

TMS320C5x DSK

Applications Guide

TMS320C5x DSK Applications Guide

Literature Number: BPRA063
Texas Instruments Europe
October 1997

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Introduction

Digital Signal Processing (DSP) enables many of the key applications that we see in our everyday lives and which we will see in the future. Mobile phones, sound cards, graphics engines, image processors all have DSP engines. DSP is therefore an important competence area for all engineers. The skills required to implement DSP applications successfully tend to be different from those used when programming other processors such as microcontrollers.

At Texas Instruments we acknowledge the need for a DSP applications guide which makes the steep learning curve a little easier to climb. Using the knowledge and the experience we have acquired since launching the world's first DSP, this application guide aims to make the C50 Digital Signal Processing Starters Kit (DSK) more accessible to first time user.

Texas Instruments 'C50 Digital Signal Processing Starter's Kit (DSK) is the ideal low-cost development platform for running and debugging real-time applications. Based on TI's industry proven TMS320C50 DSP processor, users have access to a board with features that are normally available on development systems at a considerably higher cost.

By adopting a practical approach to DSP, this guide helps you to begin using the DSK and to start exploring its capabilities. While we appreciate that producing serious applications on the device will take time, this application guide should make the first few steps easier and enable users to create and develop their own software.

With the DSK board and accompanying of documentation, you have all the tools necessary to understand the programming process. We hope that within just a few hours spent with your DSK and this guide you will be able to:

- Get started with the DSK
- Assemble and run demonstration software examples
- Use demonstration software examples as useful programs
- Guide yourself through the steps necessary to develop DSK programs
- Modify software templates to develop your own programs
- Write your own software
- Go and learn more about DSP.

The guide is divided into four main sections, followed by appendices, glossary and index. Chapter 1 outlines an introduction to 'C50 DSP and the DSK board with useful background information. Chapter 2 illustrates programming principles and techniques with examples. Chapter 3 contains detailed explanations for some of the more sophisticated software examples. This chapter is an introduction for the serious applications developer to the more complex software design techniques for 'C50 DSP. Finally, Chapter 4 presents further software examples. Fully commented sources for all software are included on the accompanying disk.

We very much hope that this guide will be a useful handbook for DSK users.

Contents

1. Welcome to the DSP Starter's Kit	1
1.1 Digital Signal Processing	1
1.1.1 Overview.....	1
1.1.2 Analogue Computers.....	1
1.1.3 Real-Time Processor Performance.....	1
1.2 Why Digital?.....	2
1.3 The role of DSP	3
1.4 Texas Instruments DSP	5
1.4.1 TMS320 Family.....	5
1.4.2 Texas Instruments DSP Firsts	7
1.5 DSP systems	7
1.5.1 Overview.....	7
1.5.2 Typical System	8
1.6 Welcome to the TMS320C5x DSK.....	10
1.6.1 Overview.....	10
1.6.2 A walk around the DSK board	10
1.7 Digital Signal Processors	14
1.7.1 Overview.....	14
1.7.2 Architecture Types.....	14
1.7.3 The need for speed	16
1.7.4 Fixed-Point and Floating-Point	16
1.7.5 Typical Fixed-Point DSPs.....	17
1.8 A walk around the TMS320C5x Processor	18
1.8.1 Overview.....	18
1.8.2 Memory.....	19
1.8.3 The Core.....	20
1.8.4 Internal Registers.....	22
1.8.5 Memory-Mapped Core Processor Registers.....	22
1.8.6 Interrupts	23
1.8.7 Peripherals	24
1.8.8 Memory-Mapped Peripheral Registers	26
1.9 Introduction to Assembly Language Instructions	26
1.9.1 Overview.....	26
1.9.2 Direct Addressing Mode	27
1.9.3 Indirect Addressing Mode	28
1.9.4 Immediate Addressing Mode.....	30
1.9.5 Dedicated Register Addressing	31
1.9.6 Memory-Mapped Register Addressing	32
1.9.7 Circular Addressing	33
1.10 More on DSK	35

1.10.1 DSK Memory Map	35
1.10.2 DSK Block Diagram	36
1.10.3 The boot-up sequence of a DSK	36
2. The Programming Process	38
2.1 The DSK Package	38
2.2 Getting the system ready	38
2.3 Trouble-Shooting	38
2.4 Using the Debugger	40
2.5 Suggested Hardware Configuration	43
2.6 Installing Demonstration Software	43
2.7 Assembling and running a program	44
2.8 PASS.ASM	46
2.9 The fundamentals of a DSK Program	48
2.9.1 Overview	48
2.9.2 Settings	51
2.9.3 Interrupt Settings	52
2.9.4 Processor Initialisation	53
2.9.5 AIC Initialisation	56
2.9.6 Main Program	64
2.9.7 Interrupt Service Routines	64
2.10 A First Program	67
2.10.1 Theory	67
2.10.2 Calculations	68
2.10.3 Example of Q15 multiplication	69
2.10.4 The Program	71
2.10.5 From PASS to SINE	73
2.10.6 Running the program	76
2.11 From Sine to Musical Notes	79
2.11.1 Background	79
2.11.2 Implementation	80
2.11.3 Octave 6	81
2.11.4 Running the program	85
2.12 A PC Controlled Tune Generator	86
2.12.1 Background	86
2.12.2 Using RS232 communications in assembly programs	90
2.12.3 The Programs	93
2.12.4 Keyboard controlled notes	96
3. Application Software	97
3.1 Introduction	97
3.2 FIR Filter	97
3.2.1 Overview	97
3.2.2 Files and Equipment	97

3.2.3 Operation.....	98
3.2.4 Background	98
3.2.5 A FIR filter Implementation on DSK.....	102
3.3 FIR ISR	104
3.4 Amplitude Modulator	107
3.4.1 Overview.....	107
3.4.2 Files and Equipment.....	107
3.4.3 Operation.....	108
3.4.4 Background	108
3.4.5 Theory	109
3.4.6 Implementation	110
3.5 Reverberate	113
3.5.1 Overview.....	113
3.5.2 Files and Equipment.....	113
3.5.3 Operation.....	113
3.5.4 Background	114
3.5.5 Theory	114
3.5.6 Implementation	115
4. More Software Examples.....	122
4.1 Introduction	122
4.2 Minuet Player.....	122
4.2.1 Overview.....	122
4.2.2 Files	122
4.2.3 Operation.....	123
4.3 Sound Recorder.....	124
4.3.1 Overview.....	124
4.3.2 Files.....	124
4.3.3 Operation.....	124
4.4 Host Driven Function Generator	126
4.4.1 Overview.....	126
4.4.2 Files.....	126
4.4.3 Operation.....	126
4.5 Host Driven DTMF Dialler	128
4.5.1 Overview.....	128
4.5.2 Files	128
4.5.3 Operation.....	128
4.6 Spectrum Analyser.....	130
4.6.1 Overview.....	130
4.6.2 Files	130
4.6.3 Operation.....	130
4.7 Oscilloscope.....	132
4.7.1 Overview.....	132

4.7.2 Files.....	132
4.7.3 Operation	133
5. Additional Information.....	134
5.1 Hints and Tips.....	134
5.1.1 DSK processor	134
5.1.2 DSK software	134
5.1.3 DSK AIC.....	134
5.2 AIC Settings.....	136
5.2.1 Loading values	137
5.2.2 TA, TB, RA, RB Values	137
5.2.3 Typical Values	138
5.3 Questions and Answers.....	139
5.4 Known Bugs	139
5.4.1 Handshake error	139
5.4.2 DTMF	139
5.4.3 Function	140
5.4.4 Network Problems.....	140
5.4.5 DSK Debugger loads	140
6. Sources of Information	141
6.1 Product Information Centre	141
6.1.1 Overview	141
6.1.2 Product Information and Technical Support.....	141
6.1.3 BBS.....	142
6.2 Internet	142
6.2.1 TI&ME™ Internet Information Service.....	142
6.3 TI Workshops	143
6.4 Literature	144
6.4.1 The DSP Teaching Kit.....	144
6.4.2 TMS320C5x Related TI Books.....	144
6.5 Further Development.....	144
6.5.1 Hardware.....	145
6.5.2 Software	146

List of Figures

Figure 1: Advantages of a Digital Solution2

Figure 2: Embedded DSPs within a car4

Figure 3: Video4

Figure 4: Modem5

Figure 5: TMS320 Family6

Figure 6: A typical DSP System8

Figure 7: Sampling Process9

Figure 8: DSK Board Layout11

Figure 9: Von Neumann and Harvard Architecture15

Figure 10: Typical Fixed-Point DSP17

Figure 11: TMS320C5019

Figure 12: Data Memory Page Pointer (DP).....27

Figure 13: The operation of a Direct Memory Addressing Instruction28

Figure 14: Auxiliary Registers28

Figure 15: The operation of an indirect memory addressing instruction.....29

Figure 16: The operation of an immediate addressing mode instruction.....30

Figure 17: The operation of a PLU instruction on DBMR31

Figure 18: The use of BMAR in data memory access32

Figure 19: Loading of a memory-mapped register32

Figure 20: The operation of a circular buffer33

Figure 21: Memory Map of the C5x DSK35

Figure 22: DSK Block diagram36

Figure 23: DSK debugger main screen40

Figure 24: Filling data memory in DSK debugger.....41

Figure 25: Modifying contents of a register41

Figure 26: Debugger register display42

Figure 27: Suggested hardware configuration43

Figure 28: Loading a DSK file45

Figure 29: Choosing a download type.....45

Figure 30: Functionality of PASS.ASM.....46

Figure 31: PASS.ASM Flow Chart50

Figure 32: Memory Addresses51

Figure 33: PASS.ASM Settings.....52

Figure 34: PASS.ASM Interrupts.....52

Figure 35: PASS.ASM Initialisation53

Figure 36: Interrupt vector address calculation55

Figure 37: PASS.ASM Board Initialisation56

Figure 38: Reset AIC	58
Figure 39: AIC Registers	59
Figure 40: Setting AIC Registers	60
Figure 41: Contents of TA and Accumulator.....	60
Figure 42: Contents of RA and Accumulator	61
Figure 43: Contents of TB and Accumulator.....	61
Figure 44: Contents of RB and Accumulator	61
Figure 45: Bits of AIC control register.....	62
Figure 46: Writing to AIC	63
Figure 47: PASS Main Program.....	64
Figure 48: PASS Interrupt Service Routine	64
Figure 49: PASS through Program	67
Figure 50: Multiplier Example	71
Figure 51: Sine wave generation	72
Figure 52: Sine Wave Program.....	72
Figure 53: Sine Wave Generator Settings	73
Figure 54: Sine Wave Processor Initialisation	74
Figure 55: Main program	74
Figure 56: Sine Wave ISR	74
Figure 57: Register Based MAC	76
Figure 58: A musical note	79
Figure 59: Musical Note frequencies and octaves	80
Figure 60: Coefficients for Octave 6 frequencies.....	80
Figure 61: Octave 6 program flow.....	81
Figure 62: Octave 6 musical notes data table.....	82
Figure 63: Interrupt Settings and Processor Initialisation	82
Figure 64: Note selection in transmit ISR	83
Figure 65: Musical note generation	84
Figure 66: DELAY subroutine	85
Figure 67: 'C5X DSK RS232 connections	87
Figure 68: Serial transmission routine	91
Figure 69: Serial reception routine.....	92
Figure 70: Flow of PC_TUNE assembly program	94
Figure 71: FIR Filter Equipment.....	98
Figure 72: A high pass filter and its effect on a signal	99
Figure 73: A sinusoid shifted by 90 degrees in phase	100
Figure 74: FIR Filter.....	101
Figure 75: FIR Flow Chart.....	102
Figure 76: FIR Coefficients	103

