommended).

Emulator System Components:

The PICMASTER Emulator Universal System consists primarily of four major components:

- **Host-Interface Card:** The PC Host Interface Card connects the emulator system to a PC compatible system. This high-speed parallel interface requires a single half-size standard AT / ISA slot in the host system. A 37-conductor cable connects the interface card to the external Emulator Control Pod.
- **Emulator Control Pod:** The Emulator Control Pod contains all emulation and control logic common to all Microchip CMOS microcontroller devices. Emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic are contained here. The Pod controls and interfaces to an interchangeable target-specific emulator probe via a 14" precision ribbon cable.
- **Target-Specific Emulator Probe:** A probe specific to microcontroller family to be emulated is installed on the ribbon cable coming from the control pod. This probe configures the universal system for emulation of a specific microcontroller. Currently, the PIC16CXX family, and the PIC17C42 microcontrollers are supported. Future microcontroller probes will be available as they are released.
- **PC Host Emulation Control Software:** Host software necessary to control and provide a working user interface is the last major component of the system. The emulation software runs in the Windows 3.X environment, and provides the user with full display, alter, and control of the system under emulation. The Control Software is also universal to all microcontroller families.

The Windows 3.X System is a multitasking operating system which allows the developer to take full advantage of the many powerful features and functions of the PICMASTER system.

PICMASTER emulation can operate in one window, while a text editor is running in a second window. PICMASTER supports the window feature Dynamic data Exchange (DDE). DDE allows data and commands to be dynamically transferred between two or more Windows programs. With this feature, data collected with PICMASTER can be automatically transferred to a spreadsheet or database program for further analysis.

Under Windows 3.X, two or more PICMASTER emulators can run simultaneously on the same PC making development of multi-microcontroller systems possible (e.g., a system containing a PIC16C5X controller and a PIC17CXX controller) or two or more of the same controller family.

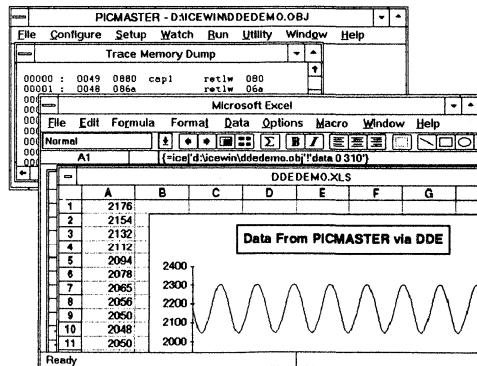
This allows data collected by PICMASTER to be automatically transferred to spreadsheets, data bases, or other analytic tools for further evaluation. DDE also allows automated control of PICMASTER which in turn allows development of automated test suites, life testing and production testers.

PRO MASTER EPROM Programmer:

The PRO MASTER Programmer system included in the PICMASTER Development System provides the product developer with the ability to program (transfer) the developer's software into PIC17CXX microcontrollers.

The programmer unit comes complete with accessories for use with a PC host computer. Supplied are interface cables and connectors to a standard PC serial port COM 1-4, power supply cable, and host operating software.

The PRO MASTER Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user as explained below.



PICMASTER-17 PIC17CXX In-Circuit Emulator

STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

When the PRO MASTER is connected to a host PC system, many more options and conveniences are available to the user such as serialized code programming. Host mode allows full interactive control over the PRO MASTER unit. A full screen, user-friendly software program is provided to assist the user.

As in stand-alone mode, parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified. Other features available in host-mode are:

Editing

A large screen buffer editing facility allows the user to change and program location in hexadecimal mode. Complete program and fuse data can be loaded and saved to DOS disk files. Files generated by the Assembler program are directly loadable into programmer memory.

VDD and VPP Adjust

The programming environment voltage settings of VDD max, VDD min, and VPP can be set and altered only on PC host mode. The voltage settings allow the user to program the part in the environment that the part will be used. The part will be programmed at VDD max and verified at VDD min. VPP is the programming voltage.

SALES AND SUPPORT - To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the factory or the listed sales offices.

PART NUMBER	DESCRIPTION
EM177001	PICMASTER PIC17CXX In-Circuit Emulator System
EM177004	PICMASTER-17 System without PRO MASTER Programmer



PRO MASTER™

CMOS Microcontroller Programmer Unit

SYSTEM FEATURES

EPROM Programmer System:

- PRO MASTER Programmer unit for the PIC16C5X, PIC16CXX, and PIC17CXX Microcontroller family.
- Operates as a Stand-alone Unit or in Conjunction with a PC Compatible host system.
- READS, PROGRAMS, and VERIFIES in Stand-alone mode.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit
- Communication Via RS-232

SYSTEM DESCRIPTION

PRO MASTER Programmer:

The PRO MASTER Programmer system provides the product developer with the ability to program user software into PIC16C5X, PIC16CXX, and PIC17CXX CMOS microcontrollers.

The programmer unit comes complete with accessories to be used with the PC host computer. Supplied are interface cables and connectors to a standard PC serial port, a universal input power supply unit, and host operating software.

The PRO MASTER Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.



CMOS Microcontroller Programmer Unit

STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

The PRO MASTER provides a very user friendly user interface which allows complete control over the programming session.

The PRO MASTER host software is a DOS windowed environment with full mouse support to allow the user to point and click when entering commands.

The Host Software communicates with the PRO MASTER via the serial port of the PC. Any of the four (COM 1-4) ports may be used. The communication is done at 19200 baud to insure fast throughput. Communication will be established with the PRO MASTER Device Programmer prior to any transfers taking place.

Serialization is done by generating a serialization file, and then using that file to serialize locations in the PIC microcontroller. Once a serialization file is generated, it may be used over different programming sessions. Serial numbers are automatically marked as used when a PIC is programmed successfully with that serial number.

Complete control over the programming environment is also provided. Control over the programming and verify voltage of Vdd insures that the Microcontroller will perform in the desired environment. Programming (Vpp) voltage is also adjustable to insure complete compatibility with future programming algorithms.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PROGRAMMER PART NUMBER	DESCRIPTION
PG007001	Programmer Kit as described above
PG007002	Programmer Kit without power supply

SOCKET PART NUMBER	DESCRIPTION
AC164001	PIC16C54 thru C57 18 & 28 Lead PDIP Socket Module
AC164002	PIC16C54 thru C57 18 & 28 Lead SOIC Socket Module
AC164003	PIC16C54/56 20 lead SSOP Socket Adapter
AC164010	PIC16C71/84 18 Lead PDIP/SOIC Socket Module
AC164011	PIC16C55/57 28 Lead SSOP Socket Adapter
AC164012	PIC16C64 40 PIN DIP Socket Module
AC164013	PIC16C64 44 PIN PLCC Socket Module
AC164014	PIC16C64 44 PIN PQFP Socket Adapter
AC174001	PIC17C42 40 Lead PDIP Socket Module
AC174002	PIC17C42 44 Lead PLCC Socket Module
AC174003	PIC17C42 44 Lead QFP Socket Module (future release)

Socket modules are sold separately.



PICSTART™ -16B

PIC16CXX Low-Cost Microcontroller Development System

SYSTEM FEATURES

EPROM Programmer System:

- EPROM Programmer unit for the PIC16C5X and PIC16CXX Microcontroller family. Supports PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC16C84.
- Operates with a PC-compatible host system.
- READS, PROGRAMS, and VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16C5X and PIC16CXX microcontroller product family.
- Macro-Assembly capability.

- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
- Output formats: INHX8S, INHX8M and INHX16.

Simulator:

- Instruction-level Simulator of the PIC16CXX microcontroller product family.
- For PC-compatible systems running the MSDOS operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

"Quick Start" Sample Kit:

- Provides the User / Developer with a sample kit of PIC16CXX parts for initial prototype use.



PIC16CXX PICSTART System

SYSTEM DESCRIPTION

The PICSTART-16B Development System provides the product development engineer with an alternative low-cost introductory microcontroller design tool set for the PIC16CXX family where full real-time emulation is not required. The equipment in the PICSTART-16B system operates on any PC compatible machine running the MSDOS/PCDOS operating system.

Provided in the System is an MSDOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPALC).

Sample software programs to be run on the simulator are provided to help the user to quickly become familiar with the development system and the PIC16CXX microcontroller line.

The user need only provide his or her own preferred text editor and the system is ready for development of end products using the PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, or PIC16C84 microcontrollers.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is also included.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS.

PICSTART-16B Development Programmer:

The Microchip device programmer system included in the PICSTART-16B Development System provides the product developer with the ability to program user software into PIC16CXX EPROM microcontrollers. It is designed to be a development programmer and not recommended for use in a production environment.

The programmer unit connects to a standard PC serial port.

A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified.

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program data can be loaded and saved to DOS disk files. Files generated by the MPALC Assembler program are directly loadable into programmer memory.

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16CXX assembler code.

The simulator is meant for use with smaller projects not requiring precise more extensive development equipment. Since the PIC16CXX architecture is essentially a single tasking microcontroller without interrupts, many applications can be developed by using a simulator program alone.

The MPSIM Simulator has the following features to assist in the debugging of software/firmware for the user.

Program Load/Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display & Alter

Provisions are made to display and alter Program Memory, Register Files, and status register bits. Also simulator information such as cycle times, elapsed time, and step count can be displayed.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use, and ASCII data.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, address ranges, registers, register contents, and others can be traced.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER
DV163001

DESCRIPTION
PICSTART-16B DEVELOPMENT SYSTEM



TOTAL ENDURANCE™

Total Endurance™ Predictive Software Model

INTRODUCTION

Microchip's Total Endurance Disk provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. Now designers can describe their system to an advanced mathematical model (with a standard Windows™ interface) which will then predict the performance and reliability of the Serial EEPROM within that environment. Design trade-off analysis that formerly consumed hours, days or weeks can now be accomplished in minutes - with a level of accuracy that delivers a truly robust design.

Users may control the following parameters:

- Serial EEPROM device type
- Bytes per cycle
- Cycling mode - byte or block
- Data patterns type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application life or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level versus the number of cycles. The model is available in both DOS and Windows™ versions and offers the following additional features.

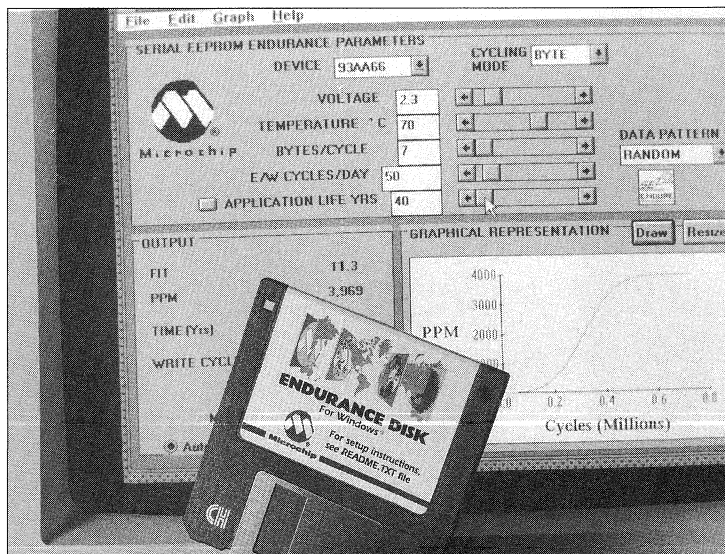
FEATURES

- IBM® PC compatibility
- Automatic or manual recalculation
- Full-screen or windowed graph view
- Hypertext on-screen help
- Key entry or slide-bar entry of parameters
- Support of Microchip's 2- and 3-wire Serial EEPROMs
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk endurance tutorial
- Real-time update of data

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- 386 or 486 processor recommended
- Math coprocessor recommended
- Windows 3.1
- 1 MB memory

Eliminate time and guesswork from your next Serial EEPROM design. Contact your local Microchip representative today and ask for the Total Endurance Disk.



Total Endurance™ Predictive Software Model

NOTES:



Serial EEPROM Evaluation Board

INTRODUCTION

The Microchip Serial EEPROM Evaluation Board (SEEVAL™) provides system design engineers with a very efficient and economical way to evaluate and program Microchip Serial EEPROMs. The board is designed for serial (RS232) connection to most IBM®-compatible PCs, and its powerful software interface runs under Microsoft® Windows™ 3.1. Graphical representation of the EEPROM array provides fast, easy access and control of data.

Application software provided with the Serial EEPROM Evaluation Board allows the user to program special features of advanced devices such as the 24C65 Smart Serial™ and 93LCS66 write-protectable serial. It also allows the user to fill the EEPROM array using any of several methods:

1. Download data from a binary file.
2. Fill with any user-defined repeating pattern.
3. Fill with "checkerboard" or "inverse checkerboard" patterns.
4. Read or write any portion of the array down to 1 byte in size.
5. Test header for scope connections.
6. Full Read/Write cycling functions

checkerboard:	5555	inverse	checkerboard:	AAAA
	AAAA			5555
	5555			AAAA
	AAAA			5555

EEPROM data can also be saved to a file on disk.

Software and system debug time and overall time-to-market can be reduced by using the Serial EEPROM Evaluation Board to write known data into the array in advance or to verify data which was stored by the application itself. The package lends itself very well to quick-turn changes and rapid verification during software development and system integration.

Instructions for use and a power-supply cable are included in the Microchip Serial EEPROM Evaluation Board Kit.

Serial EEPROM Evaluation Board

NOTES:



MICROCHIP BBS

Microchip Bulletin Board Service

Get current information and help on Microchip's Bulletin Board Service (BBS)! Microchip wants to provide you with the best responsive service possible. To accomplish this, the systems team monitors the BBS, posting the latest component data and software tool updates, providing technical help and embedded systems insights, and discussing how Microchip products provide project solutions. Extend your technical groups staff with microcontroller and memory experts through Microchip's BBS communication channel.

CONNECTING TO MICROCHIP

Connect worldwide to the Microchip BBS using the Compuserve communications network. In most cases, a local call is your only expense. The Microchip BBS connection does not use Compuserve membership services, therefore **you do not need Compuserve membership to join Microchip's BBS.**

The procedure to connect will vary slightly from country to country. Please check with your local Compuserve agent for details if you have a problem. Compuserve services allows multiple users at buad rates up to 9600. To connect:

1. Set your modem to 8 bit, No parity, and One stop (8N1). This is not the normal Compuserve setting which is 7E1.
2. Dial your local Compuserve phone number.
3. Type **<RETURN>** and a garbage string will appear because Compuserve is expecting a 7E1 setting.
4. Type **+<RETURN>** and Host Name: will appear.
5. Type **MCHPBBS<RETURN>** and you will be connect to the Microchip BBS.

To learn Compuserve's phone number closest to you, set your modem to 7E1, and dial (800) 848 8980, and follow Compuserve's directions. If you are dialing from overseas, you may call 614-457-1550 for voice information.

Connect without charge to the bulletin board. However, you are responsible for your phone charges. Access is available to all, but users are required to register the first time they "log in." No registration fees are required at this time.

USING THE BULLETIN BOARD

The Microchip Bulletin Board is a multi-faceted tool. Topic information includes:

- Special Interest Groups
- Files
- Mail
- Bug lists
- Technical assistance
- Consultant Directory

Special Interest Groups

Special Interest Groups, or SIGs, offer you the opportunity to discuss technical issues and topics with other users. Take advantage of the Microchip user community's broad background to glean information not available by any other method.

SIGs exists for most Microchip systems, including:

- PIC-SW
- PICMASTER
- PRO MASTER
- UTILITIES
- BUGS
- APP NOTE
- ENDURANCE
- MEMORY PRODUCTS

These groups are moderated by Microchip staff.

Files

The Microchip Systems BBS is used regularly to distribute bug reports, history files, and interim patches for Microchip software products.

Users can contribute files for distribution on the BBS. These files will be monitored, scanned, and approved or disapproved by the SIG moderator. No executable files are accepted from the user community in general.

Mail

The BBS can be used to distribute mail to other users of the service.

This is an excellent way to get questions answered by the Microchip staff, as well as to keep in touch with fellow Microchip product users worldwide.

The BBS is an evolving product intended to serve your needs. We welcome your ideas and input. Consider mailing a message to your SIG moderator, or to the SYSOP, if you have ideas or questions about particular Microchip products, or BBS operation.

Microchip's Bulletin Board

NOTES:

SECTION 8

ARTICLE REPRINTS FOR PIC16/17 MICROCONTROLLERS AND SERIAL EEPROMS

16C5X	Lean and Mean PIC Machines	8- 1
16C71	Enhanced PIC16Cxx Gets ADC and Interrupts	8- 7
16C84	Microchip PIC Adds EEPROM Data Memory	8- 8
17C42	Using the PIC Micro	8- 9
17C42	PIC17C42 Based DC Motor Control (Japanese)	8- 11
PICSTART-16B	Take your Pick with Controllers	8- 17
Serial EEPROM	Microchip Technology Rewrites the EEPROM	8- 18
Serial EEPROM	Serial EEPROM for Embedded Applications	8- 19
Endurance	A Tool for Calculating EEPROM Endurance	8- 23

Lean and Mean PIC Machines

One of the first things an aspiring marketer learns is to choose one's words carefully. As a junior chip peddler (er... sales consultant), I remember being told that "cheap" was a word to be avoided. Period. Pick your euphemism ("cost sensitive," "high volume," "cost over performance") but for heaven's sake, not "cheap"!

Along the same lines, simply stating your product's features is considered amateurish. You must instead relate the "benefits" that will accrue to those customers wise enough to choose you over competitors. With apologies to Woody Allen, the ultimate sales pitch is something like:

FEATURE: Super-duper See-Thru Look-ahead Pipe-dream

BENEFIT: you a) will be loved and
b) will never die.

Gee, where do I sign.

Maybe I'm burned-out from the ever-escalating hype permeating the high-tech biz, but now is time for a new era of glasnost marketing. I want to see a data sheet like:

- Faster than a bat out of hell!
- Easier to use than a politician!
- Cheap!
- Cheaper!
- Cheapest!

Here are some chips from Microchip Technology that fill the bill.

PIC AND CHOOSE

Let me just say up front, continuing with my no-bull approach, the PIC 16C5x series has what are probably the cheapest micros around. I'm talking as little as \$2 in volume for a complete OTP (One-Time Program; i.e., EPROM in a no-window package) micro with RAM, EPROM and I/O.

Many times, I have been faced with the need for a little piece of logic to perform some fairly mundane task many times. Usually, the choices boiled down to either wiring up a few 74xx TTLs or using one of the popular 80xx or 68xx micros.

The problem is often neither approach is ideal. The TTL-gates approach may be best for high-volume

applications (or for those who agree with our illustrious leader, Steve Ciarcia, when he says, "my favorite programming language is solder"), but wire-wrapping and debugging more than a few boards quickly becomes tiresome. Furthermore, you're faced with diving back into the rat's nest when you change your mind (inevitably) about how it should work.

The classic 8-bit micro may be the easiest approach, but it can be expensive. A "single-chip" micro was traditionally either ROM based (i.e., minimum feasible order size is thousands) or windowed EPROM (expensive; e.g., \$10 for an 8748). Another alternative is using a "non-single-chip" micro, with low-cost external RAM and EPROM, but then I'm back to the wire-wrapping blues (or more often than I care to admit, just throwing a \$100 single-board computer at what should be a \$5 application).

Admittedly, recent technologies have closed the gap. On the gate side, PALs can easily consolidate 5-10 74xxs. Meanwhile, the classic 8-bit to EPROM-based single-chip micros are beginning to appear in OTP versions at lower cost.

Despite this convergence on the needs of low cost and volume applications, these approaches always leave me feeling a little uncomfortable about the waste involved because I like to squeeze every bit of functionality out of a given technology. The gates approach is often speed overkill (i.e., you don't need a 50-MHz PAL to toggle an LED) while the micro approach is complexity overkill (e.g., an 8K-byte, 32-I/O-bit micro with 7.5K bytes of NOPs and 24 no connects). Read on to see how the PICs uniquely fill the gap between gates and regular 8-bit micros.

CHEAP RISC

Figure 1 shows the pin-outs for the two basic versions (1 8-pin and 28-pin) of the PIC. Right away the dual personality of the chip becomes apparent; it is an 8-bit micro in small gate-like package (especially the 18-pin DIP version). Why pay for more if you don't need to?

With so few pins, explaining their function becomes blessedly easy. Note the only I/O difference between the 18-pin and 28-pin versions is the latter has an additional 8-bit I/O port (RC0-RC7). In addition to the general-purpose I/O lines, MCLR* is a reset input while OSC1 and OSC2 are the CPU clock lines. The oscillator is unique because it can accept a simple RC (Resistor/Capacitor) as well as traditional crystal or TTL clock

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.

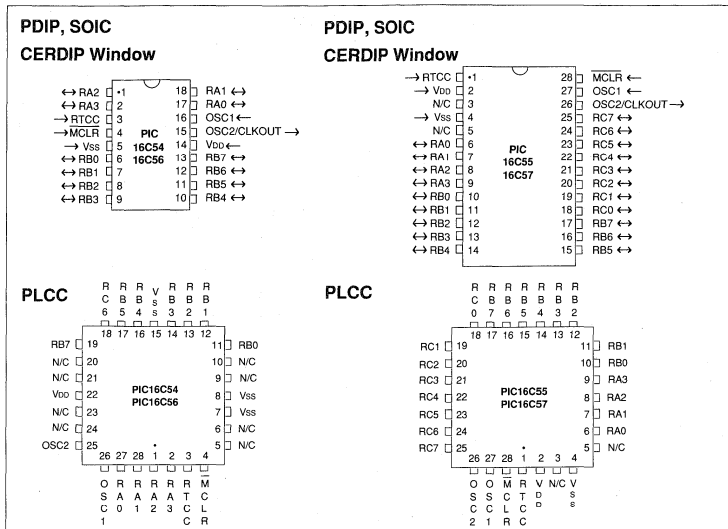


Figure 1—PIC processors are available in a number of configurations that fit a range of applications.

source. Save money by using the RC option if you're willing to accept less speed and accuracy as shown in Figure 2. RTCC is an input that can clock an on-chip 8-bit counter (with optional 1:2, 1:4, ..., 1:128 prescaler). Alternatively, the on-chip counter and prescaler can be driven by the internal clock divided by four (for example, 5 MHz for a 20-MHz PIC).

The RISCy nature of the PIC becomes clearer moving on-chip (Figure 3). Let me check off some of the traditional RISC criteria against the PIC.

- **Fixed Length Instructions:** Yep. Every PIC instruction is 12 bits, and different models offer 512 (16C54/55), 1K (16C56), or 2K (16C57) instruction capacity. Programming is easy because there are only 38 different instructions, which is about as reduced as possible.
- **Pipelined, Single-Cycle Execution:** OK, with the understanding that a single cycle is four clock periods (as shown in Figure 4). With a two-level pipeline (fetch and execute over-lapped), the PIC's "5-MIPS" performance (at 20 MHz) leaves most other 8-bit micros eating dust.
- **Load/Store:** The PIC fulfills this most fundamental of RISC precepts, subject to a little handwaving. All PIC instructions reference "W," a File Register (FR), or both. In one interpretation, W is a type of "accumulator" while FRs (the '54, '55, and '56 offer 32 while the '57 offers 80 FRs) are "RAM." In this light, the PIC isn't Load/Store because instructions can operate directly on "memory" (the FRs). However, if you consider W as a "temporary register" and FRs as "regular registers," the instructions only work on "registers" per the Load/

Store criteria. Actually, this limitation doesn't matter because Load/Store, which addresses the issue of "slow" external memory accesses versus "fast" on-chip register becomes a rather meaningless concept when all "memory" is on-chip.

- **Harvard Architecture:** this criterion refers to the use of separate bus and memories for instructions and data, which the PIC exploits. Though not really a RISC tenet, Harvard architecture is hyped as such on complex RISC chips like the AMD 29000 and Motorola 88k.

Of course, the PIC does not have cache (another meaningless concept when all memory is on-chip), delay slots, branch prediction, speculative execution, superscalar, superpipeline, or superanything. These omissions are not surprising because the PIC doesn't even have interrupts and it barely has a stack (two levels

Cext	Rext	Fosc @ 5.0V, 25°C	
20pF	5.1k	3.33 MHz	±22%
	10k	1.85 MHz	±22%
	100k	189 kHz	±38%
100pF	5.1k	1.18 MHz	±15%
	10k	668 kHz	±15%
	100k	67.8 kHz	±25%
300pF	5.1k	46 kHz	±10%
	10k	254 kHz	±13%
	100k	25.1 kHz	±21%

Figure 2—If lower speeds and less accuracy are acceptable tradeoffs for lower cost, a simple RC circuit may be used in place of a crystal.

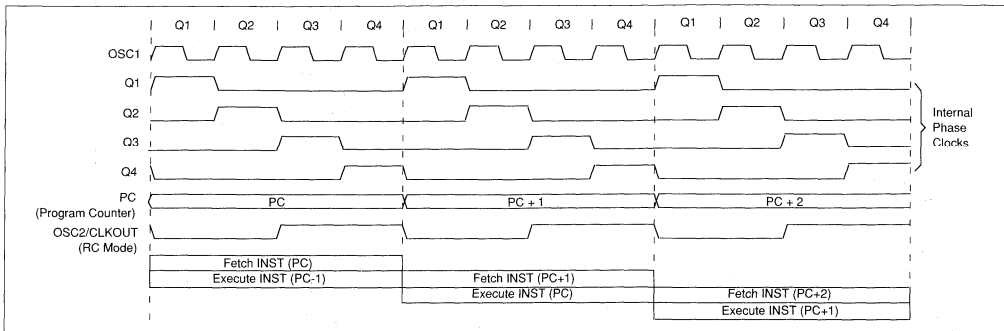


Figure 4—The PIC operates with a four-clock-period cycle and overlaps instruction execution with the fetch of the next instruction.

to an 18- or 28-pin DIP header on the other using a 6" ribbon cable, which plugs into the target PIC socket. Each board is powered by a small wall-mount transformer, so they can be used together or separately.

Entering a program on the PC using your favorite editor, and then assembling that program using PASM.EXE completes the developmental procedure. Next, you can use PEP.EXE to download the object code to the emulator, run the program, and after everything is working, burn a PIC using the device programmer. The latter includes LIF (low insertion force) sockets for both the 18- and 28-pin PICs.

A key point is the Parallax PASM redefines the instruction syntax, so it differs from the "official" definition by Microchip. Normally, such a move would be considered taboo, but I'm letting it pass for two reasons. First, I don't have to worry about maintaining compatibility with vast libraries of existing official PIC code (I'll bet you don't either). Second, and most important, in my opinion the Parallax scheme is superior.

The Parallax improvements generally fall into two classes: more consistent instruction names and formats, and macroinstructions in which a single PASM instruction generates multiple official PIC instructions.

These improvements effectively decrease the already reduced instruction set. For example, Parallax turns ten different official instructions into one generic MOV instruction. Furthermore, the Parallax macroinstructions can replace up to four official instructions.

Remember, brain cells start dying when you're in your twenties, and the population of programmers is rapidly aging. The Parallax approach postpones the day when we'll be forced to watch commercials with elderly hackers whimpering, "I've fallen—and I can't remember all the darn instructions!"

CHEAP THRILLS

The "no-bull" approach calls for less talk and more action, so let me discuss an example PIC application. Figure 5 shows the schematic for a lean, mean, four-channel digital oscilloscope based on the 28-pin 16C55-HS PIC, a 64K x 4 DRAM, and a resistor network configured as a voltage-dividing DAC. Thanks to the low price of the OTP PIC, the total parts cost for the gizmo is less than \$20—refreshing isn't it?

When the Record button is pressed, the PIC captures and stores 64K samples from the four input channels in the 64K x 4 DRAM. For display, the PIC drives the resistor ladder DAC in a manner that causes an attached oscilloscope to magically show the four input traces plus a "guide" trace reflecting the position of the scope screen (64 samples) in the trace buffer (1024 screens). The DAC implementation exploits the fact that PIC I/O lines are true CMOS, unlike those of most TTL-compatible micros (i.e., high impedance when configured as inputs and sink/source rail-to-rail when configured as outputs). The Left and Right buttons scroll the scope "screen" back and forth either a sample at a time (position designated by the small guide) or 16 screens (i.e., 1024 samples) at a time (position designated by the large guide).

The display loop starts with a sync pulse generated by driving the DAC from 0 to 255 (i.e., 5 volts). Next, the PIC runs through a loop 64 times, outputting the guide trace voltage and then a voltage for each of the four data traces. The latter are determined by adding a voltage representing each sample value (0 or 1) and a position offset voltage corresponding to the channel number.