Figure 77: FIR ISR Flow Chart	104
Figure 78: FIR ISR	104
Figure 79: The Multiplication Process	105
Figure 80: Amplitude Modulator Equipment	108
Figure 81: Operation of an AM	109
Figure 82: Frequency Spectrum	109
Figure 83: Amplitude Modulator Flow Chart	110
Figure 84: AM Filter coefficients	111
Figure 85: Amplitude Modulator ISR	112
Figure 86: Reverberation Equipment	113
Figure 87: An Echo	114
Figure 88: Continuous Echo	115
Figure 89: Echo Generator Implementation	116
Figure 90: Reverberate Flow Chart	117
Figure 91: Reverberate Settings	117
Figure 92: Reverberator Initialization	118
Figure 93: Circular Buffering	119
Figure 94: Receive Source Code	120
Figure 95: Music Player Equipment	123
Figure 96: Sound Recorder Equipment	124
Figure 97: Function Generator Equipment	126
Figure 98: DTMF Equipment	128
Figure 99: Spectrum Analyser Equipment	130
Figure 100: Oscilloscope Equipment	132

List of Tables

Table 1: Internal Registers.....	22
Table 2: Memory-Mapped Core Processor Registers.....	23
Table 3: 'C50 interrupts.....	24
Table 4: 'C50 Interrupt context save.....	24
Table 5: Memory-Mapped Peripheral Registers.....	26
Table 6: Dedicated Register Addressing Instructions.....	31
Table 7: Bit functions of CBCR.....	33
Table 8: PMST Register.....	54
Table 9: On-chip Single Access RAM configuration control.....	54
Table 10: TCR Register.....	57
Table 11: Delay values for serial communications.....	91

1. Welcome to the DSP Starter's Kit

1.1 Digital Signal Processing

1.1.1 Overview

Digital Signal Processing is defined as:

“The science concerned with the representation of signals by sequences of numbers and the subsequent processing of these number sequences.”

Processing involves either extracting certain parameters from a signal or transforming it into a form that is more applicable. The processors that perform such operations are known as **Digital Signal Processors**.

1.1.2 Analogue Computers

The human brain is an extremely powerful and complex analogue computing device which performs sophisticated analysis procedures. The brain interprets a vast array of external stimuli against pre-determined characteristics to produce an intelligent response. The nature of the brain's processor enables humans to complete a wide variety of tasks as they happen in "real-time". A good illustration of this is driving a car where the driver must co-ordinate the steering wheel, pedals and switches of the vehicle according to the changing road conditions. It is imperative that this is done as fast as the environment changes to ensure that the vehicle stays on-track. Although the brain is powerful, it does have limited computational speed and it is also reluctant to perform repetitive tasks. There are many instances when we may want a machine to take decisions for us based on its own sensor readings.

Engineers have become experienced at detecting the continuous analogue signals which are present within the natural environment, many of which humans cannot sense. Problems arise when it is necessary to interpret these signals intelligently. Computers offer the solution, but unfortunately analogue computers are not as versatile as their digital equivalents. Once a continually varying analogue signal is detected and represented in a digital form (sampled), processing is relatively easy.

1.1.3 Real-Time Processor Performance

The semiconductor market contains many different types of processor, all with different levels of performance. Processors such as those found within a PC are excellent at processing database and spreadsheet type applications. However, these type of processors are less effective at processing real-time signals. The latest generation of 586 based processors (Pentiums) are capable of implementing signal processing algorithms but in majority of cases they cannot achieve these tasks in real-time. For example, when decompressing video images, each video frame must be decompressed before ending the display of the previous frame order to prevent the picture looking jerky. Although PCs and conventional processors can decompress such images, they cannot decompress the frames sufficiently fast to make the motion appear smooth.

To meet the demands of real-time signal processing, DSPs have special architectural features designed onto silicon to enhance speed and efficiency. These features enable Digital Signal Processors to provide the necessary computational processing power for dealing in real-time with complex mathematical algorithms such as image decompression.

There is a big performance gap between the conventional processors and DSPs when benchmarked against signal processing applications. Even a low-end DSP can be ten times faster than an 8-bit microcontroller at processing signals. When DSPs and conventional processors are compared like for like, DSPs will always out-perform conventional processors in signal processing applications.

1.2 Why Digital?

Electronic analogue computers are not as versatile as their digital counterparts. On the other hand, digital computers can easily be configured to perform a variety of tasks such as word processing and spreadsheet formatting as seen with a PC.

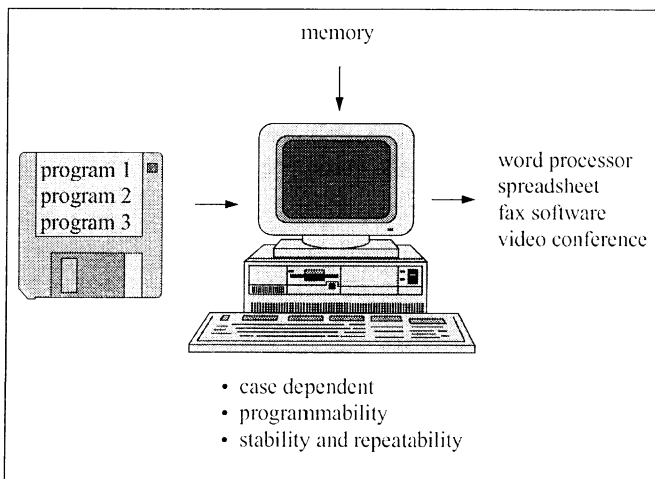


Figure 1: Advantages of a Digital Solution

When considering signal analysis in analogue systems, functions like filtering can be implemented using discrete analogue components such as inductors, capacitors and resistors. The long term stability of an analogue system is not constant and as a result the filter characteristics change. Digital implementation of signal processing has several distinct advantages:

- It is possible to accomplish many tasks inexpensively that would be either difficult or impossible with analogue electronics. For example, speech coding and Fourier Transforms.

- Digital systems are insensitive to environmental changes. Analogue components change values with temperature, humidity and age. By contrast, every digital system produced will give exactly the same result - thus aiding manufacture.
- Insensitivity to component tolerances means that, unlike analogue systems, two DSPs will give an identical result every time, ensuring predictability and repeatability. Significant production costs are incurred with analogue devices as each system must be individually tuned.
- Re-programmability enables a processor to be upgraded or reconfigured giving alternative functionality at little cost. For example, a digital filter can be reconfigured from a high pass to low pass characteristic by loading in new software.
- Design times are reduced within the digital domain as DSP processors are programmable.
- Chip areas are small and stay constant while additional features are added to the processor within software. Increasing the order of an analogue filter, for example, requires more components, whereas digital filters require only a few additional program instructions.

1.3 The role of DSP

As we have already seen, DSPs are microprocessors that are specialised at performing well in real-time digital signal processing applications. DSPs bridge the gap between Application Specific Integrated Circuits (ASIC) designs and general purpose microcomputers. For many engineers, DSPs combine the speed flexibility inherent in ASIC with the convenience of an off-the-shelf programmable processor.

In real-time digital signal processing the main concern is with the amount of processing that can be completed within a given time. The real-time processing power which DSPs offer makes a wide range of applications within the telecommunications, industrial and consumer markets feasible. The vast scope of end applications means that any DSP family must be extensive, offering a wide selection of processing characteristics. DSPs can be classified in the following ways:

- **Embedded DSPs** within automotive, consumer and telecommunications products which:
 - are inexpensive,
 - have high volume applications - hard disk drives,
 - feature cost and integration as significant factors,
 - feature low power consumption - mobile applications,
 - place less importance on ease of development and performance.

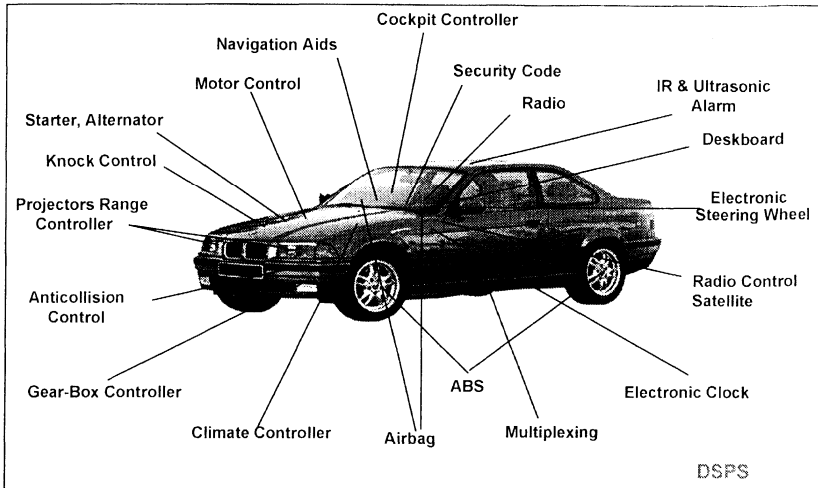


Figure 2: Embedded DSPs within a car

- **High Performance DSP systems** which:
 - have large data volumes and/or complex algorithms -such as in radar,
 - have low production volumes,
 - feature significant processing power,
 - have a multiprocessing environment,
 - are easy to use.