Three sampling modes are provided for Record. If Record is pressed by itself, the PIC simply captures 64K

Listing 1—The Record portion of the PIC code shows how buttons are debounced and how the samples are taken and stored.

```

;
; Record pressed - get final record command and read 64K
; samples into DRAM
;
handle_record    call    clear_address    ;reset address
                clrb    bit              ;clr bit to flag left/right

:debounce       mov     temp,#100        ;100 cycles of debounce
:wait           sb      left             ;if left button. dir=0
                clrb    direction
                sb      right            ;if right button, dir=1
                setb   direction
                snb    left              ;if either button. set bit
                sb      right
                setb   bit
                jnb    record,:debounce  ;if record, reload debounce
                djnz   temp,:wait        ;record not pressed, 100 yet?

                mov     w,#101000b      ;get + or - edge for option
                snb    direction         ;according to direction
                mov     w,#111000b
                mov     option,w        ;load option with w

                mov     w,#regular_loop  ;if left or right button is
                snb    left              ;still pressed, then do
                sb      right            ;per-sample triggering
                mov     w,#trigger_loop

write_entry     mov     temp,w           ;save loop address in temp
                clrb    we              ;clr we to enable dram write
                jnb    bit,regular_loop  ;if no button, regular

trigger_loop    clr     rtcc             ;reset rtcc
:wait           jnb    record,clear_to_end ;if record button, abort
                jnb    rtcc.0,:wait      ;wait for trigger edge

regular_loop    mov     rc,address_low   ;2 cyc   ;write row address
                clrb    ras              ;1       ;latch into dram
                mov     rc,address_high  ;2       ;write column addr
                clrb    cas              ;1       ;latch in dram. write
                setb   cas               ;1       ;end write cycle
                setb   ras               ;1       ;return ras high, too
                mov     w,temp           ;get loop addr in w
                inc     address_low      ;1       ;inc low address
                snz    ;1/2s             ;skip if not 0
                incsz  address_high     ;1/2s    ;inc hi addr
                ; skip if 0
                jmp     w                ;2       ;jmp to loop
                ;(14 cycles/loop)

                setb   we               ;done, return we high
                jmp     make_ra_input    ;compensate for write_entry

```

increment the address and exit the loop when 64K samples are captured. Note how the final jump instruction loops back to either trigger_loop (for sample-per-trigger mode) or regular_loop, depending on the address in W (which is saved/ restored from variable temp). The core loop takes only 2.8 μ s to execute (14 cycles x 4 clocks/cycle x So ns/clock) thanks to the fact the PIC is running at 20 MHz, thus, it acquires the data at over 350 kHz. Not bad at all. In fact, it acquires data much faster than any number of more expensive micros that shall remain nameless (lucky for them).

Oh well, the overpriced competitors have one big advantage over the PIC—at least they aren't cheap.

Tom Cantrell holds a B.S. and an M.B.A. from UCLA. He owns and operates Microfuture Inc., and has been in Silicon Valley for ten years working on chip, board, and system design and marketing.

Enhanced PIC16Cxx gets ADC and interrupts

Eight-bit microcontrollers (μ Cs) are taking over the low-cost, down-in-the-dirt, embedded-system world. Low-cost small pinouts, critical peripherals, and easy design-in are requirements in this tough design environment. Microchip Technology's upgraded versions of its PIC 16Cxx 8-bit μ C line, the PIC 16C71, now has additional functions that include interrupt processing, an A/D converter, and high-current I/O for driving LEDs directly.

PIC μ Cs combine high processor-throughput rates with small pinouts (18 to 28 pins) and simple register-based architecture. PIC μ Cs evolved from an I/O controller for GE mainframes and have taken on distinct RISC-like features: a small number of fixed-length instructions and a pipelined operation (2-stage). Unlike RISC, PICs have an accumulator (working register) and handle dynamic data in registers, not RAM. The 16C71 adds significant power to the PIC 16Cxx family. The instruction word is lengthened from 12 to 14 bits, which makes control easier; interrupts have been added (earlier chips required continual polling); and a 4-channel A/D converter (20 μ sec) has been added, which eliminates the need for an off-chip converter.

The PIC 16C71 runs at 20 MHz and executes most instructions in one pipeline cycle-200 nsec. Branches take two cycles, or 400 nsec.

The PIC 16C71 has four interrupt sources and an 8-level hardware stack. Interrupts or subroutine calls automatically push the 13-bit program counter onto the

Microchip PIC 16C71 8-bit μ C

- 8-bit μ C, 20-MHz clock
- 35 14-bit instructions
- 200-nsec instruction execution, 2-stage pipeline
- 36-byte RAM
- 1k-word (14 bits) ROM
- 4-channel, 8-bit A/D converter (20 μ sec)
- Sleep mode with A/D wake-up
- 3 to 5V (2 mA at 3V, 4 MHz)
- \$3.25 (10,000) in 18-pin DIP or SOIC

stack and pop it off on return. (Earlier PICs had a 2-level stack that could easily overflow.)

SLEEP mode cuts power by turning off the oscillator; the A/D converter can be turned off, too. The converter does conversions while the CPU is in SLEEP mode, waking up the CPU with a conversion-complete interrupt.

A more powerful version of the PIC family, the 17C42, adds interrupts, a larger instruction set, external-memory capability, and a wider, 16-bit instruction word.

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.

Using the PIC Micro

Automotive engine carburetors and points have gone the way of vacuum tube radios and starter cranks, they have been made obsolete by advances in technology. Thanks in large part to recent advances made in cost effective single-chip microcontrollers, functions traditionally addressed by mechanical and electromechanical methods continue to be replaced by sophisticated embedded control systems. Increasing use of distributed intelligence has resulted in automobiles that provide better performance, more convenience features, a high degree of safety, better serviceability, and better long-term reliability. This article explores the advantages offered by Microchip's PIC microcontrollers used in this field.

KEYLESS ENTRY

Although the small rf transmitter for an automotive Keyless Entry system may appear simple from the outside, it actually requires the microcontroller inside to meet specific and rather stringent requirements. These requirements include very small physical size, extremely low current drain (for long multi-year battery life), and low system cost.

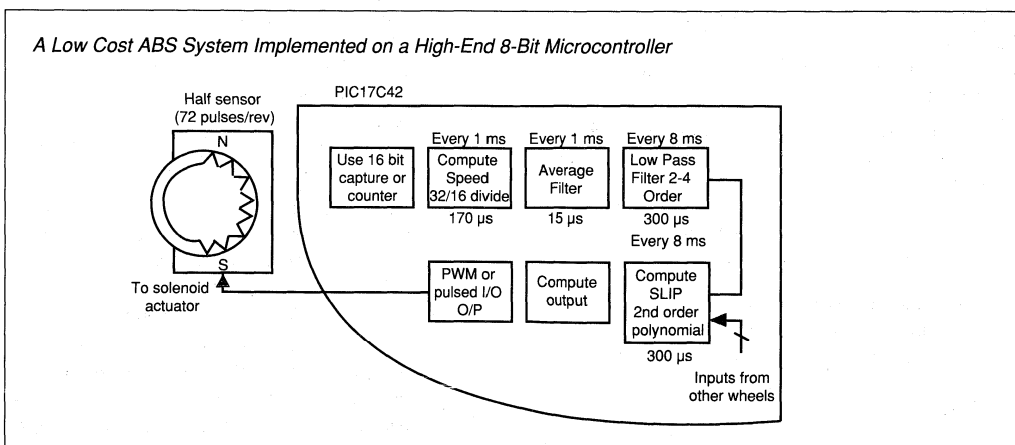
One microcontroller device that has proven very successful in Keyless Entry systems is Microchip Technology's 8-bit PIC16C54. It is available in the industry's smallest microcontroller package, the 18

lead ssop package. The operating current drain is less than 15 μ A at low operating frequency, and the standby current is less than 3 μ A. And in high volume, the price of the 8-bit PIC16C54 is designed to offer the lowest possible system cost.

An additional capability that further sets the PIC16C54 apart from competitive devices is called Serialised Quick Turn Programming (SQTP). SQTP is a unique programming service available from Microchip for the PIC16C54, and other members of the product family. With SQTP, the eeprom program memory is programmed by Microchip with the same identical software algorithm plus a security code unique to each specific device.

SQTP programming provides several important benefits to Keyless Entry systems and other security applications:

- 1) Due to the unique programmed security code each transmitter will work with only the specific automobile whose receiver has the identical security code. A SQTP programmed PIC16C54 (or larger member of the family) is normally used in the automobile receiver as well.
- 2) System cost is reduced since no external circuitry or switches are required to specify the security code.
- 3) Due to the excellent software code protection of the eeprom based microcontroller, potential car thieves would find it virtually impossible to read out the software algorithm or unique security code.



PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.

In addition to the Keyless Entry systems SQTP would be useful in other automotive security and Vehicle Identification systems as well. For example, high-end auto theft deterrent systems will use SQTP to program certain critical information that would be useful to help find and identify a stolen vehicle. Other future application plans being discussed include an Automatic Vehicle Identification system that would allow an automobile to transmit its vehicle ID number automatically to roadside tool booths and security check points. In both applications the automobile would transmit information over a radio channel.

IN-CIRCUIT PROGRAMMABILITY

In some automotive applications it would be very useful if microcontrollers could be programmed with the software algorithm in the actual application circuit at the very last possible moment. The system designers of such an application would have maximum flexibility to make last minute changes to the software algorithm. Fortunately some new microcontrollers are moving in this direction.

Typically these unique microcontroller devices use eeprom (Electrically Erasable Programmable Read Only Memory) for both the program and data memories. Eeprom memories do not require special programming voltages, they can be programmed in the actual application circuit. A good example of such a device is Microchip's recently announced low cost 8-bit PIC16C84 microcontroller. It contains 1k words of eeprom program memory and 64 bytes of data memory. The program memory can be programmed serially which also contributes to the ease of in-circuit application programming.

In-circuit programming of eeprom based microcontrollers offers several potential advantages for automotive applications:

- 1) As mentioned already, last minute software code changes can be made painlessly in-circuit.
- 2) Electronic modules can be upgraded by simply reprogramming the software algorithm. In many cases this would save the time and effort required to replace the entire electronic module or sub-system.
- 3) Over time, equipment parameters may change and the new parameters may need to be stored. For example, in an active suspension system the shocks

may begin to wear out. To compensate, new parameters may need to be saved, under software control to the eeprom to maintain proper ride and road handling characteristics.

- 4) In an automobile local area network (lan) in-circuit programming could be useful in establishing addresses on the lan. Standard modules could be placed as lan nodes throughout an automobile. Each node's microcontroller could learn and permanently store its individual lan address in-circuit.

ABS

ABS designs have traditionally used 16-bit microprocessors or microcontrollers. However new high end 8-bit microcontrollers such as Microchip's PIC17C42 now offers the high execution performance necessary while offering a considerable saving in system cost. Below is a block diagram of low an ABS brake system could be implemented with the PIC17C42.

And ABS systems must be ultra reliable and thus so must the ABS microcontroller. New high-end devices go to considerable lengths to ensure ultra reliable execution of the software algorithm. In the case of the PIC17C42, for example, these steps include:

- 1) A very reliable on-chip Watchdog Timer with its own independent on-chip RC timebase. The purpose of the Watchdog Timer is to reset the microcontroller should it detect improper execution of the software algorithm. The Watchdog Timer can not be disabled through software execution.
- 2) Due to a rather unique Harvard Architecture, there are separate busses for Program and Data flow. There is no chance that the PIC17C42 could somehow execute Data information as if it were part of the program.

In summary a high end 8-bit microcontroller can now offer the performance necessary for most ABS applications at a considerable saving in system cost. Also the newer devices take great pains to ensure ultra reliable execution of the software algorithm.

In this article we have explored several different automobile applications. We have pointed out how some new developments in microcontroller devices have helped make possible better automobiles with low design and manufacturing costs.

PIC17C42でのDCサーボ・コントロール

要約

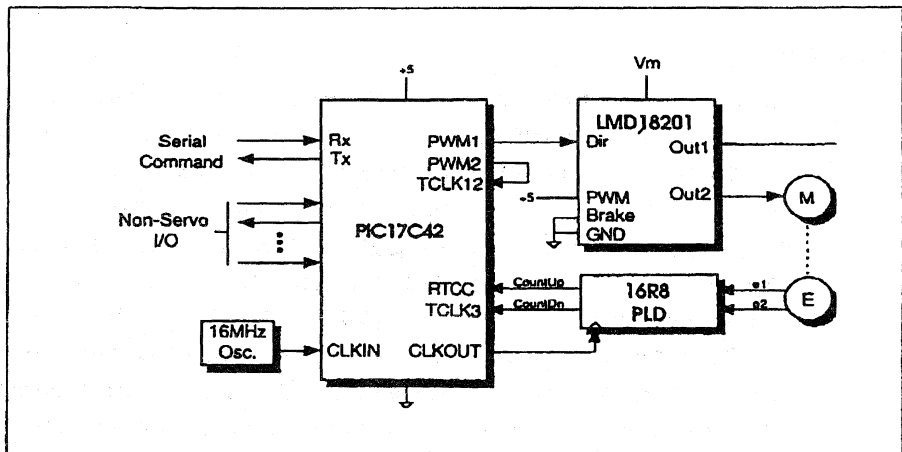
PIC17C42プロセッサ（以下、PIC17C42と表記する）は、埋込み型アプリケーションにおけるコスト/パフォーマンスがよいサーボ・コントロールの設計に非常に適しています。PIC17C42のハード、RISCライク・アーキテクチャにより、サーボ・ループを高速でクローズできます。本アプリケーション・ノートは、DCモータのサーボ・コントローラとしてのPIC17C42の使い方について述べます。また、PID制御の計算は16MHzの場合360 μ S以内で行なえるので、制御ループのサンプル・タイムは2KHzの範囲内であることを示します。さらに、PIC17C42はペリフェラルを内蔵しているので、最小限のコストで完全なシステムを構成できます。

序文

PIC17C42は、オプティカル・エンコーダのフィードバックの機能をもつ1軸のDCサーボ・コントローラの中核として使えます。以下のサーボ・システムの例では、PIC17C42によって生成される高分解能のPWMをHブリッジ・モータ・ドライバへの入力として使います。このシステムで、最大3MHzのエンコーダ・フィードバック信号を処理できます。図1に示すように、この制御システム全体は、PIC17C42、PLDが1個およびHブリッジのドライブ・チップで構成されます。

PICモータ制御

このようなシステムは、プリンタ、プロッタまたはスキャナのポジショニング・コントローラとして使われます。PIC17C42を使うと低いコストでコントローラを設計できるので、このシステムはステッパ・モータ・システムに対して有利にでき、また、以下のように多くの利点を得られます：



PIC17C42でのDCサーボ・コントロール

- 加速度（速度）の向上
- 効率の向上
- 可聴ノイズの軽減
- 外来ノイズの完全な除去

エンコーダ・フィードバック

上記のシステムの例のポジション・フィードバックは、モータ・シャフトに取り付けられた直交エンコーダによって行ないます。位置と方向の両方の増分は、この低価格のデバイスによって符号化されます。図1に示すように、直交エンコーダの信号は、16R8系のPLDデバイスで処理されます。PLDIは、直交パルスを、カウント・アップとカウント・ダウンの2つのパルス・ストリームに変換します。次に、図1に示すように、これらの信号はPIC17C42のRTCCとTMR 3の入力に送られます。

PIC17C42は、内蔵の16ビットのタイマであるRTCCとTMR 3の2つの差分を計算することによって、モータの増分の位置に追従制御します。サーボ・サンプル・タイムごとにこの処理を行ない、増分を前の位置に加えることによってカレントの位置を計算します。タイマは両方とも16ビット幅なので、エンコーダの信号の周波数がサンプルの周波数の32,767倍より高くないかぎり、オーバフローに注意を払う必要はありません。たとえば、サーボ・サンプル・タイムが1mSであれば、エンコーダの最高速度は3.2767MHzになります。

2つのカウンタ間の差分のみを使うので、カウンタのラップアラウンドについては問題となりません。2の補数の減算により、これを自動的に処理します。上記の例は、2バイト、つまり16ビット精度で位置を追従制御します。しかし、内部の位置レジスタの大きさには限界がありません。したがって、サンプル・タイムごとに16ビットの増分の位置をNバイトのソフトウェア・レジスタに加えることによって、Nバイトの位置を保持できます。

モータの駆動

PIC17C42には、すぐれたPWM (Pulse Width Modulation) サブシステムの機能があります。これを簡単なスイッチング・パワー段と組み合わせた場合、効率のよいパワーD/Aコンバータを構成できます。PIC17C42のPWMサブシステムの分解能は62.5nSです。これは、15.6KHzのサンプル・レートで10ビット、または20KHzで800分の1 ($9 \cdot 1/2$ ビット)の分解能に変換します。これにより、変調周波数を人間の聴覚の限界以上に保ちながら、すぐれた電圧制御が行なえます。これは、最小限のノイズを設計の目標とするオフィス・オートメーション機器に特に適しています。

モータは、PWM出力のデューティ・サイクルを時間平均するようにPWMの出力段に応答します。モータには電気的時定数があるので、ほとんどのモータはゆっくり反応し、PWM出力のデューティ・サイクルを時間平均するようにPWMの出力段に応答します。また、ほとんどのモータは、0.5mS以上の電気的時定数と20.0mS以上の機械的時定数でゆっくり反応します。15KHzのPWM出力は、実際はリニア・アンプ

PIC17C42でのDCサーボ・コントロール

の出力に相当します。

図1に示すシステムにおいて、HブリッジのDirection入力端子はPIC17C42のPWM出力端子に直接接続されています。HブリッジにはDC供給電圧 V_m がかかっています。この構成では、PWM信号が50%のデューティ・サイクルのときモータには0ボルトがかかり、0%のデューティ・サイクルのとき $-V_m$ ボルト、および100%のデューティ・サイクルのとき $+V_m$ ボルトがかかります。

PIDアルゴリズム

PIDは、最も広く使われているサーボ・モータ・コントロールのアルゴリズムです。このコントローラがすべてのアプリケーションに最適であるとはかぎりませんが、理解しやすく、また、調整が容易です。

図2に、標準のデジタルPIDアルゴリズムの形を示します。U(k)はポジション・エラー、また、Y(k)は出力です。

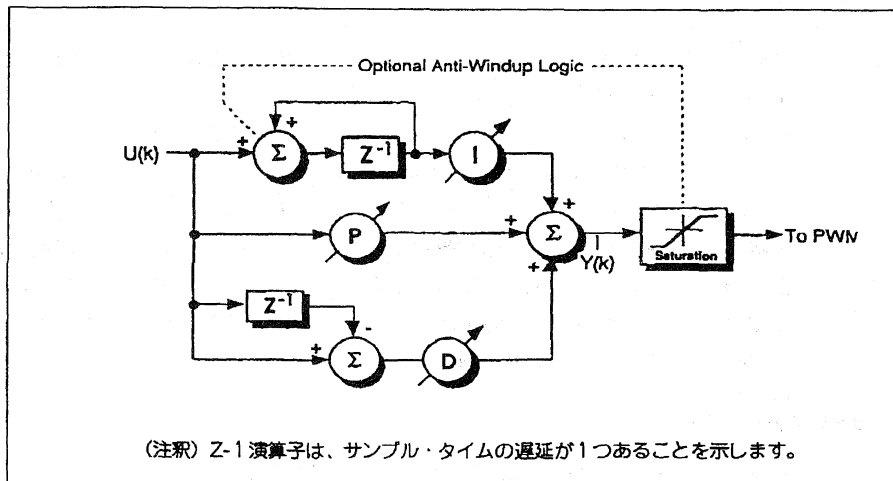


図2 デジタルPIDの仕組み

このアルゴリズムは、計算の効率と数値安定度のために、標準の差分方程式としてつくられています：

$$Y(k) = Y(k-1) + C0U(k) + C1U(k-1) + C2U(k-2)$$

ここで、以下のことが容易にわかります：

$$C0 = P + I + D$$

$$C1 = - (P + 2D)$$

$$C2 = D$$

C1、C2およびC3は、オーバーヘッドをできるかぎり小さくするために、サンプル・タイムの割り込みルーチン外で計算されます。

PIC17C42でのDCサーボ・コントロール

PICモータの制御

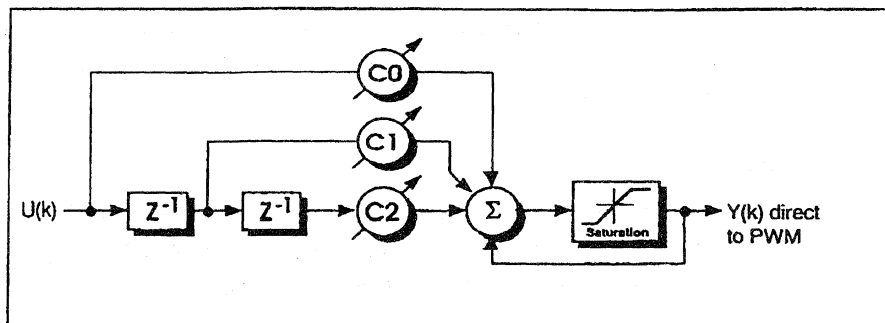


図3 アンチ・wind・アップ機能をもつ、計算によって改良されたPID

図3に、この差分方程式のとおり設計されたPIDを示します。PIC17C42に必要なサイクル数は571サイクルのみで、その結果、PIDの実行時間は0.36mSになります。

積分器のwind・アップとは、システムの中で大きなエラーが発生する場合、たとえば、大きなステップ・レスポンスが命令されたことによってPIDコントローラにおいて起こる状態です。積分器は、たとえこの出力が飽和しても、このエラーの状態の同積分し続けます。その後、積分器は、サーボ・システムが最後のデスティネーションに達し過大発振を起こすと“アンwind”します。この問題は、出力が飽和した場合に積分器の動作を止めることで避けられます。ここに示すPIDの設計には、単に $Y(k-1)$ を最後のサイクルでPWMタイマに実際に出力される値まで飽和させることによって、積分器の“アンチ・wind・アップ”を行なえるという利点があります。

サーボ・システムの構成

上記のコード例で、PIC17C42を使ってサーボ・システムを容易に構成できます。このプログラムの構造は非常に簡単です。すなわち、割り込みサービス・ルーチンがサーボ・コントロールの計算を処理し、また、フォアグラウンド・ループを使ってユーザ・インタフェース、シリアル・コミュニケーションおよびすべての例外処理（すなわち、リミット・スイッチ、ウォッチドッグ・タイマなど）を行ないます。フォアグラウンド・ループは本アプリケーション・ノートの範囲外なので、ここでは説明しません。

割り込みサービス・ルーチンの構造は簡単です。サーボ・コントロールを行なうためには、すべて一定のあらかじめ設定されたレートでエンコーダを読み込み、PID制御を計算し、また、PWMの出力を設定することが必要です。これを行なう最も簡単な方法は、割り込みサービス・ルーチンを、PIC17C42のハードウェアのタイマのうちの1つによって開始させることです。サーボの計算が常にPWMサブシステムと同期して確実に行なわれるように、PWM2出力をTMR1とTMR2の入力ピンに（TMR1を内部でクロック同期がとられる8ビットのタイマ・モードで、また、TMR2を外方でクロック同期がとられる8ビットのカウンタ・モードで）接続します。PR2レジスタにNビットをロードすると、サンプル・レートはNで割

PIC17C42でのDCサーボ・コントロール

ったPWMレートになります。

以下は、割り込みサービス・ルーチンの中で行なわれなければなりません：

- ・ タイマ (RTCCとTMR3) の読み込み
- ・ 参照位置の更新
- ・ エラーの計算 $U(k) = \text{参照位置} - \text{カレントの位置}$
- ・ PIDを使った $Y(k)$ の計算
- ・ PWM出力の設定
- ・ 他のハウスキーピング・タスクの管理 (すなわち、シリアル文字の処理)

速度を制御するために参照位置を計算する場合、サンプル・タイムごとに、単に、サンプル・タイムあたりのカウント数で表す速度を最後の参照位置に加えます。また、位置を制御する場合は、サンプル・タイムごとに、目的とする位置の軌道に基づいて参照位置を計算します。

サーボ・システムの構成は、上記のように非常に簡単です。

完全なシステム

PIC17C42を使って、デモンストレーションのサーボ・システムを構成しました。このシステムには、完全なRS-232インタフェース、内蔵のスイッチング電源、Hブリッジのモータ・ドライブ、オーバ・カレント・プロテクション、リミット・スイッチ入力およびデジタルI/Oがあります。システム全体が、5×3.5インチのプリント回路板上に収まっています。このシステムによって、サーボ・アプリケーションにおけるPIC17C42を評価できます。

デモンストレーションのサーボ・システムにおけるすべての未使用のPIC17C42のピンは、プロトタイプ製作時にI/Oコネクタで使えます。まもなく、完全に当社のPIC17C42のサーボ・デモンストレーション・システムが供給されます。

◆◆◆◆◆

Article Re-Prints For PIC17CXX Microcontrollers

NOTES:



Electronic Times, May 27, 1993

TAKE YOUR PIC WITH CONTROLLERS

Microchip's PICSTART-16B microcontroller development tool provides evaluation and development support for the company's 8 bit PIC microcontroller family.

It supports the range of low-end PIC16C5X microcontrollers as well as the mid-range PIC16C71 which includes adc and the forthcoming PIC16C84 complete with eeprom.

The PICSTART-16B package includes Microchip's MPALC Assembler, a pc based symbolic cross-assembler; the MPSIM Simulator, a discrete event simulator; and a 3 x 5 in development programmer board. Also included is a copy of Microchip's embedded controller handbook. PICSTART-16B's programmer board connects to a pc and accepts 18 and 28 lead otp pdip PIC16CXX devices. The kit includes software to read and program all PIC16CXX microcontroller products.

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.

PIC is a registered trademark of PIC Gesellschaft für wissenschaftliche, technische und kommerzielle Datenverarbeitung mbH in Germany.

Microchip Tech rewrites the E²PROM

Chandler, Ariz.—Seeing a fundamental shift in the way serial E²PROMs are used, Microchip Technology Inc. is altering the design of these previously jelly-bean parts, moving them into the high-density, smart-memory arena.

The changes mark a new product direction for Microchip and might also signal a new course for serial E²PROMs in general, along with a possible reconsideration of the role technology will play in systems design.

“Traditionally, small serial E-squareds have been used as a replacement for DIP switches,” observed Microchip senior product marketing engineer Peter Sorrells. “But that is changing.”

In many designs, the electrically erasable but non-volatile parts just store a few installation parameters or pieces of calibration data in a 1- or 2-kbit chip. The parts’ serial interface is valued because it requires little board space and only a few pins on a microcontroller. The data in E²PROM is rarely changed; moreover, such chips are never required to store large amounts of information. Consequently, customers have seen tiny packages and spare-change prices as the greatest virtue of the devices.

But Microchip claims a new generation of systems—from cellular phones to data-acquisition systems to keyless entry systems—values the non-volatility and compactness of the devices, but is much more demanding.

“In the cellular telephone market, for instance, we have talked to people who are using 64-kbit parallel E²PROMs now and are moving to serial,” Sorrells said. “They are storing hundreds of phone numbers, big configuration tables and code to support several different cellular protocols, so they need a lot of memory. But they don’t need parallel speed—a fast serial I/O scheme is sufficient.

SMART GENERATION

Microchip is responding to these needs with the first of what promises to be a new generation of smart, but not application-specific, memory parts. While retaining the familiar I²C serial interface, the chips differ from conventional 1- or 2-kbit serial E-squareds in almost every other regard.

The most obvious change is size. Microchip’s 24C65 is the world’s first 64-bit serial E²PROM—four times the capacity of the largest previously announced part. This gives the user enough space not only for the DIP-switch settings, but also for storage of larger data structures and

even code segments that traditionally would have ended up in a parallel EPROM or masked ROM. Systems designers have found—particularly in space-constrained handheld systems—that the ability to save a package by making the tiny serial E²PROM do double duty can more than offset the part’s higher cost-per-bit.

But this means the chip will require both density and considerably greater endurance that has been typical of small E²PROMs. Microchip has addressed this issue by splitting the memory array into two sections.

One 4-kbit block of the memory uses Microchip’s fully redundant cell design to give very high endurance—typically 1 million erase/write cycles. The remaining 60 kbits use a higher density cell rated for at least 10,000 cycles. By steering information of the memory array, customers can maintain both highly volatile data and non-volatile code or data tables in the same device.

To act as a multipurpose memory, the chip will need high bus bandwidth as well as high capacity and endurance. Consequently, Microchip has augmented the I²C bus to run at a minimum of 400 kHz. Sorrells claimed the interface had run considerably faster in the lab, but that 400 kHz would be the initial spec. The address format has been expanded to permit up to a 512-kbyte address range, allowing multiple chips to inhabit the buds in a single address bank.

In addition, the chip has been furnished with a 64-byte input write cache to keep the bus free during the rather long self-timed write operations. This permits a microcontroller to burst anything from an individual byte to several pages of data over the bus, and then go on with its business, addressing other chips.

In an unusual feature specifically for the instrumentation market, the write cache can be configured as a capture buffer. In this mode, the cache shifts incoming data until a signal tells to freeze and perform a write operation.

In keeping with the industry’s power fetish, the new architecture also has integral power management. Since the memory array is inherently non-volatile, Microchip has developed circuitry to completely power the array down between operations. On completion of a write or read operation, the chip goes to standby, a state in which the current is under 5 microamps. Power-up is fast enough to be transparent to the bus.

Finally, the chip is protected against incidental erasure or write operations during power hits, according to Microchips’s memory director Mitch Little.

With the architectural and circuit changes Microchip has made, Little seems confident that the part can take on an expanded role in systems designs.

SERIAL EEPROM FOR EMBEDDED APPLICATIONS

The optimum non-volatile memory technology for a system that requires small board area, byte level flexibility, low power demand and favourable costing? Consider serial EEPROMs, says Richard Fisher

Serial EEPROM technology has recently emerged as a leading non-memory solution for embedded control applications. The technology offers a number of benefits, which make it well-suited for a wide variety of applications.