Figure 3: Video

- **PC Based DSPs** for use on video and sound multimedia systems which:
 - are low-cost,
 - offer high performance,
 - enable high integration,
 - are capable of multitasking.

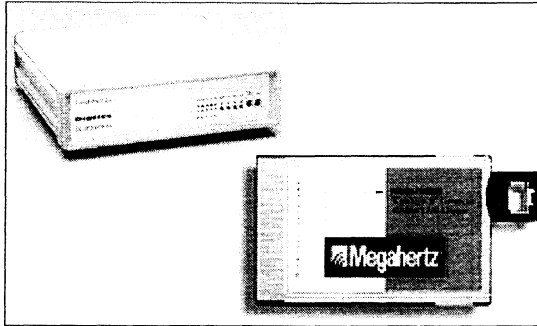


Figure 4: Modem

1.4 Texas Instruments DSP

1.4.1 TMS320 Family

The roots of Digital Signal Processing lie in the seventeenth and eighteenth centuries when mathematicians first developed the algorithms that modern day processors execute. Filtering, transforms and speech coding principles were all developed before the technology was available to implement them. This made the first digital applications highly specialised, with an associated cost that could rarely be justified.

The advent of the low-cost single chip DSP device from Texas Instruments changed the situation. In the early 1980s the TMS320C10 processor from TI enabled the relatively simple implementation of high performance digital filters and other mathematical functions which had real-time applications. The TMS320C10 won the Electronic Products Magazine "Product of the Year" award in 1982.

As demands from users have increased, manufacturers such as Texas Instruments have responded by providing ever more powerful processors at lower costs. In 1995 TMS320C80 with its highly advanced architecture and 7 separate processors on a single piece of silicon won a 'Byte Award'. Texas Instruments launched the world's very first DSP and since then has pioneered the way with innovation and foresight.

The TMS320 family consists of different types of DSP processor which are suited to a wide range of applications. The family covers many different applications from low-end embedded DSPs used in automotive systems to telecommunications chip-sets used in

applications such as mobile phones. Some examples of the wide range of applications where TI's family members are used include:

- Music Systems - CD players,
- Toys - sound synthesis,
- VideoPhones - compression techniques for low transmission rates,
- Modems - high baud rates making Internet connections effective,
- Telephone - voice mail, mobile communications and answering machines,
- 3D graphics - real-time computation of graphics for virtual reality,
- Image Processing - finger print recognition, visual inspection and interpretation.

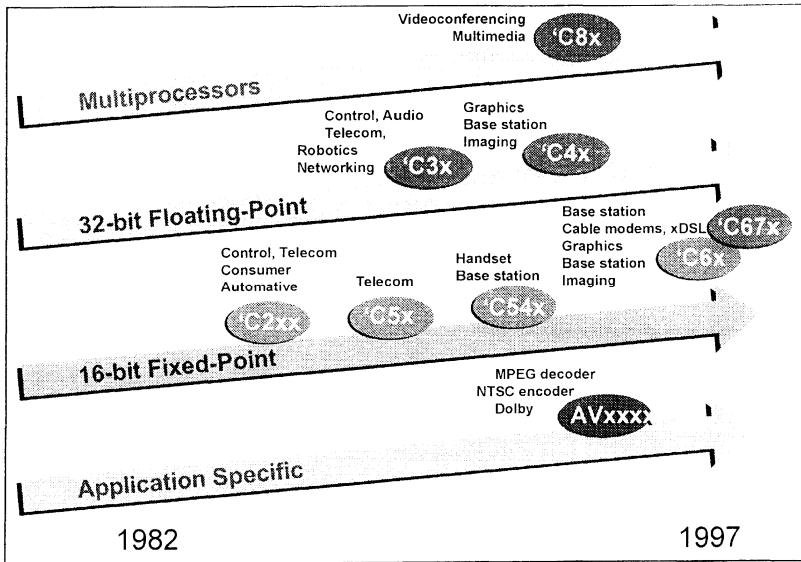


Figure 5: TMS320 Family

The TMS320 family consists of 12 generations of processor, each with many family variations which provide different combinations of peripherals, memory and packaging options to suit their design.

The trends in DSP technology are lowering system cost, increasing processing power, reducing power consumption and saving board space. Such developments are making the DSP more viable in everyday applications. For this reason, it is important for design engineers not only to be aware of DSP but to be competent when developing DSP applications. DSPs will continue to play a central role in an expanding range of products in a way similar to that of microprocessors and microcontrollers.

1.4.2 Texas Instruments DSP Firsts

Since the launch of TMS320C10, Texas Instruments has produced a number of DSP 'firsts', leading innovation, encouraging new applications, creating new markets. Most of these 'firsts' helped to turn new technologies into real applications. Here are a few of them;

1981	TI establishes DSP University Program to support interest in DSP technology
1982	TI introduces the first programmable general-purpose DSP - the TMS320C10 operating at 5 to 10 million operations per seconds. Targeted markets include modems and defence.
1984	TI introduces a second generation DSP, the TMS320C2x
1985	TI manufactures the first DSP using CMOS process technology TI introduces a technical hotline taking the lead in offering support to designers
1987	First DSP (TMS320C17)s used in consumer toys Published text book - Digital Signal Processing with the TMS320 family
1988	TI launches new range of processors for high performance applications World's first DSP-based hearing aid uses the TMS320C1x
1989	TI launches a new generation of processor - TMS320C5x
1990	TI offers the first DSP C-source debugger and optimising ANSI C compiler TI launches the first starter kit based on the TMS320C2x generation
1991	TI lowers the price of the TMS320C1x family to compete with microcontrollers TI sponsors the First Educators' Conference for DSP educators and researchers
1993	TI publishes "A Simple Approach to Digital Signal Processing" text book TI forms the Software Co-operative with over one hundred Third Parties
1994	TI launches the second starter kit based on the TMS320C5x processor
1995	TI launches the Elite Lab. program to award distinguished electrical engineering faculties with DSP tools TMS320C2xx generation is launched achieving a new price-performance ratio in the industry TMS320C54x generation is launched, aimed at telecommunications applications such as GSM TI launches a \$100,000 (US) world-wide competition within universities for DSP applications
1996	TI publishes "DSP Teaching Kit" for Universities and self learning
1997	Most powerful DSP even designed: TMS320C6x delivers 1600 MIPS Most powerful floating-point DSP Core: TMS320C67x delivers 1600 MIPS

1.5 DSP systems

1.5.1 Overview

The natural environment which we wish to interpret is analogue. Signals vary continually with time and may take any value. The digital domain is fixed between two values, high and low. Conversion from the analogue domain to the digital domain (and back) is vital for sensing signals, manipulating them and reintroducing them to the

environment. This may seem troublesome, but the advantages gained from digital manipulation justifies the conversion. Special devices which perform this conversion are manufactured especially to interface with DSP processors.

1.5.2 Typical System

Typical Digital Signal Processing systems are commonly made up of the following components:

- Analogue to Digital converter, A/D,
- Digital to Analogue converter, D/A,
- Memory, RAM,
- Digital Signal Processor, DSP,
- Memory, ROM.

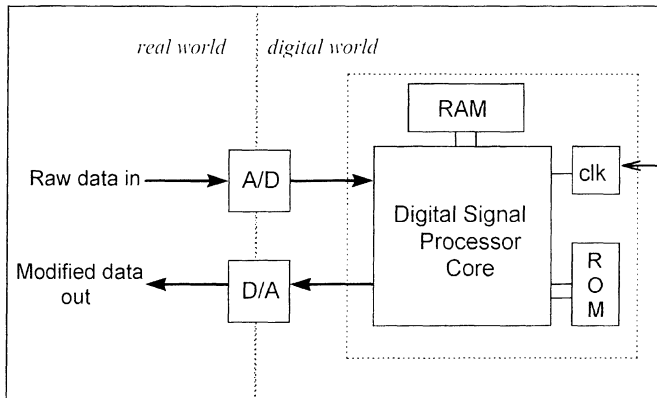


Figure 6: A typical DSP System

Figure 6 shows a typical DSP system. Let us follow a signal from the real world into the DSP system. A continuous analogue signal, such as the output of a microphone, is fed into the analogue to digital converter or A/D. The A/D performs sampling and digitisation on the input signal. Sampling consists simply of taking snapshots of a continuous signal at regular intervals. These data snapshots are then converted into a series of '0's and '1's to digitally represent the original signal. The rule in sampling is that the samples must be taken at a rate which is at least twice the frequency of the signal. Usually the sampling rate is set at a frequency slightly higher than twice the signal frequency to ensure that the signal is correctly represented.

1.4.2 Texas Instruments DSP Firsts

Since the launch of TMS320C10, Texas Instruments has produced a number of DSP 'firsts', leading innovation, encouraging new applications, creating new markets. Most of these 'firsts' helped to turn new technologies into real applications. Here are a few of them;

1981	TI establishes DSP University Program to support interest in DSP technology
1982	TI introduces the first programmable general-purpose DSP - the TMS320C10 operating at 5 to 10 million operations per seconds. Targeted markets include modems and defence.
1984	TI introduces a second generation DSP, the TMS320C2x
1985	TI manufactures the first DSP using CMOS process technology TI introduces a technical hotline taking the lead in offering support to designers
1987	First DSP (TMS320C17)s used in consumer toys Published text book - Digital Signal Processing with the TMS320 family
1988	TI launches new range of processors for high performance applications World's first DSP-based hearing aid uses the TMS320C1x
1989	TI launches a new generation of processor - TMS320C5x
1990	TI offers the first DSP C-source debugger and optimising ANSI C compiler TI launches the first starter kit based on the TMS320C2x generation
1991	TI lowers the price of the TMS320C1x family to compete with microcontrollers TI sponsors the First Educators' Conference for DSP educators and researchers
1993	TI publishes "A Simple Approach to Digital Signal Processing" text book TI forms the Software Co-operative with over one hundred Third Parties
1994	TI launches the second starter kit based on the TMS320C5x processor
1995	TI launches the Elite Lab. program to award distinguished electrical engineering faculties with DSP tools TMS320C2xx generation is launched achieving a new price-performance ratio in the industry TMS320C54x generation is launched, aimed at telecommunications applications such as GSM TI launches a \$100,000 (US) world-wide competition within universities for DSP applications
1996	TI publishes "DSP Teaching Kit" for Universities and self learning
1997	Most powerful DSP even designed: TMS320C6x delivers 1600 MIPS Most powerful floating-point DSP Core: TMS320C67x delivers 1600 MIPS

1.5 DSP systems

1.5.1 Overview

The natural environment which we wish to interpret is analogue. Signals vary continually with time and may take any value. The digital domain is fixed between two values, high and low. Conversion from the analogue domain to the digital domain (and back) is vital for sensing signals, manipulating them and reintroducing them to the

environment. This may seem troublesome, but the advantages gained from digital manipulation justifies the conversion. Special devices which perform this conversion are manufactured especially to interface with DSP processors.