For example, the small footprint, the ability to operate at low voltages and the lower power dissipation of serial EEPROM devices compared to parallel types makes them a good solution in portable systems such as cellular telephones, cameras and keyless entry systems; byte-level erase, write and read is of particular use in TV tuners; and the availability of multiple non-volatile functions in the same application offers many advantages in VCRs.

SIMPLIFIED DESIGN

A serial EEPROM requires only 10 percent of the board space required by a parallel device, significantly fewer I/O lines from the microcontroller, and draws much less current (see Fig. 1). The only significant limitation of

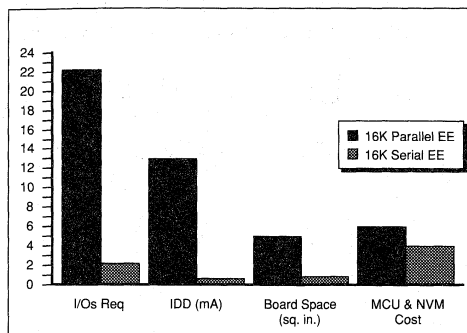


Figure 1— 16k serial, vs 16k parallel memory across a number of parameters

a serial EEPROM is a lower read speed but, in many applications, this parameter is restricted more by the protocol than the hardware. For example, in two-wire I²C (Inter-Integrated Circuit) systems, large internal delays are needed to slow the part to meet the 100 kHz protocol requirements. Characterization of three-wire bus serial EEPROMs has indicated that clock frequencies of over 6 MHz can be supported.

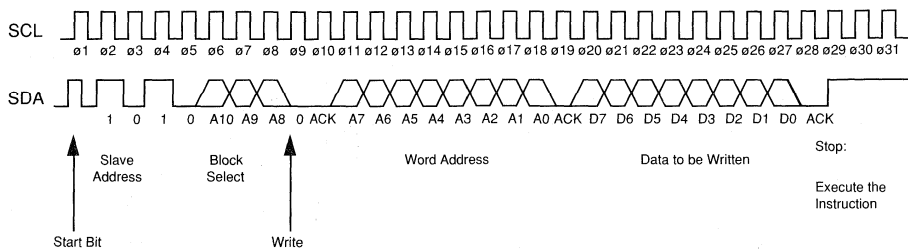
PROTOCOL OPTIONS

Once a designer has decided to use a serial EEPROM solution, the next key step is to choose either a two- or three-wire protocol. Often, this decision is taken for reasons of familiarity, rather than based on a full appraisal of benefits.

TABLE 1

3-Wire Bus Serial EEPROMs	2 Wire Bus Serial EEPROMs
Single VDD supply of <2V to 5.5V	Single VDD supply of <2V to 5.5V
Very low current consumption	Very low current consumption
Reduced overall component cost	Reduced overall component cost
Four pins (other than VCC & GND are required for operation)	Two pins (other than VCC & GND are required for operation)
X16 bit and X8 bit data widths	X8 bit data width
Software WRITE Protection	Hardware WRITE Protection
Edge triggered clocks and signals filters for high noise immunity	Level triggered clocks and signals and inputs glitch
2MHz+ operation	I ² C standard 100kHz and 400kHz protocols with a 1MHz option
Ready/Busy data polling	Page WRITE capability to 16 bytes
Security options available	Software and hardware compatible from 2k to 16k densities
Less complex protocol	

FIGURE 3A—2-wire bus example; a) byte write cycle timing



A two-wire product is usually used in applications that require an I2C bus, noise immunity, or a write buffer for multiple bytes to be stored in one instruction, and where there is limited microcontroller I/O pin availability. A three-wire protocol is the preferred solution in applications that have limited protocol requirements, an SPI protocol, higher clock frequency requirements, or a x16 data width. (See Table 1).

Many serial EEPROM data sheets are written in a conventional memory data sheet format that emphasises the features of the part more than the basic operating principles. Serial EEPROMs are not conventional memories, due to the serial communications protocols involved.

Three-wire bus operation Microchip's devices for three-wire bus operation are contained in the 93XXX family, ranging in density from 256bits to 4kbits. These devices require four pins in addition to Vcc and Gnd: CS (chip select), CLK (clock), DI (data in) and DO (data out). All 93XXX parts are hardware compatible for these four pins, although there may be compatibility issues for other pins. Software compatibility is a key issue, since there may be subtle differences in each manufacture's protocol. Software compatibility for density migration

should also be reviewed by designers on a case-by-case basis. There is no software industry compatibility from a 256bit to 4kbit part.

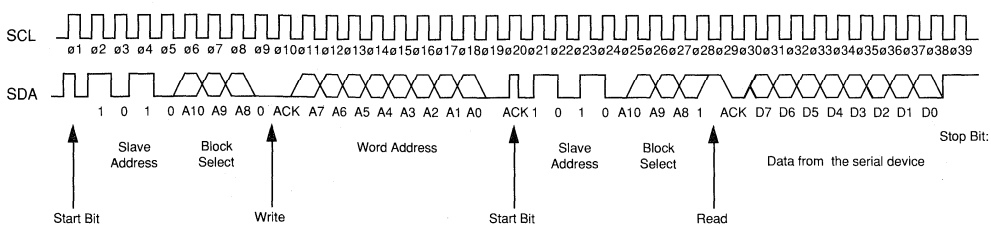
Data can be organised as either x8 or x16. This selection is determined either by the ORG pin or by purchasing a standard x16 organisation. Units always power-up in an EWDS (Erase/Write Disable State). All Erase and Write functions are disabled until the EWEN (Erase/Write Enable) instruction is performed. This feature prevents accidental data corruption. An Auto-Erase cycle is performed during each Write cycle.

Each instruction set requires the following: a start bit, which is the first Data-in high signal clocked in after CS is high; two bits of opcode to identify the instruction; a number of address bits depending on device type; and data. All members of the 93XXX family have separate Data-in and Data-out pins. However, these pins may be tied together for true three-wire operation.

TWO-WIRE BUS OPERATION

The 24XXX and 85XXX families of devices operate in two-wire bus systems. Only the SCL (serial clock) and SDA (serial data) pins are essential for bus operation.

FIGURE 3B—2-wire bus example; b) read cycle timing



The other pins on the devices, WP (active high Write protection) and A0/A1/A2 (chip or clock select), are supplementary. Data is organised as x8, and signals are level triggered, not edge triggered. Also, filters on the inputs will filter noise glitches less than 100nsec wide. As with the three-wire devices, an Auto-Erase cycle is performed during each Write cycle.

I²C BUS

One of the most popular two-wire protocols is I²C, which uses master/slave bi-directional communication. A device that sends data onto the bus is defined as a transmitter and a device that is receiving data is the receiver. Both the master and the slave can act as either transmitter or receiver. The bus must be controlled by a master device (usually a microcontroller) which generates the serial clock, controls the bus direction and generates the Start and Stop conditions.

The serial EEPROM is the slave. It will be the bus transmitter during Read operations and at times when it must acknowledge data transmitted by the master.

Bus activity is controlled by Start and Stop bits from the master. After a Start bit, each command must begin with an 8bit control byte. This byte must identify the serial EEPROM as the slave addressed on the bus; select the specific serial EEPROM or the internal memory block on the bus (there may be up to eight serial EEPROMs on the bus); and to select the Read or Write function for the next command transmitted by the master. With this selection scheme, devices from 2kbit to 16kbit are software compatible. For example, four 2kbit devices or one 8kbit device could be connected to the bus with the same software.

CONCLUSION

As the architecture of serial EEPROMs has evolved, and density has increased, these devices have become a realistic alternative in many applications that were previously restricted to using parallel non-volatile memory. Once the advantages and disadvantages of the two protocols reviewed above are understood by the designer, the serial EEPROM has much to offer.

Article Re-Prints For EEPROMS

NOTES



A TOOL FOR CALCULATING EEPROM ENDURANCE

Despite the widespread use of serial EEPROMs, EEPROM endurance is still not very well defined or understood concept in the industry. We are all used to blanket claims such as '1 million erase/write cycles typical', but such figures are not terribly useful - e2 endurance is dependent not only on the design of the device but also on the application environment in which it sits. Voltage, number of writes per day and the number of bytes written all have an effect.

Up until now there has been no tool available for predicting the endurance of a particular EEPROM device within a set of application parameters. Trade off analysis can be painfully time consuming and only marginally accurate without specific knowledge of the behaviour of the device under different conditions of use.

To address this problem Microchip has announced the Total Endurance Disk - a software emulator of the serial EEPROM hardware environment that allows the designer to place specific serial EEPROM devices into a theoretical application during the design and trade off stages of system development.

The Microchip Total Endurance disk allows you to trade off voltage, temperature, write cycles, number of bytes written, number of writes per day, ppm and FIT rates, as well as years of use in order to optimise the system accurately predict product lifetimes and reliability.

APPLICATION EXAMPLE

Here is an example using the Endurance Disk to help design an electronic phone book/auto dialer.

The auto-dialer may have new numbers added or changed several times per day, but how can manufacturers specify the life of the unit, and at what rate of update of the phone numbers? First the designer must make some assumptions. If we assume that the average user will change or add 50 phone numbers per day, and the manufacture is willing to live with a 0.1 percent failure rate (1,000 ppm) after 10 years of use, then we have almost enough information to verify whether we are in the ball park given the physics of the EEPROM device which will store the numbers.

We need to know the operating voltage and temperature of the applications; we will say that a 3.3V lithium button battery is powering the unit and the temperature range is limited to that for which the Icd display will function: 0 to 70degC. End of life voltage for the battery is approximately 2.0V; assuming that asic or microcontroller in the applications will operate down to 2.5V, the EPROM also has 2.5V requirement. The designer would like to be able to store 100 phone numbers of 16 bytes each, which results in a 1.6kbyte requirement for the serial EEPROM. Because 1.6kbytes is equal to 12.8kbits, a 16kbit 2-wire serial EEPROM will more than suffice. Specifically, Microchip's 24LC16B will operate down to 2.5V and even includes a write-protect feature which can be used to block inadvertent writes in a noisy environment.

Here is a summary of the application:

Device	24LC16B
Voltage	2.5V - 3.3V
Temperature	0 to 70degC (55degC typ)
Cycles per day	50
Bytes per cycle	16
Application life	10 years

Once these values are entered into the Total Endurance program, it outputs the following:

Device Data	Input Parameter
Device	24LC16B
Voltage	3.3V
Temperature	55
Bytes/Cycles per day	16
E/W	50
Application life (Years)	10 years
Cycling Mode	byte
Pulse Widths (Ms)	N/A
Data Pattern	Random

Device Data	Input Parameter
FIT	21.0
PPM	1,842
Time	10.00
Write Cycles	182,500

Unfortunately for our designer, the desired 0.1 percent failure rate has almost doubled to 0.18 percent (1842ppm). But fortunately for the designer, the Endurance disk makes tradeoff analysis very simple and fast. At this point there are at least three options: 1)

Article Re-Prints For EEPROMS

Components in Electronics, July/August, 1993

live with almost 2000ppm 2) look at the endurance plot and check whether there is a reasonable number of erase/write cycles which will provide a 1000ppm failure rate or 3) specify a ppm rate to the Endurance program and let it crank out the number of cycles it will take.

You can see by reducing the number of cycles from the 182500 which resulted from our first trial to about 100,000, we can achieve a ppm rate of about 1000 (0.1 percent). But how does 100,000 cycles translate into applications life or cycles per day?

Given a 1000ppm failure rate, you can ask for the application life of the product. Here are the results:

Device Data	Input Parameter
Device	24LC16B
Voltage	3.3V
Temperature	55
Bytes/Cycles per day	16
E/W	50
PPM Level (Years)	1000
Cycling Mode	byte
Pulse Widths (Ms)	N/A
Data Pattern	Random

Device Data	Input Parameter
PPM	1000
Time	5.97
Write Cycles	109,000

Now we have some more options: 1) specify the product life at five years or 2) trade off other parameters of the application or 3) device which is more important - a 10 years product lifetimes or the ability to change 50 numbers every single day. Realistically a user may enter or change quite a few numbers the first week or two of the application, and after that the unit will be used mostly for reading and dialing numbers.

Changing the number of erase/write cycles to 20 a day gives us the following results:

Device Data	Input Parameter
Device	24LC16B
Voltage	3.3V
Temperature	55
Bytes/Cycles per day	16
E/W	20
PPM Level (Years)	1000
Cycling Mode	byte
Pulse Widths (Ms)	N/A
Data Pattern	Random

Device Data	Input Parameter
PPM	1000
Time	14.93
Write Cycles	109,000

So reducing the number of cycles per day not only brought us back to a 10 year life, it gave some margin on that too. Keeping all the other parameters the same and forcing a 10 year lifetime gives us the following final results:

Device Data	Input Parameter
FIT	7.1
PPM	625
Time	10.00
Write Cycles	73,000

The new ppm rate of 625 gives our designer more than 30 percent margin on his ppm target of 1000.

This example shows the significant reduction in time for design tradeoff analysis and time to market which can be achieved with the Endurance Disk. In addition, it demonstrates the increase in robustness of the system design by proving known quantities and readily accessible handles to modify these quantities in the tradeoff analysis. This tool can literally reduce weeks of effort into a few minutes of point and clock.

The Total Endurance disk runs under Windows and costs \$29. Users can copy plots and past them into other windows programs or print them. Microchip plans to update the disk twice a year, adding data for new serial EEPROMs as they are introduced.



APPENDIX OFFICE LOCATIONS

Factory Representatives	A-1-1
Distributors	A-2-1
Field Sales Offices	A-3-1

Factory Representatives

Factory Representatives

Factory Representatives

ASIA

Hong Kong

Memec (Asia Pacific) Ltd.
Unit No. 2520-2525
Tower 1, Metroplaza
Hing Fong Road, Kwai Fong
N.T., Hong Kong
Tel: 852 410 2760
Fax: 852 418 1600

Korea

Memec (Asia Pacific) Ltd.
JE Woong Bldg. Third Floor
176-11 Nonhyum-Dong
Kangnam-ku
Seoul, Korea
Tel: 82 2 518 8181
Fax: 82 2 518 9419

Singapore

Memec (Asia Pacific) Ltd.
Singapore Representative Office
10 Anson Road #14-02
International Plaza
Singapore 0207
Tel: 65 222 4962
Fax: 65 222 4939

Taiwan

Memec (Asia Pacific) Ltd.
14F-1, No. 171, Sec. 5,
Min Sheng East Road
Hai Hwa Bldg.
Taipei, Taiwan
Tel: 886 2 7602028
Fax: 886 2 7651488

CANADA

Alberta

Enerlec Sales, Ltd.
37 Evergreen Terrace
Calgary, Alb. T2Y 2V9
Tel: 403 256 3627
Fax: 403 254 9121

British Columbia

Enerlec Sales, Ltd.
3671 Viking Way #7
Richmond, B.C. V6V 1W1
Tel: 604 273 0882
Fax: 604 273 0884

Ontario

Dynasty Components, Inc.
1140 Morrison Drive - Unit 110
Ottawa, Ontario K2H 859
Tel: 613 596 9800
Fax: 613 596 9886

Dynasty Components, Inc.
357 Hillside Avenue, E.
Toronto, Ontario M4S 1T9
Tel: 416 587 2278
Fax: 416 489 3527

Quebec

Dynasty Components, Inc.
1870 Blvd. des Sources, # 304
Pointe Claire, P.Q. H9R 5N4
Tel: 514 984 5342
Fax: 514 694 6826

EUROPE

Ireland

Eltech Agencies Ltd.
27 Maccurtain Street
Cork
Tel: 353 21 509366
Fax: 353 21 509344

SOUTH AMERICA

Brazil

Aplicacoes Eletronicas Artimar Ltda.
Rua Marques De Itu, 70-10 And.
Caixa Postal 5881 e 9498
CEP 01223 - Sao Paulo, Brazil
Tel: 231 0277
Fax: 255 0511

USA

Alabama

Electramark, Inc.
500 Wynn Drive, Suite 521
Huntsville, AL 35816
Tel: 205 830 4400
Fax: 205 830 4406

Arizona

Western High Tech Marketing, Inc.
9414 E. San Salvador, Suite 206
Scottsdale, AZ 85258
Tel: 602 860 2702
Fax: 602 860 2712

Arkansas

Comptech Sales, Inc.
9810 E. 42nd Street, Suite 219
Tulsa, OK 74146
Tel: 918 622 7744
Fax: 918 660 0340

California

Trinity Technologies
1261 Oakmead Parkway
Sunnyvale, CA 94086
Tel: 408 733 9000
Fax: 408 733 9970

Competitive Technology, Inc.
200 Baha Street, Suite 101
Costa Mesa, CA 92626
Tel: 714 540 5501
Fax: 714 540 5171

Eagle Technical Sales
1900 Sunset Drive, Suite A
Escondido, CA 92025
Tel: 619 743 6550
Fax: 619 743 6585

Colorado

Western Region Marketing, Inc.
9176 Marshall Place
Westminster, CO 80030
Tel: 303 428 8088
Fax: 303 426 8585

Connecticut

S. J. Associates Inc.
10 Copper Ridge Circle
Guilford, CT 06437
Tel: 203 458 7558
Fax: 203 458 1181

S.J. Associates Inc.
15 Coventry Lane
Naugatuck, CT 06770
Tel: 203 723 4707
Fax: 203 723 1629

Delaware

S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609 866 1234
Fax: 609 866 8627

Factory Representatives

Florida

Electramark Florida, Inc.
14021-B North Dale Mabry
Tampa, FL 33618
Tel: 813 962 1882
Fax: 813 961 0664

Electramark Florida, Inc.
401 Whooping Loop, Suite 1565
Altamonte Springs, FL 32701
Tel: 407 830 0844
Fax: 407 830 0847

Electramark Florida, Inc.
10360 NW 18 Manor
Plantation, FL 33322
Tel: 305 424 2872
Fax: 305 452 1974

Georgia

Electramark, Inc.
6030H Unity Drive
Norcross, GA 30071
Tel: 404 446 7915
Fax: 404 263 6389

Illinois

Janus Incorporated
650 E. Devon Ave.
Itasca, IL 60143
Tel: 708 250 9650
Fax: 708 250 8761

Indiana

Electro Reps, Inc.
407 Airport North Office Park
Fort Wayne, IN 46825
Tel: 219 489 8205
Fax: 219 489 8408

Electro Reps, Inc.
7240 Shadeland Station #275
Indianapolis, IN 46256
Tel: 317 842 7202
Fax: 317 841 0230

Iowa

Spectrum Sales
1364 Elmhurst Dr. NE
Cedar Rapids, IA 52402
Tel: 319 366 0576
Fax: 319 366 0635

Kansas

Spectrum Sales
5382 W. 95th Street
Prairie Village, KS 66207
Tel: 913 648 6811
Fax: 913 648 6823

Kentucky

Electro Reps, Inc.
7240 Shadeland Station #275
Indianapolis, IN 46256
Tel: 317 842 7202
Fax: 317 841 0230

TMC Electronics
7838 Laurel Avenue
Cincinnati, OH 45243
Tel: 513 271 3860
Fax: 513 271 6321

Louisiana

CompTech Sales, Inc.
15415 Katy Fwy., Suite 209
Houston, TX 77094
Tel: 713 492 0005
Fax: 713 492 6116

CompTech Sales, Inc.
2401 Gateway Dr., Suite 114
Irving, TX 75063
Tel: 214 751 1181
Fax: 214 550 8113

Maryland

Microchip Technology Inc.
Hauppauge, New York
Tel: 516 273 5305
Fax: 516 273 5335

Massachusetts

S. J. Associates Inc.
44 Mall Road
Burlington, MA 01803
Tel: 617 272 5552
Fax: 617 272 5515

Michigan

J.L.Montgomery Associates, Inc.
34405 W. 12 Mile Rd., Suite 149
Farmington Hills, MI 48331-5617
Tel: 313 489 0099
Fax: 313 489 0189

Minnesota

Comprehensive Technical Sales, Inc.
6525 City West Parkway
Eden Prairie, MN 55344
Tel: 612 941 7181
Fax: 612 941 4322

Mississippi

Electramark, Inc.
4910 Corporate Dr., Suite J
Huntsville, AL 35805
Tel: 205 830 4400
Fax: 205 830 4406

Missouri

Spectrum Sales
100 St. Francois Street, Suite 114
Florissant, MO 63031
Tel: 314 921 1313
Fax: 314 921 0701

Spectrum Sales
5382 W. 95th Street
Prairie Village, KS 66207
Tel: 913 648 6811
Fax: 913 648 6823

Nebraska

Spectrum Sales
5382 W. 95th Street
Prairie Village, KS 66207
Tel: 913 648 6811
Fax: 913 648 6823

Nevada

Western High Tech Marketing, Inc.
(Clark County Only)
9414 E San Salvador, Suite 206
Scottsdale, AZ 85258
Tel: 602 860 2702
Fax: 602 860 2712

New Jersey

Parallax
734 Walt Whitman Rd.
Melville, NY 11747
Tel: 516 351 1000
Fax: 516 351 1606

S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609 866 1234
Fax: 609 866 8627

Factory Representatives

New Mexico

Western High Tech Marketing, Inc.
9414 E San Salvador, Suite 206
Scottsdale, AZ 85258
Tel: 505 884 2256
Fax: 505 884 2258

New York

Apex Associates, Inc.
1210 Jefferson Rd.
Rochester, NY 14623
Tel: 716 272 7040
Fax: 716 272 7756

Parallax
734 Walt Whitman Road
Melville, NY 11747
Tel: 516 351 1000
Fax: 516 351 1606

North Carolina

Zucker Associates
4070 Barrett Drive
Raleigh, NC 27609
Tel: 919 782 8433
Fax: 919 782 8476

North Dakota

Comp. Technical Sales, Inc.
6525 City West Parkway
Eden Prairie, MN 55344
Tel: 612 941 7181
Fax: 612 941 4322

Ohio

TMC Electronics
7838 Laurel Avenue
Cincinnati, OH 45243
Tel: 513 271 3860
Fax: 513 271 6321

TMC Electronics
7017 Pearl Rd.
Middleburg Heights, OH 44130
Tel: 216 885 5544
Fax: 216 885 5011

Oklahoma

Comptech Sales, Inc.
18700 Woodbriar Lane
Catoosa, OK 74015
Tel: 918 622 7744
Fax: 918 660 0340

Oregon

Micro Sales
1865 NW 169th Place, Suite 210
Beaverton, OR 97006
Tel: 503 645 2841
Fax: 503 645 3754

Pennsylvania

S-J Mid-Atlantic, Inc.
131-D Gaither Drive
Mt. Laurel, NJ 08054
Tel: 609 866 1234
Fax: 609 866 8627

TMC Electronics
(Western Pennsylvania)
7838 Laurel Avenue
Cincinnati, OH 45243
Tel: 513 271 3860
Fax: 513 271 6321

Rhode Island

Microchip Technology Inc.
Five The Mountain Road
Suite 120
Framingham, MA 01701
Tel: 508 820 3334
Fax: 508 820 4326

South Carolina

Zucker Associates
4070 Barrett Drive
Raleigh, NC 27609
Tel: 919 782 8433
Fax: 919 782 8476

South Dakota

Comp. Technical Sales, Inc.
6525 City West Parkway
Eden Prairie, MN 55344
Tel: 612 941 7181
Fax: 612 941 4322

Tennessee

Electramark, Inc.
(Western Tennessee)
4910 Corporate Dr., Suite J
Huntsville, AL 35805
Tel: 205 830 4400
Fax: 205 830 4406

Zucker Associates
(Eastern Tennessee)
4070 Barrett Drive
Raleigh, NC 27609
Tel: 919 782 8433
Fax: 919 782 8476

Texas

CompTech Sales, Inc.
11130 Jollyville Rd., Suite 200
Austin, TX 78759
Tel: 512 343 0300
Fax: 512 345 2530

CompTech Sales, Inc.
1721 Villa Santos
El Paso, TX 79935
Tel: 915 590-4590
Fax: 915 590-4577

CompTech Sales, Inc.
15415 Katy Fwy., Suite 209
Houston, TX 77094
Tel: 713 492 0005
Fax: 713 492 6116

CompTech Sales, Inc.
2401 Gateway Dr., Suite 114
Irving, TX 75063
Tel: 214 751 1181
Fax: 214 550 8113

CompTech Sales, Inc.
85 NE Loop 410, Suite 202
San Antonio, TX 78216
Tel: 210 525 9913
Fax: 210 979 0311

CompTech Sales, Inc.
505 E. Jackson, Suite 300
Harlingen, TX 78550
Tel: 512 421 4501
Fax: 512 421 3265

Factory Representatives

Utah

Western Region Marketing, Inc.
3539 S Main - Suite 210
Salt Lake City, UT 84115
Tel: 801 268 9768
Fax: 801 268 9796

Virginia

Delta III Associates Inc.
12605 Wilde Lake Court
Richmond, VA 23233
Tel: 804 360 7507
Fax: 804 360 5258

Washington

Micro Sales
2122-112th Avenue, NE
Bellevue, WA 98004-1493
Tel: 206 451 0568
Fax: 206 453 0092

West Virginia

TMC Electronics
7838 Laurel Avenue
Cincinnati, OH 45243
Tel: 513 271 3860
Fax: 513 271 6321

Wisconsin

Janus, Inc.
375 Williamstowne
Delafield, WI 53018
Tel: 414 646 5420
Fax: 414 646 5421

Wyoming

Western Region Marketing, Inc.
9176 Marshall Place
Westminster, CO 80030
Tel: 303 428 8088
Fax: 303 426 8585

Distributors

Distributors

AFRICA

South Africa

Pace Electronic Components Pty
Cnr. Van Acht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
Tel: 11 974 1211
Fax: 11 974 1271

ASIA

Hong Kong

Maxisum Limited
Unit 2520-2525 Tower 1
Metroplaza, Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852 4180909
Fax: 852 4181600

Japan

Dainichi Contronics Inc.
Dainichi Bldg. 1-7 Karaku
1-Chome, Bunkyo-Ku
Tokyo 112, Japan
Tel: 3 3818 8081
Fax: 3 3818 8088

Marubeni Hytech Co., Ltd.
Marubeni Hytech Building
4-20-22, Koishikawa
Bunkyo-Ku
Tokyo 112, Japan
Tel: 3 817 4921
Fax: 3 817 4880

Nippon Precision Device Corp.
Nichibei Time 24 Bldg.
35 Tansu-Cho, Shinjuku-Ku
Tokyo 162, Japan
Tel: 3 3260 1411
Fax: 3 3260 7100

Korea

ProChips Inc.
779-12, Daelim 3-Dong
Youngdeungpo-Ku
Seoul, Korea
Tel: 02 849 8567
Fax: 02 849 8659

Taiwan, R.O.C.

Pinnacle Technologies Co. Ltd.
3F, 5 Lane 768, Sec. 4
Pa-Teh Road
Taipei, Taiwan
Tel: 02 788 4800
Fax: 02 788 5969

Solomon Technology Corp.
5th Floor, No. 293, Sec. 5
Chung Hsiao E. Rd.
Taipei, Taiwan
Tel: 886 2 760 5858
Fax: 886 2 756 9959

AUSTRALIA

NSD Australia
205 Middleborough Road
Box Hill, Victoria 3128
Tel: 61 03 890 0970
Fax: 61 03 899 5191

CANADA

Alberta

Future Electronics
3833 - 29th Street N.E.
Calgary, Alberta, T1Y 6B5
Tel: 403 250 5550
Fax: 403 291 7054

ITT Multicomponents
3015 - 5th Ave.
Suite 210
Calgary, Alberta, T2A 6T8
Tel: 403 273 2780
Fax: 403 273 7458

Future Electronics
4606 - 97th Street
Edmonton, Alberta, T6E 5N9
Tel: 403 438 2858
Fax: 403 434 0812

British Columbia

Future Electronics
1695 Boundary Road
Vancouver, B.C., V5K 4X7
Tel: 604 294 1166
Fax: 604 294 1206

Hamilton Hallmark
Burnaby, B.C., V5A 4N6
Tel: 800 332 8638

ITT Multicomponents
8525 Baxter Place, Unit 101
Production Court
Burnaby, B.C., V5A 4V7
Tel: 604 421 6222
Fax: 604 421 0582

Semad Electronics
3700 Gilmore Way
Burnaby, B.C., V5G 4M1
Tel: 604 451 3444
Fax: 604 451 3445

Manitoba

Future Electronics
106 King Edward
Winnipeg, Manitoba, R3H ON8
Tel: 204 786 7711
Fax: 204 783 8133

ITT Multicomponents
1313 Border Street
Unit 35
Winnipeg, Manitoba, R3H OX4
Tel: 204 697 2300
Fax: 204 697 2293

Ontario

Future Electronics
1050 Baxter Road
Ottawa, Ontario, K2C 3P2
Tel: 613 820 8313
Fax: 613 820 3271

Hamilton Hallmark
Nepean, Ontario, K2E 7J5
Tel: 800 332 8638

ITT Multicomponents
39 Robertson Road
Suite 506, Bell Mews
Nepean, Ontario, K2H 8R2
Tel: 613 596 6980
Fax: 613 596 6987

Semad Electronics
2781 Lancaster Road, Suite 302
Ottawa, Ontario, K1B 1A7
Tel: 613 526 4866
Fax: 613 523 4372