1.5.2 Typical System

Typical Digital Signal Processing systems are commonly made up of the following components:

- Analogue to Digital converter, A/D,
- Digital to Analogue converter, D/A,
- Memory, RAM,
- Digital Signal Processor, DSP,
- Memory, ROM.

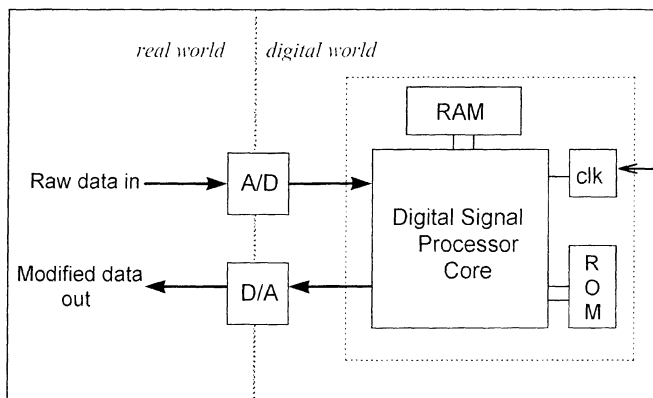


Figure 6: A typical DSP System

Figure 6 shows a typical DSP system. Let us follow a signal from the real world into the DSP system. A continuous analogue signal, such as the output of a microphone, is fed into the analogue to digital converter or A/D. The A/D performs sampling and digitisation on the input signal. Sampling consists simply of taking snapshots of a continuous signal at regular intervals. These data snapshots are then converted into a series of '0's and '1's to digitally represent the original signal. The rule in sampling is that the samples must be taken at a rate which is at least twice the frequency of the signal. Usually the sampling rate is set at a frequency slightly higher than twice the signal frequency to ensure that the signal is correctly represented.

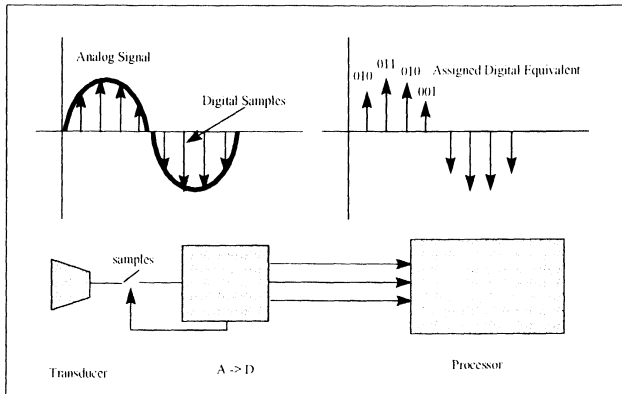


Figure 7: Sampling Process

Figure 7 shows the sampling process of a sinusoidal input wave form. At regular time intervals, the sine wave is sampled. A binary code is then assigned to the sample, reflecting its magnitude. In this example a three bit code is used. This series of digits (010 011 010 001) now represents half a period of the sine wave. By representing the other half of the period in the same way a full digital representation of the signal is created.

Following A/D conversion, the digitised input signals are read into the processor and modified according to a set of predefined rules which is known as the program. With an architecture suited for very fast calculations, the DSP is capable of reading, modifying and outputting a fast stream of input data.

After processing, the digitised signal is passed out of the processor to a digital to analogue converter or D/A. This device converts the digital code from the DSP back to an analogue equivalent, effectively the opposite of A/D. The signal is now ready for the 'real' world.

In most cases, it is the accuracy and speed of A/D and D/A converters that determine the performance and quality of many DSP systems.

While the three core components of a digital system have already been described, supporting devices are also essential. In Figure 6 the block RAM (Random Access Memory) represents volatile memory only holding data when the power is on. A DSP system may have a number of such blocks. Memory is used to hold program instructions, data and even coefficients which are all used within program execution.

ROM (Read Only Memory) is a permanent location for program and data values. ROMs hold data even when the power is turned off. Many DSP based systems use an EPROM which is an Electrically Programmable ROM for storing information. When the DSP is powered up, the EPROM contents are loaded into the DSP RAM. The program then executes from the on-chip RAM. This ensures faster execution of instructions as access to on-chip RAM is faster.

Some DSPs have on-chip ROM which can be used to store programs. Once the software is tested and proven, it can be put into DSP's on-chip ROM by using a mask during the manufacturing process. This method is only cost justifiable for high volume production. It reduces the required board area and the number of external components in a given design.

All DSP systems require a clock which provides timing for internal operations of the processor. This is represented by 'clk' in Figure 6. The DSP's actions are governed by the rising and falling edges of the clock. The speed of the clock is typically in the MHz region.

1.6 Welcome to the TMS320C5x DSK

1.6.1 Overview

After their initial launch in 1990, Texas Instruments' family of DSP Starter's Kits (DSK) have proved themselves to be an essential industry. In the past, evaluating a DSP required a significant investment for an entry-level hardware-development platform. Now the ground breaking DSK is making DSP accessible to everyone for only US\$99.

The simple, useful hardware provides the user with the freedom to create and run programs. DSK may also be easily expanded as all signals are on accessible pin headers. The kit combines assembler and debugger software to provide an easy-to-use development environment. Some of the TMS320C50 DSK's features include:

- 40 MHz TMS320C50 DSP with 10k x 16 words of on-chip RAM,
- TLC320C40 Analogue Interface Circuitry (AIC) with 14 bit resolution (single chip A/D + D/A),
- RS232 serial port for PC connection,
- Two standard RCA jacks for direct communication to analogue devices,
- On-board EPROM that allows the DSK to communicate with a PC.

1.6.2 A walk around the DSK board

Figure 8 shows the DSK board. A more detailed circuit diagram is included in Appendix A of **DSK User's Guide** that is supplied with the DSK. This is useful when expanding memory or adding new peripherals to DSK.

Let us now look at each component of the DSK.

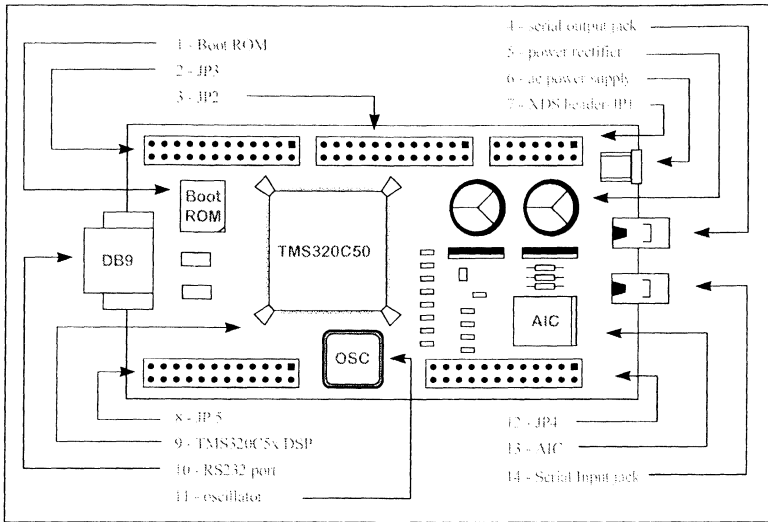


Figure 8: DSK Board Layout

1 - Boot ROM

The Boot ROM is an external source of program information.

The DSK has a 32K x 8-bit ROM that stores the instructions necessary to interface the DSK with a host PC for software loading and running the debugger. This program is called the 'communications kernel' or 'kernel' for short. The kernel program is loaded into RAM on boot-up, a process that is initiated by the DSP chip on reset. After downloading its contents to the processor, the Boot ROM cannot be accessed.

We shall talk about the boot-up sequence in more detail at the end of this chapter.

2 - JP3

The JP3 header is a set of 24 pins (12 x 2) that provide the DSK user with easy access to the address lines and a number of interrupt signals of the TMS320C50. The address lines are used for reading and writing information on and off-chip. The TMS320C50 device has 224K x 16 bit maximum addressable external memory space. This external memory space is organized as 64K program, 64K data, 64K peripheral and 32K global memory.

Access to these pins means that the DSK is an expandable platform for larger DSP applications. External memory and other peripherals can be built on a separate board and linked to DSK via the various JP headers.

3 - JP2

The 24 pin header labelled JP4 provides access to data lines (D0-D15) of the processor and a number of control signals. This header would be used together with other pin headers for board expansion.

4 - Serial Output Jack - OUT

The RCA jack connects the output of the D/A converter with the real world. This analogue interface enables a simple standard link to be made between the DSK and laboratory equipment such as a Cathode Ray Oscilloscope (CRO) or an amplified speaker. The signal is sufficient to give an audible output from a small, reasonably sensitive 8Ω loudspeaker. Please note that the serial output jack is labelled on the PCB as "OUT".

5 - Power Rectifier

The board is supplied with 9V AC linear power rectification circuitry that supplies the DSP and auxiliary units with the correct power supply.

Input:	9V ac 250mA
Output:	$\pm 5V$ dc regulated
	$\pm 12V$ dc unregulated

6 - AC power supply Input

The DSK requires a 9V ac power supply which must be supplied via a 2.1 mm jack. We recommend* the output should be:

9V ac
250 mA min
50/60 Hz

7 - XDS Header - JP1

The DSK also provides designers with a facility to use TI's Extended Development System (XDS) to emulate the processor. The XDS performs boundary scan emulation through JTAG. This type of facility, available on such a low-cost product is unprecedented within the industry.

JTAG scanning logic is for testing and emulation purposes only. It can be used to test pin to pin connectivity as well as performing operational tests on peripheral devices surrounding the TMS320C50. JTAG also interfaces with other internal scanning logic circuitry providing access to all of the on-chip resources. The emulation port conforms to a subset of IEEE 1149.1 JTAG standard.

8 - JP5

The JP5 header is used to access the processor's handshaking and clocking signal pins that are used for interfacing external devices to the board.

* Texas Instruments will not accept liability for any damage caused by an incorrect power supply.

9 - TMS320C50

The TMS320C50 is the DSP processor at the heart of the DSK. We shall talk about the architecture and properties of 'C50 later on in this chapter. The processor is supported by auxiliary devices on the board such as an oscillator, A/D D/A converters and a power rectifier. Although there is no external memory on the board, the 10K on-chip memory (RAM) is sufficient to run many DSP applications software.

10 - RS232 Port

The DB9 RS232 connector mounted on the board connects the DSK with a PC via a standard RS232 cable. The cable should use straight connections and no cross over is required. The link is used for downloading data to the processor, for debugging and can be programmed as a two way link with the PC to exchange program data, for a graphic display or user interface.

RS232 requires: TX, TR, DTR, Ground
 $\pm 5V$ min into 600Ω

DSK User's Guide has a full description of the port. We shall talk about the operation of this port in detail in a later chapter.

11- Oscillator

The system clock provides the processor with its timing signal. It is the heart beat of the processor and runs at 40MHz.

12 - JP4

The 24 pin header provides access to control and data signals for the serial port. This pin header would typically be used for building multiprocessor 'C50 systems and for extending the capabilities of the existing serial port.

13 - Analogue Interface Circuitry - AIC

The AIC provides the necessary conversion between the analogue and digital domain. It incorporates A/D, D/A and filters, all on a single chip. With a maximum sampling rate of 19.2kHz, the TLC320C40 is a general purpose device that is suited for applications such as telecommunications, data capture and voice processing.

Most functions of the AIC are configurable in software with the AIC usually connecting to the serial port of the 'C50. The DSK User's Guide contains a data sheet for the AIC within Appendix B. We shall be using and programming the AIC extensively.

14 - Serial Input Jack - IN

The RCA jack connects the input of the A/D converter with the real world. The analogue interface enables a simple standard link to be made between the DSK and either laboratory equipment, such as a signal generator, or a microphone. In some cases microphones do not generate the signal level necessary for the A/D to detect and convert the signal. Using a dynamic or pre-amplified microphone ensures that the appropriate signal level is achieved. Please note that the serial input jack is labelled on the PCB as "IN".

1.7 Digital Signal Processors

1.7.1 Overview

All digital processors consist of several fundamental modules:

- a processing unit to perform mathematical and logical operations,
- memory to store data and program instructions,
- a bus structure for efficient transfer of data and program.

As for any stored-program machine, a processor must be told what operation to perform in every machine cycle. Typically, a processor fetches an instruction and some data from memory, operates on this and then returns the manipulated data to storage. Instructions, data fetches and returns are performed in different ways by processors with differing architectural structures. Primarily two architectures are prominent within the semiconductor industry: Von Neumann and Harvard. It is the application that usually governs the type of architecture to be employed together with memory and peripheral requirements.

1.7.2 Architecture Types

It is the Von Neumann architecture that has set the standard for computer development over the past forty years. Essentially, the architecture is very simple with both program and data residing in the same memory mapped space. Figure 9 shows the organisation of a typical Von Neumann architecture processor.

The disadvantage with this architecture is that there is only one communication channel to memory, memory holds both data and program words. Therefore to access data, first the instruction must be fetched, then the processor can fetch data. Therefore data accesses take a minimum of two machine cycles. Von Neumann architecture forms the basis of many processors such as the 80x86 and 68xxx ranges which are more suitable for general purpose computing.

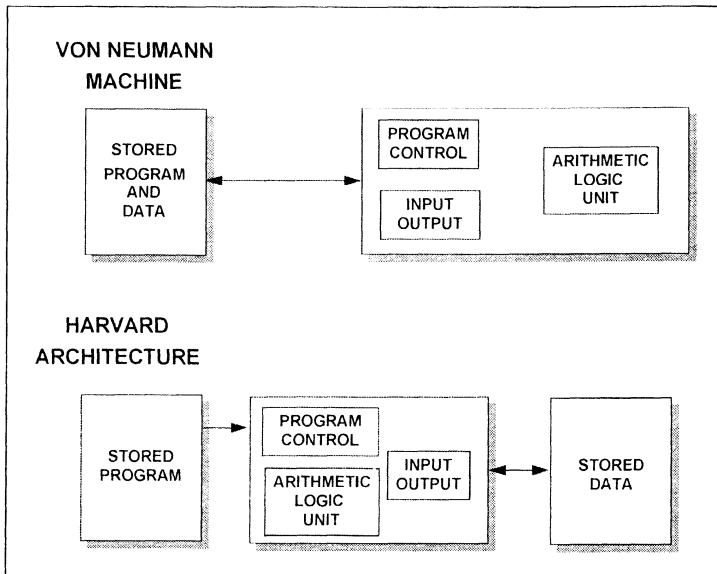


Figure 9: Von Neumann and Harvard Architecture

Where fast data manipulation is vital, accessing both program and data memory in a single machine cycle is advantageous. In such an architecture, data accesses do not need to wait for instruction fetches. This type of architecture, where program and data memory are separate, is named Harvard. The lower portion of Figure 9 shows the structure of Harvard architecture. Having two buses to serve each address space ensures that data and program accesses occur in parallel, increasing the operational processing speed. Unfortunately, any increase in processing power comes with a cost penalty. Two memory spaces require twice as many addresses and data pins. The more the pins on the chip, the higher the cost of the chip. An elegant solution is to make the processor internally Harvard but use a single data and address bus to access the external world. In other words, internally have all the advantages of the Harvard architecture but externally reduce the number of pins used. Many of the DSP processors supplied by Texas Instruments use this "modified" Harvard architecture to reduce cost for the customer while maintaining speed within the processor.

Some of the more common features within a typical modified Harvard architecture DSP chip are listed below:

- Internally separate program address, program data, data and data address buses,
- A central arithmetic unit where mathematical and logical operations are performed,

- An auxiliary arithmetic unit which is used to perform mathematical and logical operations whilst the central arithmetic unit is occupied,
- Serial ports for high speed communication with other DSPs, data converters and the outside world in general,
- On-chip data and program memory, avoiding frequent accesses to external memory. External memory generally takes longer to access which slows down the operational speed of the DSP.

1.7.3 The need for speed

The main concern for signal processing and real-time algorithms is the amount of processing that can be achieved in a given period of time. Most signal processing algorithms involve a multiply and an add operation which can be written as follows in its general form:

$$A = BC + D$$

This operation is typically named as MAC, Multiply and ACcumulate. The addition function is quite simple and can be performed in a single clock cycle. The same is true of subtraction. Where most computers subtract by negating one number and then adding it to the other. However multiply functions take much longer. A general purpose processor may take several hundreds of clock cycles to implement such a calculation. The reason behind this is that most general purpose processors implement multiply as a series of add instructions.

What is needed is a machine which can perform a multiply and an add in just one clock cycle. For this the processor would need to have a hardware multiplier capable of producing a result in a single machine cycle. Further, the processor should be capable of performing an add operation in the same cycle. DSPs have hardware multipliers and are hardwired for completing a multiply and an add within a single clock cycle. It is this silicon base that allows the execution of MAC instruction in a single clock cycle in DSPs giving the real-time performance advantage.

Pipelining is an additional method of speeding up the instruction throughput of a processor. The simplest analogy is that of a car production line. It might take 10 hours to assemble a complete car, but because the construction of the car is broken down into a number of subsections, a car might be finished every 10 minutes. Similarly, processor instructions can be broken down into stages such as fetching the instruction, decoding the instruction, fetching any data, executing the instruction, and storing the result. The net result is that the core processor never needs to wait for an instruction fetch once the pipeline is full.

1.7.4 Fixed-Point and Floating-Point

DSP processors can be categorised into two distinct groups: floating-point and fixed-point. Each has a different architecture which is beneficial for different applications.

- Fixed-point DSP processors represent a number in a fixed range with finite precision. For example, a 16 bit processor will give $\pm 2^{15}$ range and a precision of 1

in 32768. The earliest DSPs were based around this technology and for the majority of applications today, the industry chooses 16 bit fixed point processors. The price advantage of fixed point 16 bit processors is significant.

- Floating-point DSP processors are much more recent. They express numbers as a mantissa lying between +1.0 and -1.0 combined with a scaling function called the exponent. This method of representation gives a greater dynamic range and therefore reduces the chance of overflow. Signal processing algorithms are much easier to implement on floating-point processors and are therefore more suited to optimising high level language compilers. However, this type of processor tends to be slower and more expensive than its fixed-point counterpart.

1.7.5 Typical Fixed-Point DSPs

A typical fixed-point DSP consist of a core unit, program memory, data memory and peripherals. A simplified block diagram of such a DSP is shown in Figure 10. The internal architecture of the processor is Harvard because there are separate program and data memory areas.

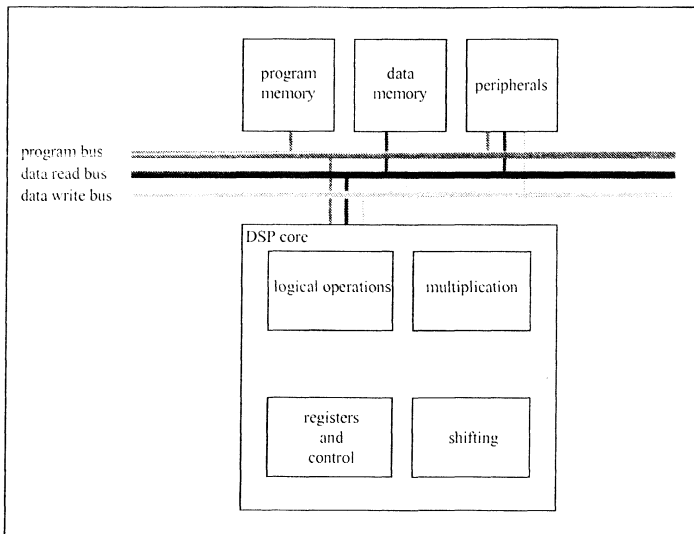


Figure 10: Typical Fixed-Point DSP

Essentially the processor is split up into two parts; the memory and the core. Memory is divided again into program and data memory. Program memory holds the series of instructions that the processor executes. Data memory holds constants, coefficients and the results from program execution. Splitting memory into two such blocks means

that the two areas can be accessed within the same clock cycle, increasing the processor performance significantly.