Future Electronics
5935 Airport Road, Suite 200
Mississauga, Ontario, L4V IW5
Tel: 416 612 9200
Fax: 416 612 9185

Hamilton Hallmark
Mississauga, Ontario, L5T 2L1
Tel: 800 332 8638

ITT Multicomponents
300 N. Rivermede Road
Concord, Ontario, L4K 2Z4
Tel: 416 798 4884
Fax: 416 798 4889

Semad Electronics
85 Spy Court
Markham, Ontario, L3R 4Z4
Tel: 416 475 3922
Fax: 416 475 4158

Distributors

Quebec

Hamilton Hallmark
Ville St. Laurent
Quebec H4T 1V6
Tel: 800 332 8638

ITT

5713 Shemin Street Francois
Ville St. Laurent
Quebec H4S 1W9
Tel: 514 335 7697
Fax: 514 335 9330

Semad Electronics

243 Place Frontenac
Pointe Claire, Quebec, H9R 4Z7
Tel: 514 694 0860
Fax: 514 694 0965

Future Electronics

1000 Ave. St. Jean Baptiste
Suite 100
Quebec City, Quebec, G2E 5G5
Tel: 418 877 6666
Fax: 418 877 6671

Future Electronics

237 Hymas Blvd.
Point Claire P.Q. H9R 5C7
Tel: 514 694 7710
Fax: 514 695 3707

EUROPE

Austria

Elbatex Electronics GmbH
Rotermuehlgasse 26
A-1120 Wien
Tel: 43 222 8135646
Fax: 43 222 834276

Elbatex Gruppe

Eitnergasse 6
A1233 Wien
Tel: 43 1 81602 0
Fax: 43 1 863211 201

Belgium

Sonetech N.V.
De Limburg Stirumlaan 243
Bus 3
B-1780 Wemmel
Tel: 32 2 460 0707
Fax: 32 2 460 1200

Denmark

Exatec A/S
Mileparken 20E
2740 Skovlund
Tel: 45 44 92 7000
Fax: 45 44 92 6020

England

Farnell Electronics
Canal Road
Leeds, LS12 2TU
Tel: 44 532 636311
Fax: 44 532 633411

Future Electronics Ltd.

Future House Poyle Road
Colnbrook, Berks, SL3 0EZ
Tel: 44 753 687000
Fax: 44 753 689100

H.B. Electronics Ltd.

Lever Street
Bolton, Lancashire, BL3 6BJ
Tel: 44 204 2 5544
Fax: 44 204 384911

Hawke Components Ltd.

26 Campbell Court
Bramley
NR Basingstoke, Hants, RG26 5EG
Tel: 44 256 880800
Fax: 44 256 880325

ITT Multicomponents

346 Edinburgh Ave.
Slough, Berks, SL1 4TU
Tel: 44 753 824131
Fax: 44 753 824160

Polar Electronics PLC

Cherrycourt Way
Leighton Buzzard
Bedfordshire LU7 8YY
Tel: 44 525 377093
Fax: 44 525 378367

France

ITT MULTicomposants
16 Avenue des Andes
ZA de Courtaboeuf - BP 16
91941 LES ULIS Cedex A
Tel: 33 1 64 460200
Fax: 33 1 64 463898

Mecodis

Parc d'Activites
3 Allée des Erables
94042 CRETEIL Cedex
Tel: 33 1 43 994400
Fax: 33 1 43 999828

Germany

Avnet E2000 GmbH
Postfach 820127
D-81801 Muenchen 82
Tel: 089 45110 01
Fax: 089 45110 210

Future Electronics Deutschland

GmbH
Munchner Strasse 18
85774 Unterfohing
Munich, Germany
Tel: 49 89 957 1950

Metronik GmbH

Postfach 1328
D82003 Unterhaching
Tel: 89 611080
Fax: 89 6112246

Semitron W. Roeck & Co.

Im Gut 1
D79790 Kuessaberg
Tel: 49 7742 80010
Fax: 49 7742 6901

Greece

P. Caritato & Associates S.A.

Iliia Iliou 31
Athens 11743
Tel: 30 1 9020115
Fax: 30 1 9017024

Israel

Elina Electronics Ltd.
14 Raoul Wallenberg St.
P.O. Box 13190
Tel Aviv 61131
Tel: 972 3 498543
Fax: 972 3 498745

Italy

Eurelettronica Srl
Viale E. Fermi 8
20090 Assago Milano
Tel: 39 2 457 841
Fax: 39 2 488 0275

Intesi

ZA Milanofiori E/5
20090 Assago Milano,
Tel: 39 2 8247 0215
Fax: 39 2 8247 0278

Kevin

Via del Gradenigo, 3
20148 Milano
Tel: 39 2 4870 6300
Fax: 39 2 4870 6500

Distributors

Netherlands

Semicon B.V.
Gulberg 33
P.O. Box 258
NL-5670 AG Nuenen
Tel: 31 40 837075
Fax: 31 40 832300

Norway

Odin Elektronik AB
P.O. Box 9376 Gronlund
N0135 Oslo
Tel: 47 22 677 290
Fax: 47 677 380

Spain

Sagitron
Corazon de Maria 80/82
28002 Madrid
Tel: 34 1416 9261
Fax: 34 1415 8652

Sweden

MEMEC Scandinavia AB
Kvarnholmsvagen 52
131 31 Nacka
Tel: 46 8 6434190
Fax: 46 8 6431195

Switzerland

Elbatex Gruppe AG
Hardstr. 72
CH-5430 Wettingen
Tel: 41 1 56 275111
Fax: 41 1 56 275454

Turkey

Inter Muehendislik Danismanlik
Ve Ticaret A.S.1
Hasircibasi Caddesi No. 55
81310 Kadikoy
Istanbul
Tel: 90 1 349 94 00
Fax: 90 1 349 94 30

SOUTH AFRICA

Isando

Pace Electronic Components Ltd.
Cnr. Vanacht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
Tel: 27 11 974 1211/6
Fax: 27 11 974 1271

SOUTH AMERICA

Brazil

Aplicacoes Electronicas Artimar
Rue Marques de Itu,70-1- AND.
Caixa Postal 5881*9498
Cep 01223
Sao Paulo, Brazil
Tel: 55112310277
Fax: 55112550511

USA

Alabama

Future Electronics
4835 University Square, Suite 16
Huntsville, AL 35816
Tel: 205 830 2322
Fax: 205 830 6664

Hamilton Hallmark
Huntsville, AL 35816
Tel: 800 332 8638

Pioneer Technologies
4835 University Square #5
Huntsville, AL 35816
Tel: 205 837 9300
Fax: 205 837 9358

Reptron Electronics
4835 University Square
Suite 12
Huntsville, AL 35816
Tel: 205 722 9500
Fax: 205 722 9565

Arizona

Bell Industries
140 S. Linden Lane #102
Tempe, AZ 85281
Tel: 602 966 7800
Fax: 602 967 6584

Future Electronics
4636 E. University Dr.
Suite 245
Phoenix, AZ 85034
Tel: 602 968 7140
Fax: 602 968 0334

Hamilton Hallmark
Phoenix, AZ 85040
Tel: 800 332 8638

California

Bell Industries
1161 N. Fair Oaks Ave.
Sunnyvale, CA 94089
Tel: 408 734 8570
Fax: 408 734 8875

Hamilton Hallmark
Sunnyvale, CA 94089
Tel: 800 332 8638

Future Electronics
2220 O'Toole Avenue
San Jose, CA 95131
Tel: 408 434 1122
Fax: 408 433 0822

Hamilton Hallmark
San Jose, CA 95131
Tel: 800 332 8638

Pioneer Technical Products
134 Rio Robles
San Jose, CA 95134
Tel: 408 954 9100
Fax: 408 954 9113

Bell Industries
4311 Anthony Court, Suite 100
Rocklin, CA 95677
Tel: 916 652 0414
Fax: 916 652 0403

Hamilton Hallmark
Rocklin, CA 95677
Tel: 800 332 8638

Bell Industries
11095 Knott Ave., Suite E
Cypress, CA 90630
Tel: 714 895 7801
Fax: 714 891 4570

Bell Industries
30101 Agoura Court, Suite 118
Agoura Hills, CA 91301
Tel: 818 879 9494
Fax: 818 991 7695

Pioneer Standard
5126 Clareton Drive
Agoura Hills, CA 91301
Tel: 818 865 5800
Fax: 818 865 5814

Future Electronics
9301 Oakdale Ave.
Suite 210
Chatsworth, CA 91311
Tel: 818 772 6240
Fax: 818 772 6247

Distributors

California (cont.)

Hamilton Hallmark
Wooland Hills, CA 91327
Tel: 800 332 8638

Future Electronics
1692 Browning Ave.
Irvine, CA 92714
Tel: 714 250 4141
Fax: 714 250 4185

Pioneer Standard
217 Technology Drive #110
Irvine, CA 92718
Tel: 800 535 1430
Fax: 714 753 5074

Aegis Electronic Group, Inc.
1015 Chestnut Ave.
Suite G2
Carlsbad, CA 92008
Tel: 619 729 2026
Fax: 619 729 9295

Bell Industries
7827 Convoy Court, Suite 403
San Diego, CA 92111
Tel: 619 268 1277
Fax: 619 268 3733

Future Electronics
5151 Shoreham Place
Suite 220
San Diego, CA 92122
Tel: 619 625 2800
Fax: 619 625 2810

Hamilton Hallmark
San Diego, CA 92123
Tel: 800 332 8638

Colorado

Bell Industries
1873 S. Bellaire St.
Denver, CO 80222
Tel: 303 691 9010
Fax: 303 691 9036

Future Electronics
12600 W. Colfax Avenue
Suite B110
Lakewood, CO 80215
Tel: 303 232 2008
Fax: 303 232 2009

Colorado (cont.)

Hamilton Hallmark
Englewood, CO 80111
Tel: 800 332 8638

Hamilton Hallmark
Colorado Springs, CO 80915
Tel: 800 332 8638

Connecticut

Future Electronics
24 Stony Hill Road
Bethel, CT 06801
Tel: 203 743 9594
Fax: 203 798 9745

Hamilton Hallmark
Cheshire, CT 06410
Tel: 800 332 8638

Hamilton Hallmark
Meriden, CT 06450
Tel: 800 332 8638

Phase 1 Technology Corporation
36A Padanarm Road
Danbury, CT 06811
Tel: 203 791 9042
Fax: 201 790 6128

Pioneer Standard
Two Trap Falls #101
Shelton, CT 06484
Tel: 203 929 5600
Fax: 203 929 9791

Florida

Bell Industries
650 S. Northlake Blvd, Suite 400
Altamonte Springs, FL 32701
Tel: 407 339 0078
Fax: 407 339 0139

Future Electronics
650 S. Northlake Blvd.
Suite 520
Altamonte Springs, FL 32701
Tel: 407 767 8414
Fax: 407 834 9318

Pioneer Technologies
337 South-North Lake #1000
Altamonte Springs, FL 32701
Tel: 407 834 9090
Fax: 407 834 0865

Hamilton Hallmark
Winter Park, FL 32792
Tel: 800 332 8638

Florida (cont.)

Future Electronics
2200 Tall Pines Drive
Suite 108
Largo, FL 34641
Tel: 813 530 1222
Fax: 813 538 9598

Hamilton Hallmark
Largo, FL 34641
Tel: 800 332 8638

Reptron Electronics
14401 McCormick Drive
Tampa, FL 33626
Tel: 813 854 2351
Fax: 813 855 0942

Pioneer Standard
674 S. Military Trail
Deerfield Beach, FL 33442
Tel: 305 428 8877
Fax: 305 481 2950

Hamilton Hallmark
Ft. Lauderdale, FL 33309
Tel: 800 332 8638

Reptron Electronics
3320 N.W. 53rd Street
Suite 206
Ft. Lauderdale, FL 33309
Tel: 305 735 1112
Fax: 305 735 1121

Georgia

Hamilton Hallmark
Duluth, GA 30136
Tel: 800 332 8638

Pioneer Technologies
4250 C Rivergreen Parkway
Duluth, GA 30136
Tel: 404 623 1003
Fax: 404 623 0665

Future Electronics
4960 Peachtree Industrial Blvd.
Suite 230
Norcross, GA 30071
Tel: 404 446 1300
Fax: 404 446 2991

Reptron Electronics
3040 Business Park Drive
Suite H
Norcross, GA 30071
Tel: 404 446 1300
Fax: 404 446 2991

Distributors

Illinois

Bell Industries
870 Cambridge Drive
Elk Grove Village, IL 60007
Tel: 708 640 1910
Fax: 708 640 0474

Future Electronics
3150 W. Higgins Rd.
Hoffman Estates, IL 60195
Tel: 708 882 1255
Fax: 708 490 9290

Hamilton Hallmark
Bensonville, IL 60106
Tel: 800 332 8638

Hamilton Hallmark
Wood Dale, IL 60191
Tel: 800 332 8638

Pioneer Standard
2171 Executive Drive #200
Addison, IL 60101
Tel: 708 495 9680
Fax: 708 495 9831

Indiana

Bell Industries
3433 E. Washington Blvd.
Fort Wayne, IN 46803
Tel: 219 422 4300
Fax: 219 423 3420

Bell Industries
5230 W. 79th St.
Indianapolis, IN 46268
Tel: 317 875 8200
Fax: 317 875 8219

Hamilton Hallmark
Indianapolis, IN 46268
Tel: 800 332 8638

Pioneer Standard
9350 N. Priority Way W. Dr.
Indianapolis, IN 46240
Tel: 317 573 0880
Fax: 317 573 0979

Kansas

Hamilton Hallmark
Lenexa, KS 66219
Tel: 800 332 8638

Kentucky

Hamilton Hallmark
Lexington, KY 40511
Tel: 800 332 8638

Maryland

Bell Industries
8945 Guilford Rd., Suite 130
Columbia, MD 21046
Tel: 410 290 5100
Fax: 410 290 8006

Future Electronics
6716 Alexander Bell Drive #101
Columbia, MD 21046
Tel: 410 290-0600
Fax: 410 290 0328

Hamilton Hallmark
Columbia, MD 21046
Tel: 800 332 8638

Pioneer Technologies
9100 Gaither Road
Gaithersburg, MD 20877
Tel: 301 921 0660
Fax: 301 921 4255

Vantage Components, Inc.
6925 R. Oakland Mills Road
Columbia, MD 21045
Tel: 401 720 5100
Fax: 401 381 2172

Massachusetts

Bell Industries
100 Burt Road Suite 106
Andover, MA 01810
Tel: 508 474 8880
Fax: 508 474 8902

Future Electronics
41 Main Street
Bolton, MA 01740
Tel: 508 779 3000
Fax: 508 779 5143