The core is the heart of the DSP. All the calculations such as adding, subtracting, multiplication, shifting and logical operations are performed in the core. Most DSP cores have a large degree of parallelism whereby a number of operations can be performed simultaneously.

The internal hardware of the DSP executes functions that other processors typically implement in software or in micro-code. For example, the device contains hardware for single cycle 16 x 16-bit multiplication, data shifting and address manipulation. This hardware-intensive approach provides the processing power previously unavailable on a single chip.

The constituent parts of the processor are all connected together by a series of wires which are referred to as the buses. In modified Harvard architecture, DSPs program and data memory require two separate buses, one for each area of memory. Figure 10 depicts such a configuration. Data memory in this particular case has two buses: data read and data write. This structure is another speed enhancing feature of the Harvard architecture. For a memory which has dual access capability, these buses are used to read from and write to the same memory in the same instruction cycle. The program bus channels instructions from program memory. The bus structure also connects the core and memory to on-chip peripherals.

Peripherals connect the processor to the off-chip digital and analogue devices. Providing the link to other processors or data converters is essential. It is this link that supplies the core with the data input and sends modified data out. Different DSPs have a different set of peripherals, depending on the target application. Most DSPs have a peripheral to connect to data converters so that sampled data can be passed in and out of the processor with ease.

1.8 A walk around the TMS320C5x Processor

1.8.1 Overview

The DSK features the TMS320C50 16 bit fixed-point DSP. The processor offers unprecedented affordability, performance, integrated memory and power management features which have led it to become an industry standard. It is in millions of everyday products such as cellular telephones and modems.

- **Key Specifications**
- 40 MIPS performance
- Flexible power down modes
- Compatible with TMS320C1x/2x/2xx
- On-chip emulation
- Enhanced instruction set
- 10K words RAM
- **Key Applications**
- Cellular/cordless phones
- High speed modems
- Personal communications
- Sound systems, voice processing
- Laser printers and copiers
- Multimedia applications
- Hard disk drives

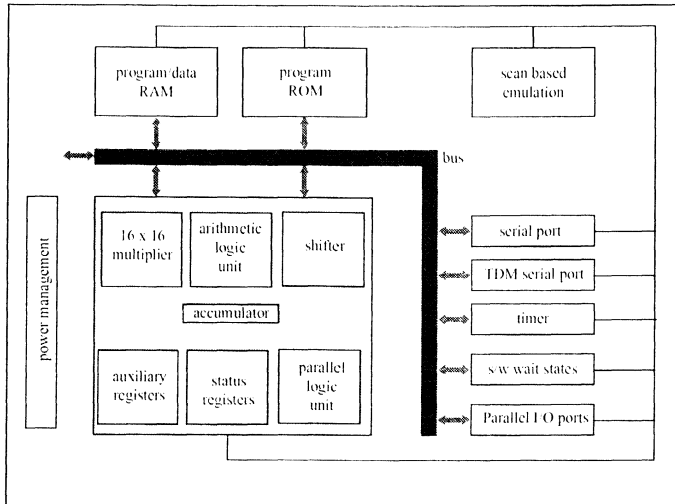


Figure 11: TMS320C50

1.8.2 Memory

Figure 11 shows the simplified internal structure of TMS320C50.

Program/Data RAM

The contents of RAM can be both read from and written to an infinite amount of times, making the RAM ideal for storing data and temporary program information. Having on-chip RAM offers a significant performance advantage as it is faster to access on-chip memory than going off-chip.

Certain sections of RAM can be configured as program or data memory. This feature offers another significant advantage to the user where depending on the application, a smaller RAM area can be allocated for program and a larger section for data or vice versa. It is such flexible features that make the 'C50 popular in many applications.

A section of on-chip RAM is designed as Dual Access RAM (DARAM). This area can be written to and read from simultaneously in a single clock cycle. In some signal processing applications, such a feature has a significant impact on the processing speed of algorithms.

Usually on start up, RAM is downloaded with the necessary configuration information, program and data from an external device. Then the program starts executing from on-chip RAM.

Program ROM

'C50 has 2K words of boot ROM. The boot ROM includes a divide functionality test and boot code. Other members of the 'C5x family have differing sizes of on-chip ROM and some have maskable ROM which is used to hold application software. It saves additional board space required for external ROM. Such designs also are more compact and use less power. Once the application has been developed and tested, the software can be sent to Texas Instruments to be masked onto the ROM. The 'C5x is then manufactured with the application on-chip. This approach is suitable for very high volume production.

1.8.3 The Core

The enhanced TMS320C50 core retains compatibility with other members of the TMS320 family, namely the TMS320C1x and TMS320C2xx generations. The core is capable of performing high-speed arithmetic executions within a short cycle because of its parallel architecture, having the following main components:

16 x 16 multiplier

The fast on-chip fixed point multiplier allows the device to perform fundamental DSP operations such as convolution, correlation and filtering with optimum efficiency. The multiplier unit performs a 2's complement 16 by 16 multiplication of two binary numbers, which yields a 32 bit signed result. The unit has three major elements:

TREG	- a temporary register that holds the multiplicand
PREG	- a product register that stores the product
multiplier array	- the unit that performs the calculation

Arithmetic Logic Unit (ALU)

The 32 bit ALU together with the accumulator performs general purpose 2's complement arithmetic operations such as addition and subtraction as well as Boolean operations like ANDing and ORing. The output of the ALU is passed to the accumulator.

Shifter

The shifter unit's input is connected to the data bus and the output is connected to the ALU. The unit provides both left and right shifts, filling the least significant bits with zeros. The device is essential when using number formats such as Q15, and to prevent overflow. We shall consider Q15 format in the next chapter.

Accumulator

The accumulator is a general purpose register that serves as a 32 bit storage facility. The accumulator is a universal storage register between the multiplier, the memory, the ALU and the shifter. For fast temporary storage, the accumulator has an associated buffer which is also 32 bits in length.

The accumulator is accessible in two parts, the upper half (ACCH) and the lower half (ACCL) which gives the programming flexibility.

Auxiliary Registers

The TMS320C50 has 8 auxiliary registers labelled AR0 to AR7. They can either be configured as temporary storage or for indirect addressing. Indirect addressing uses the auxiliary registers as pointers to areas of memory. They can then be incremented or decremented with relative ease, allowing a series of sequential memory accesses. Auxiliary registers also have an Auxiliary Register Arithmetic Unit (ARAU), allowing parallel address calculations while the central ALU is busy with other operations.

Status and Control Registers

The TMS320C50's four status and control registers hold and toggle settings which govern the functionality of the processor:

- Status Register 0 (ST0),
- Status Register 1 (ST1),
- Processor Mode Status Register (PMST),
- Circular Buffer Control Register (CBCR).

Different modes can be switched on and off by setting bits within the various registers to one or zero. The registers can be directly written to, enabling the whole processor to be reconfigured efficiently. TMS320C5x User's Guide has detailed information on bit settings in these registers in Chapter 3. We shall explain these bit settings in later chapters as we use them in our programs.

Parallel Logic Unit (PLU)

The parallel logic unit or PLU performs logical operations on data without affecting the contents of the accumulator. Bit manipulation operations such as setting, clearing and testing can all be executed in parallel with the main function of the code. Bit manipulation means that status registers can be altered immediately without degrading performance.

1.8.4 Internal Registers

Registers in the 'C5x family are organised into two main sections: internal registers and memory mapped registers. Functionally, there is no difference between the two types of register. The following table lists the internal registers.

Table 1: Internal Registers

Name	Symbol	Function
ACC umulator	ACC ACCH ACCL	A 32-bit accumulator accessible in two 16-bit halves. Accumulator is used to store the output of ALU
ACC umulator Buffer	ACCB	A register used to temporarily store the 32-bit contents of the accumulator
Pre scaler Count Register	COUNT	A 4-bit register that contains the value for the prescaling operation. BIT and BITT instructions operate on this register
Pro duct REG ister	PREG	A 32-bit register used to hold the multiplier's output.
Pro gram Counter	PC	A 16-bit program counter used to address program memory
Re Peat Coun Ter	RPTC	A 16-bit counter used to control the repeated execution of a single instruction
Stack	STACK	An 8x16-bit hardware stack used to store the PC during subroutine calls and interrupt service routines
ST atus Registers	ST0 ST1	Two internal status registers that hold status and control bits. There are two more status registers which are memory-mapped: PMCR and CBCR

1.8.5 Memory-Mapped Core Processor Registers

Twenty-eight core processor registers are mapped into the data memory space. They are listed in the table 2. Assembly language instructions refer to the register by its name. The second column shows the address of the register in data memory space. Since these registers are memory-mapped, it is the actual address where they reside. The third column is the description of the register. The name of the register is usually an acronym related to this description.

These registers play an important role in the behaviour of the processor. The most commonly used registers are IMR, IFR, PMST, TREG0, AR0-7 and INDX.

Table 2: Memory-Mapped Core Processor Registers

Name	Address		Description
	Dec	Hex	
	0-3	0-3	Reserved
IMR	4	4	Interrupt Mask Register
GREG	5	5	Global Memory Allocation REGISTER
IFR	6	6	Interrupt Flag Register
PMST	7	7	Processor Mode Status Register
RPTC	8	8	RePeaT Counter Register
BRCR	9	9	Block Repeat Counter Register
PASR	10	A	Block Repeat Program AccesS Register
PAER	11	B	Block Repeat Program Address End Register
TREG0	12	C	Temporary REGISTER Used for Multiplicand
TREG1	13	D	Temporary REGISTER Used for Dynamic Shift Count (5 bits only)
TREG2	14	E	Temporary REGISTER Used as Bit Pointer in Dynamic Bit Test (4 bits only)
DBMR	15	F	Dynamic Bit Manipulation Register
AR0	16	10	Auxiliary Register Zero
AR1	17	11	Auxiliary Register One
AR2	18	12	Auxiliary Register Two
AR3	19	13	Auxiliary Register Three
AR4	20	14	Auxiliary Register Four
AR5	21	15	Auxiliary Register Five
AR6	22	16	Auxiliary Register Six
AR7	23	17	Auxiliary Register Seven
INDX	24	18	INDeX Register
ARCR	25	19	Auxiliary Register Compare Register
CBSR1	26	1A	Circular Buffer 1 Start Register
CBER1	27	1B	Circular Buffer 1 End Register
CBSR2	28	1C	Circular Buffer 2 Start Register
CBER2	29	1D	Circular Buffer 2 End Register
CBCR	30	1E	Circular Buffer Control Register
BMAR	31	1F	Block Move Address Register

1.8.6 Interrupts


Interrupts are exceptions. The processor can be interrupted from its normal processing either by an external hardware signal or by a software instruction. The processor checks whether there has been an interrupt after every instruction. If there has been an interrupt, the program counter is loaded with the address of the appropriate Interrupt Service Routine (ISR) and the processor starts executing instructions in ISR until a RETE or RETI instruction is executed.

The following table shows 'C50 interrupts. The (!) symbol means that the signal is asserted when low.

Table 3: 'C50 interrupts

Label	Function
!RS	External reset interrupt
!NMI	Nonmaskable interrupt
!INT1	External user interrupt 1
!INT2	External user interrupt 2
!INT3	External user interrupt 3
TINT	Internal timer interrupt
RINT	Serial port receive interrupt
XINT	Serial port transmit interrupt
TRNT	TDM port receive interrupt
TXNT	TDM port transmit interrupt
!INT4	External user interrupt 4
TRAP	Trap instruction vector

Highest priority



Lowest priority

Increasing priority

When an interrupt is executed, the following registers are saved automatically. This is called a context save.

Table 4: 'C50 Interrupt context save

PC	Program Counter
ACC	Accumulator
ACCB	Accumulator buffer
PREG	Product register
ST0	Status register 0
ST1	Status Register 1
PMST	Processor Mode Status Register
TREG0	Temporary register for multiplier
TREG1	Temporary register for shift count
TREG2	Temporary register for bit test
INDX	Indirect addressing index register
ARCR	Auxiliary register compare register

Program counter is saved on **8-deep internal hardware stack**. **The same stack** is used for **subroutine calls**. Therefore care must be taken when calling subroutines from within interrupt service routines. **The rest of the registers** are pushed onto a **one-deep stack**. Because of this **one-deep stack**, the 'C50 does **not support nested interrupts**.

1.8.7 Peripherals

Peripherals interface the core of the processor with off-chip devices. The effectiveness of the peripherals are vital to the efficient operation of the processor. Slow peripherals may limit the performance of the DSP by creating a bottleneck. The TMS320C50 has a number of highly specified peripherals capable of operating in harmony with the core processor to give efficient communications.

Scan-Based Emulation

The JTAG scanning logic is used for testing and emulation purposes only. The JTAG scan logic provides the boundary scan to and from the interfacing devices. It can be used to test pin to pin connectivity as well as performing operational tests on peripheral devices surrounding the TMS320C50.

The dedicated emulation port is a subset of IEEE 1149.1 JTAG standard which can be accessed by using the XDS510 emulator from Texas Instruments.

The processor JTAG output pins are mirrored on the DSK board for easy access.

Serial Port

The serial input/output port connect the processor with off-chip serial devices such as AICs for high speed serial interfacing. This port is extremely important, and probably the most frequently used in all DSP applications.

The serial port is synchronous, full duplex and double buffered to aid efficiency.

Time Division Multiplexed (TDM) Serial Port

Time Division Multiplexed Serial Port is used in multi-processor systems for inter-chip communication. This port allows the device to communicate with up to seven other 'C5x devices processor.

Timer

The processor incorporates a 16 bit user programmable timing circuit which gives a divide down ratio of a $\frac{1}{2}$ to $\frac{1}{32}$. The timer can be stopped and started, set and reset all within the software. The internal timer can be used to supply a master clock signal to peripheral devices such as an AIC.

Software Wait States

The wait state generator enables the processor to interface slower off-chip memory without the need for external hardware. The processor can be made to wait 1, 2, 3 or 7 cycles, allowing external memory to keep pace with the DSP. Wait states are controlled by a set of registers.

Parallel I/O Ports

The TMS320C50 has 64K I/O ports (parallel input/output ports) which are addressed through the command set. The processor can easily interface with external devices requiring minimal off-chip decoding circuits.

Power Management

Power management circuitry puts the processor into power-down mode reducing the power consumption substantially when it is not actively used. This is very useful in portable equipment and helps the lifetime of the battery. Power-down mode can be initiated by external signals or by executing certain instructions.

1.8.8 Memory-Mapped Peripheral Registers

Peripheral circuits are operated and controlled through access of memory mapped control and data registers. The operation of serial ports, timer and wait state generators are all controlled through these memory-mapped peripheral registers. These are listed in the following table. The first column is the name of the register the second column is the address of the registers in the data memory space. We shall be using some of these registers in the programs featured within this guide.

Table 5: Memory-Mapped Peripheral Registers

Name	Address		Description
	Dec	Hex	
DRR	32	20	Data Receive Register
DXR	33	21	Data Transmit Register
SPC	34	22	Serial Port Control Register
	35	23	Reserved
TIM	36	24	Timer Register
PRD	37	25	Period Register
TCR	38	26	Timer Control Register
	39	27	Reserved
PDWSR	40	28	Program/Data S/W Wait-State Register
IOWSR	41	29	I/O S/W Wait-State Register
CWSR	42	2A	S/W Wait-State Control Register
	43-47	2B-2F	Reserved
TRCV	48	30	(Time Division Multiplex) TDM ReCeIve Data Register
TDXR	49	31	TDM Transmit Data Register
TSPC	50	32	TDM Serial Port Control Register
TCSR	51	33	TDM Channel Select Register
TRTA	52	34	TDM Receive/Transmit Address Register
TRAD	53	35	TDM Received ADdress Register

For more information on the architecture of the TMS320C50 processor please refer to the processor User's Guide that is supplied with the DSK.

1.9 Introduction to Assembly Language Instructions

1.9.1 Overview

The 'C5x instruction set is designed to support numerically intensive signal processing applications. The instruction set provides six basic memory addressing modes:

- Direct,
- Indirect,
- Immediate,
- Dedicated register,
- Memory-mapped register,
- Circular.

In the following sections, we shall consider some examples of each of these addressing modes.

1.9.2 Direct Addressing Mode

This mode uses the Data Memory Page Pointer (DP) register. This is an internal processor register which is not memory mapped. The DP is 9-bits long, providing MSBs of data memory address to a direct addressing mode instruction. The DP divides the data memory space into 512 pages of 128 words. Figure 12 shows the paging arrangement using the DP. The Data Page Pointer is not initialised to Zero on Reset, therefore the contents of the DP is not defined on Reset.

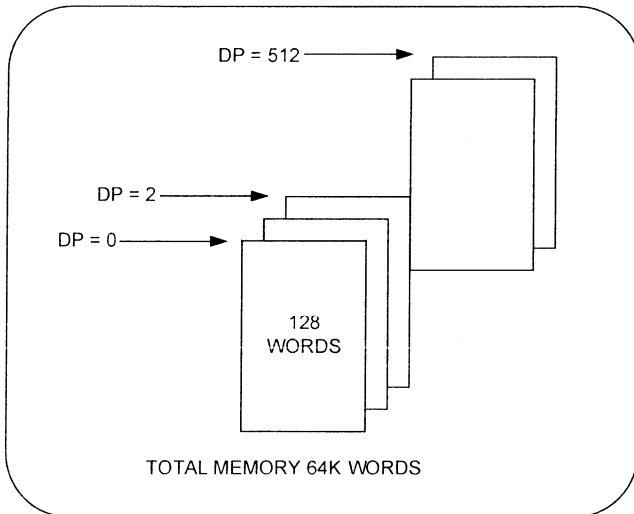


Figure 12: Data Memory Page Pointer (DP)

Let us consider an example.

AND DAT16 ; DP is currently pointing to page 4

This instruction will logically AND the contents of the memory location at 210h with the contents of the accumulator. The address calculation is as follows;

$$210h = 528d = (4 \text{ pages} \times 128 \text{ bytes each}) + 16 \text{ (DAT16)}$$

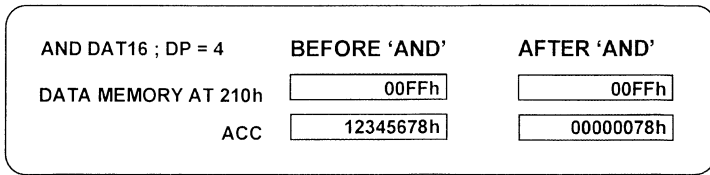


Figure 13: The operation of a Direct Memory Addressing Instruction

Other examples of Direct Memory Addressing Mode instructions include:

ADD adds contents of ACC to the contents of the pointed data memory location

OR Logically 'OR's the contents of ACC with the contents of the pointed data memory location

LST Load Status Register with the contents of the pointed memory location.

The above instructions can also be used with some other addressing modes. Chapter 4 of TMS320C5x User's Guide has detailed explanation of each instruction and a short summary of the whole instruction set. A brief study of this chapter will help you to familiarise yourself with the instruction set before we start programming in the next chapter.

1.9.3 Indirect Addressing Mode

This is the most complex addressing mode in the 'C5x. Eight auxiliary registers, AR0-AR7, provide the pointing mechanism for this mode. The following figure shows the arrangement of auxiliary registers.

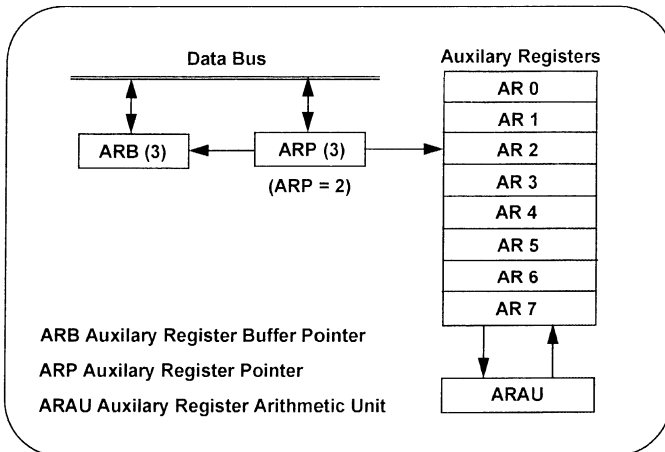


Figure 14: Auxiliary Registers

An auxiliary register is selected by loading Auxiliary Register Pointer (ARP) with a value from 0 to 7. The contents of auxiliary registers are operated on by the Auxiliary Register Arithmetic Unit (ARAU). ARAU implements increments and decrements, and performs other arithmetic operations on the auxiliary registers. The Load Auxiliary Register (LAR) instruction loads the address into the current auxiliary register. AR(ARP) denotes that the auxiliary register is to be selected by ARP.