Hamilton Hallmark
Peabody, MA 01960
Tel: 800 332 8638

Pioneer Standard
44 Hartwell Ave.
Lexington, MA 02173
Tel: 617 861 9200
Fax: 617 863 1547

Michigan

Future Electronics
35200 Schoolcraft Road
Suite 106
Livonia, MI 48150
Tel: 313 261 5270
Fax: 313 261 8125

Hamilton Hallmark
Novi, MI 49418
Tel: 800 332 8638

Pioneer Standard
13485 Stamford
Livonia, MI 48150
Tel: 313 525 1800
Fax: 313 427 3720

Minnesota

Digi-Key Corporation
701 Brooks Ave. So.
P.O. Box 677
Thief River Falls, MN 55344
Tel: 218 681 6674
Fax: 218 681 3380

Future Electronics
10025 Valley View Road, Suite 196
Eden Prairie, MN 55344
Tel: 612 994 2200
Fax: 612 994 2520

Hamilton Hallmark
Bloomington, MN 55431
Tel: 800 332 8638

Pioneer Standard
7625 Golden Triangle
Eden Prairie, MN 55344
Tel: 612 944 3355
Fax: 612 944 3794

Missouri

Future Electronics
12125 Woodcrest Executive Dr.
Suite 220
St. Louis, MO 63141
Tel: 314 469 6805
Fax: 314 469 7226

Hamilton Hallmark
Earth City, MO 63045
Tel: 800 332 8638

Distributors

New Jersey

Bell Industries
271 Route 46 West
Suites F202-203
Fairfield, NJ 07004
Tel: 201 227 6060
Fax: 201 227 2626

Future Electronics
12 East Stow Road
Suite 200

Marlton, NJ 08053
Tel: 609 596 4080
Fax: 609 596 4266

Future Electronics
1259 Route 46 East
Parsippany, NJ 07054
Tel: 201 299 0400
Fax: 201 299 1377

Hamilton Hallmark
Parsippany, NJ 07054
Tel: 800 332 8638

Hamilton Hallmark
Cherry Hill, NJ 08003
Tel: 800 332 8638

Phase 1 Technology Corporation
295 Molnar Drive
Elmwood Park, NJ 07407
Tel: 201 791 2990
Fax: 201 791 2552

Pioneer Standard
14A Madison Road
Fairfield, NJ 07006
Tel: 201 575 3510
Fax: 201 575 3454

Vantage Components, Inc.
23 Sebago Street
Clifton, NJ 07013
Tel: 201 777 4100
Fax: 201 777 6194

New Mexico

Bell Industries
11728 Linn N.E.
Albuquerque, NM 87123
Tel: 505 292 2700
Fax: 505 275 2819

Hamilton Hallmark
Albuquerque, NM 87109
Tel: 800 332 8638

New York

Future Electronics
200 Salina Meadows, Suite 130
Syracuse, NY 13212
Tel: 315 451 2371
Fax: 315 451 7258

Future Electronics
333 Metro Park
Rochester, NY 14623
Tel: 716 272 1120
Fax: 716 272 7182

Hamilton Hallmark
Rochester, NY 14623
Tel: 800 332 8638

Hamilton Hallmark
Ronkonkoma, NY 11779
Tel: 800 332 8638

Future Electronics
200 Salina Meadows Parkway
Suite 130
Syracuse, NY 13212
Tel: 315 451 2371
Fax: 315 451 7258

Pioneer Standard
840 Fairport Park
Fairport, NY 14450
Tel: 716 381 7070
Fax: 716 381 5955

Pioneer Standard
1249 Front Street, Suite 201
Binghamton, NY 13904
Tel: 607 722 9300
Fax: 607 722 9562

New York (cont.)

Phase 1 Technology Corporation
46 Jefryne Blvd.
Deer Park, NY 11729
Tel: 516 254 2600
Fax: 516 254 2693

Future Electronics
801 Motor Parkway
Hauppauge, NY 11788
Tel: 516 234 4000
Fax: 516 234 6183

Pioneer Standard
60 Crossways Park West
Woodbury, NY 11797
Tel: 516 677 1726
Fax: 516 921 2143

Vantage Components, Inc.
1056 West Jericho Turnpike
Smithtown, NY 11787
Tel: 516 543 2000
Fax: 516 543 2030

North Carolina

Future Electronics
5225 Capital Blvd.
1 North Commerce Center
Raleigh, NC 27604
Tel: 919 790 7111
Fax: 919 790 9022

Hamilton Hallmark
Raleigh, NC 27604
Tel: 800 332 8638

Hamilton Hallmark
Charlotte, NC 28217
Tel: 800 332 8638

Pioneer Technologies
2200 Gateway Centre Blvd.
Suite 215
Morrisville, NC 27560
Tel: 919 460 1530
Fax: 919 460 1540

Distributors

Ohio

Future Electronics
6001-F Landerhaven Dr.
Mayfield Heights, OH 44124
Tel: 216 449 6996
Fax: 216 449 8987

Hamilton Hallmark
Solon, OH 44139
Tel: 800 332 8638

Pioneer Standard
4800 East 131st St.
Cleveland, OH 44105
Tel: 216 587 3600
Fax: 216 587 3906

Bell Industries
444 Windsor Park Drive
Dayton, OH 45459
Tel: 513 435 8660
Fax: 513 435 6765

Pioneer Standard
4433 Interpoint Blvd.
Dayton, OH 45424
Tel: 513 236 9900
Fax: 513 236 8133

Hamilton Hallmark
Worthington, OH 43085
Tel: 800 332 8638

Pioneer Standard
6421 East Main #201
Reynoldsburg, OH 43068
Tel: 614 221 0043
Fax: 614 759 1955

Oklahoma

Hamilton Hallmark
Tulsa, OK 74146
Tel: 800 332 8638

Pioneer Standard
9717 E. 42nd St.
Tulsa, OK 74146
Tel: 918 664 7840
Fax: 918 665 1891

Oregon

Bell Industries
9275 S.W. Nimbus
Beaverton, OR 97005
Tel: 503 644 1500
Fax: 503 520 1948

Future Electronics
15236 N.W. Greenbrier Pkwy.
Beaverton, OR 97006
Tel: 503 645 9454
Fax: 503 645 1559

Pennsylvania

Bell Industries
2550 Metropolitan Drive
Trevose, PA 19053
Tel: 215 953 2800
Fax: 215 364 4928

Pioneer Technologies
500 Enterprise Road
Keith Valley Business Center
Horsham, PA 21567
Tel: 215 674 4000
Fax: 215 674 3107

Pioneer Standard
259 Kappa Drive
Pittsburgh, PA 15238
Tel: 412 782 2300
Fax: 412 963 8255

Texas

Bell Industries
1701 Greenville Ave., Suite 306
Richardson, TX 75081
Tel: 214 690 0468
Fax: 214 690 0467

Future Electronics
1850 N. Greenville Ave.
Suite 146
Richardson, TX 75081
Tel: 214 437 2437
Fax: 214 669 2347

Hamilton Hallmark
Dallas, TX 75243
Tel: 800 332 8638

Texas (cont.)

Pioneer Standard
13765 Beta Road
Dallas, TX 75244
Tel: 214 386 7300
Fax: 214 490 6419

Hamilton Hallmark
Austin, TX 78727
Tel: 800 332 8638

Pioneer Standard
1826-D Kramer Lane
Austin, TX 78758
Tel: 512 835 4000
Fax: 512 835 9829

Pioneer Standard
8200 Interstate 10 W. #705
San Antonio, TX 78230
Tel: 512 377 3440
Fax: 512 377 3626

Future Electronics
11271 Richmond Ave., Suite 106
Houston, TX 77082
Tel: 713 556 8696
Fax: 713 589 7069

Hamilton Hallmark
Houston, TX 77063
Tel: 800 332 8638

Pioneer Standard
10530 Rockley Road
Houston, TX 77099
Tel: 713 495 4700
Fax: 713 495 5642

Distributors

Utah

Bell Industries
6912 S. 185 West, Suite B
Midvale, UT 84047
Tel: 801 561 9691
Fax: 801 255 2477

Future Electronics
3450 S. Highland Drive, Suite 301
Salt Lake City, UT 84106
Tel: 801 467 4448
Fax: 801 467 3604

Hamilton Hallmark
Salt Lake City, UT 84121
Tel: 800 332 8638

Washington

Bell Industries
8553,- 154th Ave. N.E.
Redmond, WA 98052
Tel: 206 867 5410
Fax: 206 867 5159

Future Electronics
19102 N. Creek Parkway South
Suite 118
Bothell, WA 98011
Tel: 206 489 3400
Fax: 206 489 3411

Hamilton Hallmark
Redmond, WA 98052
Tel: 800 332 8638

Wisconsin

Bell Industries
W. 226 N. 900 Eastmound Dr.
Waukesha, WI 53186
Tel: 414 547 8879
Fax: 414 547 6547

Future Electronics
20875 Crossroads Circle
Suite 200
Waukesha, WI 53186
Tel: 414 786 1884
Fax: 414 786 0744

Hamilton Hallmark
New Berlin, WI 53146
Tel: 800 332 8638

Pioneer Standard
120 Bishops Way #163
Brookfield, WI 53005
Tel: 414 784 3480
Fax: 414 784 8207

Factory Sales

Factory Sales

JAPAN

Microchip Technology International Inc.
Shinyokohama Gotoh Bldg. 8F, 3-22-4
Shinyokohama, Kohoku-Ku,
Yokohama-Shi
Kanagawa 222 Japan
Tel: 81 45/471 6166
Fax: 81 45/471 6122

ASIA/PACIFIC

Microchip Technology Inc.
Unit No. 2520-2525
Tower 1, Metroplaza
Hing Fong Road, Kwai Fong
N.T., Hong Kong
Tel: 852 410 2716
Fax: 852 410 2518

EUROPE

United Kingdom

Arizona Microchip Technology LTD.
Unit 3, Meadow Bank, Furlong Road
Bourne End, Bucks SL8 5AJ
Tel: 44 062 885 1077
Fax: 44 062 885 0178

Germany

Arizona Microchip Technology GMBH
Alte Landstrasse 12-14
D-8012 Ottobrunn, Germany
Tel: 49 089 609 6072
Fax: 49 089 609 1997

France

Arizona Microchip Technology SARL
2, Rue Du Buisson aux Fraises
F-91300 Massy, France
Tel: 33 01 6930 9090
Fax: 33 01 6930 9079

UNITED STATES

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786 7200
Fax: 602 899 9210

Northeast Region

Microchip Technology Inc.
Five The Mountain Road
Suite 120
Framingham, MA 01701
Tel: 508 820 3334
Fax: 508 820 4326

Mid-Atlantic Region

Microchip Technology Inc.
150 Motor Parkway
Suite 416
Hauppauge, NY 11788
Tel: 516 273 5305
Fax: 516 273 5335

Southeast Region

Microchip Technology Inc.
1521 Johnson Ferry Road NE
Suite 170
Marietta, GA 30062
Tel: 404 509 8800
Fax: 404 509 8600

North Central Region

Microchip Technology Inc.
665 Tollgate Road, Unit C
Elgin, IL 60123-9312
Tel: 708 741 0171
Fax: 708 741 0638

South Central Region

Microchip Technology Inc.
17480 N Dallas Parkway
Suite 114
Dallas, TX 75287
Tel: 214 733 0391
Fax: 214 250 4631

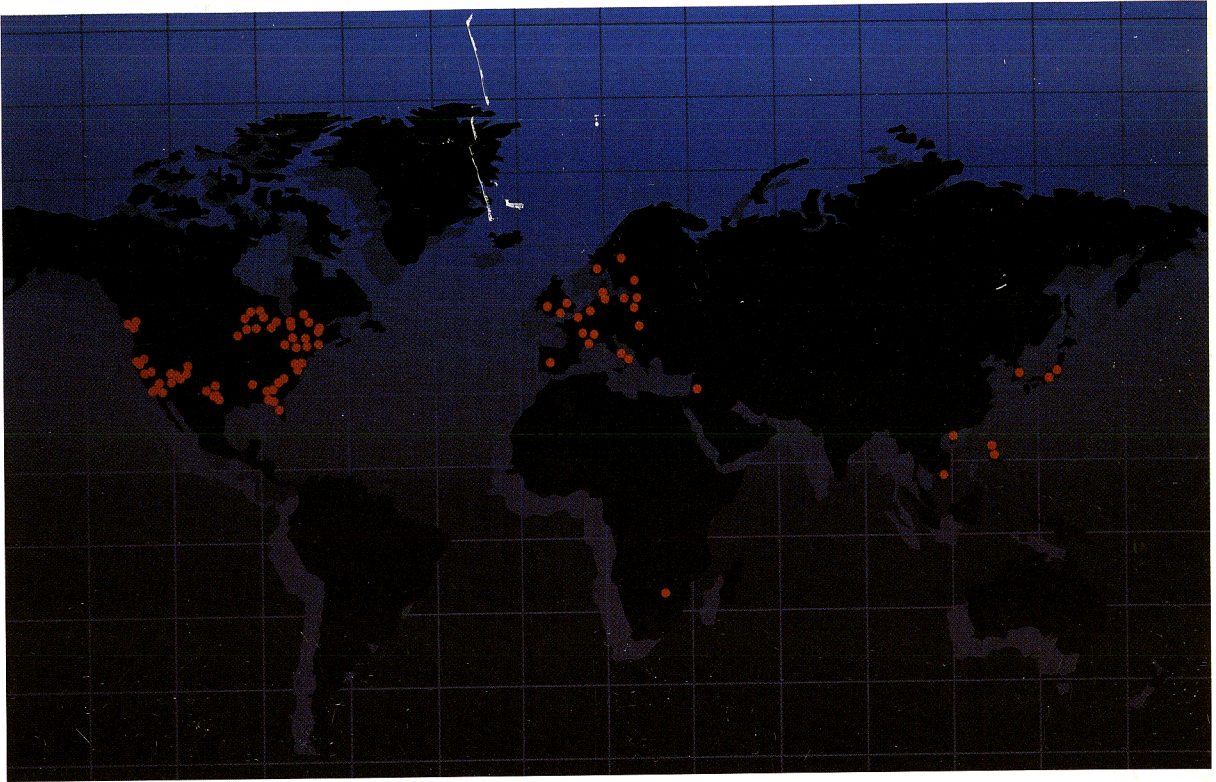
Northwest Region

Microchip Technology Inc.
2107 N First Street
Suite 410
San Jose, CA 95131
Tel: 408 436 7950
Fax: 408 436 7955

Southwest Region

Microchip Technology Inc.
18201 Von Karman
Suite 455
Irvine, CA 92715
Tel: 714 263 1888
Fax: 714 263 1338

NOTES



Microchip Worldwide Sales and Distribution



Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200, Fax: 602 899-9210