The following instructions operate on the auxiliary registers:

ADRK; Add to auxiliary register short immediate

SBRK; Subtract from auxiliary register short immediate

MAR; Modify auxiliary register

Auxiliary registers may also be loaded by using memory-mapped writes to the auxiliary registers. However the use of this mode requires some care. The 'C5x User's Guide has more detailed information in chapter 4 about indirect addressing mode.

The indirect addressing mode uses a number of symbols. These are best demonstrated by examples.

Example 1:

ADD *, 8

* means that the contents of AR(ARP) are used as the data memory address. + autoincrements the current auxiliary register by 1 after the instruction is executed. 8 means that the data at the memory location pointed to by the contents of the current auxiliary register is left shifted by 8 bits before addition. The instruction adds to the accumulator the contents of the data memory location pointed to by the contents of the current auxiliary register after left shifting this data by 8 bits. The current auxiliary register is autoincremented after the instruction.

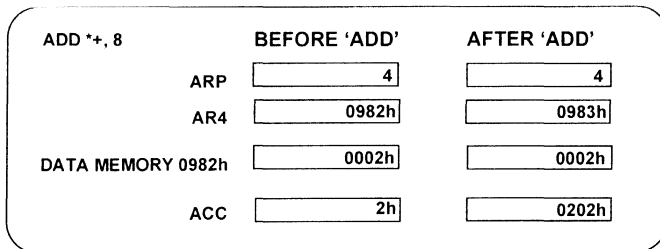


Figure 15: The operation of an indirect memory addressing instruction

Example 2:

ADD *, 8

Same as example 1 but no increment of the auxiliary register.

Example 3:

ADD *, 8

Same as example 1 but the auxiliary register is decremented after the instruction.

Example 4:

ADD *0+, 8

0 means that INDX register is used in the address calculation. In this case the contents of INDX is added to the current auxiliary register.

Example 5:

ADD *0-, 8

Same as example 4 but the contents of INDX is subtracted from the current auxiliary register.

Example 6:

ADD *+,8,AR3

Same as in example 1 but the ARP is loaded with the value 3 for subsequent instruction.

Example 7:

ADD *BR0-, 8

BR stands for Bit Reversed. Same as example 5 but the contents of INDX are subtracted from the current auxiliary register with reverse carry propagation. This instruction is very popular in performing FFTs.

Example 8:

ADD *BR0+, 8

Same as in example 7 except that the contents of INDX are added to the current auxiliary register with reverse carry propagation.

1.9.4 Immediate Addressing Mode

Immediate addressing is much simpler to understand. The instruction contains the value of the operand. In the following example one is added to the contents of the accumulator.

ADD #1h ; add 1 to the accumulator

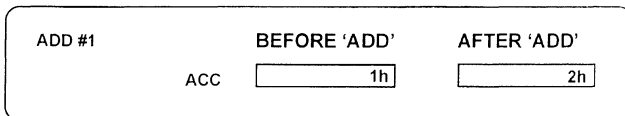


Figure 16: The operation of an immediate addressing mode instruction

The instruction below is another example of immediate addressing mode. The instruction loads the named auxiliary register with the immediate value. After the instruction is executed AR4 will contain 2FFFh.

LAR AR4, #2FFFh ;load AR4 with 2FFFh

1.9.5 Dedicated Register Addressing

There are nine instructions in the 'C5x instruction set that can use one of the two special-purpose memory-mapped core CPU registers. These two registers are:

BMAR ; Block Move Address Register

DBMR ; Dynamic Bit Manipulation Register

The following table summarizes the instructions that operate on these two registers. TMS320C5x User's Guide has detailed information on the operation of these instructions in Chapter 4.

Table 6: Dedicated Register Addressing Instructions

Register	Instruction	Description
DBMR	APL OPL CPL XPL	When an immediate value is not specified, these parallel unit instructions use the contents of DBMR as one of the operands
BMAR	BLDD BLDP BLPD	These instructions can use BMAR to point to the source or the destination space of a block move
BMAR	MADD MADS	These multiply and accumulate instructions can also use BMAR register to address an operand in data memory

The example below demonstrates the operation on DBMR;

OPL DAT10 ; Current data page is 6(DP = 6), OR the contents of pointed memory
; location with the contents of DBMR

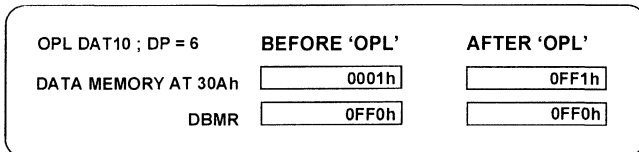


Figure 17: The operation of a PLU instruction on DBMR

The following example shows the use of BMAR in dedicated addressing mode.

BLDD DAT0, BMAR ; Current data page is 6 (DP = 6)

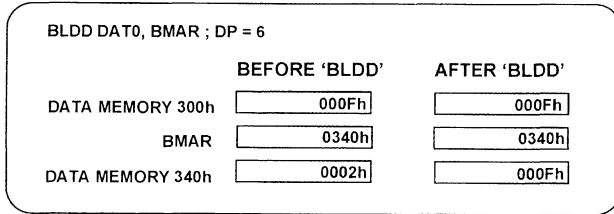


Figure 18: The use of BMAR in data memory access

1.9.6 Memory-Mapped Register Addressing

Memory-mapped registers reside in Data page zero and can be addressed by using data page pointer DP. However Memory-Mapped Register Addressing mode modifies these registers at any location in data page 0 without affecting the contents of data page pointer DP. The advantage of this addressing mode is that it removes the overhead associated with modification of DP when addressing memory-mapped registers.

- LAMM ; Load Accumulator with Memory-Mapped register
- SAMM ; Store Accumulator with Memory-Mapped register
- LMMR ; Load Memory-Mapped Register
- SMMR ; Store Memory-Mapped Register

The following example demonstrates the use of one of these instructions.

LMMR DBMR, #320h ; Load DMBR from data memory location 320h

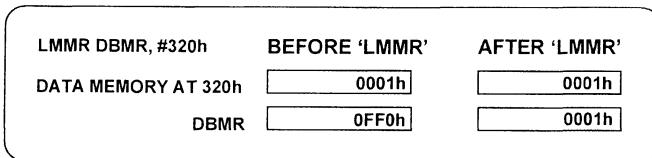


Figure 19: Loading of a memory-mapped register

1.9.7 Circular Addressing

Digital signal processing requires extensive use of circular buffers in waveform generation, convolution, correlation, filters and many more applications. The following figure shows the operation of a circular buffer. When the physical end of buffer is reached, the buffer pointer automatically wraps around and continues.

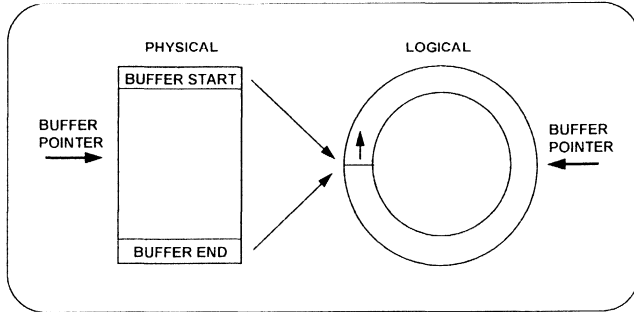


Figure 20: The operation of a circular buffer

The 'C5x supports two concurrent circular buffers. Auxiliary registers are used as buffer pointers. The following memory-mapped registers control the circular buffer operation:

- CBSR1 ; Circular Buffer One Start Register
- CBSR2 ; Circular Buffer Two Start Register
- CBER1 ; Circular Buffer One End Register
- CBER2 ; Circular Buffer Two End Register
- CBCR ; Circular Buffer Control Register

Start and End registers hold the appropriate addresses. CBCR enables and disables buffers and identifies which auxiliary register will be used as a pointer. The following table shows the definition of bits within CBCR.

Table 7: Bit functions of CBCR

Bit	Name	Description
0-2	CAR1	identifies the auxiliary register for circular buffer 1
3	CENB1	Circular buffer one enable = 1
4-6	CAR2	identifies the auxiliary register for circular buffer 2
7	CENB2	Circular buffer two enable = 1

The following example demonstrates the use of circular buffer addressing. Firstly the circular buffer one is set up. AR6 is assigned to be the buffer pointer. Data values can be read from the table with ease using a single load accumulate instruction.

LACL	#200h	load low accumulator with 200h and clear high accumulator
SAMM	CBSR1	store the contents of accumulator in circular buffer one start register
		this sets the start address for circular buffer one at 200h
SAMM	AR6	store the same address in AR6 so that it points to buffer start
ADD	#100h	add 100h to the accumulator, this will calculate the buffer end address
		accumulator contains 200h, add 100h, now it contains 300h
SAMM	CBER1	store this value in circular buffer one end register
		circular buffer is now set up
LACC	*+	load the accumulator from the table and increment AR7
		AR6 now points to the next sample
		further accesses in the same way will take more values from the table
		when AR6 comes to the end of buffer, it will simply wrap around
		and continue from the start of buffer

TMS320C5x User's Guide has further details of instructions of the processor. It also has a summary list of assembly instructions in chapter 4. A brief study of some of the instructions and a look at the summary instruction tables will make the reading of chapter 4 much easier.

1.10 More on DSK

1.10.1 DSK Memory Map

There is no external memory on the board except for 32 Kbyte ROM which is used only on power up. However the 'C5x DSK has 10K on-chip RAM. The DSK User's Guide has the memory map for the board but we shall repeat it here for ease of reference. The following figure shows the memory map of 'C5x DSK.

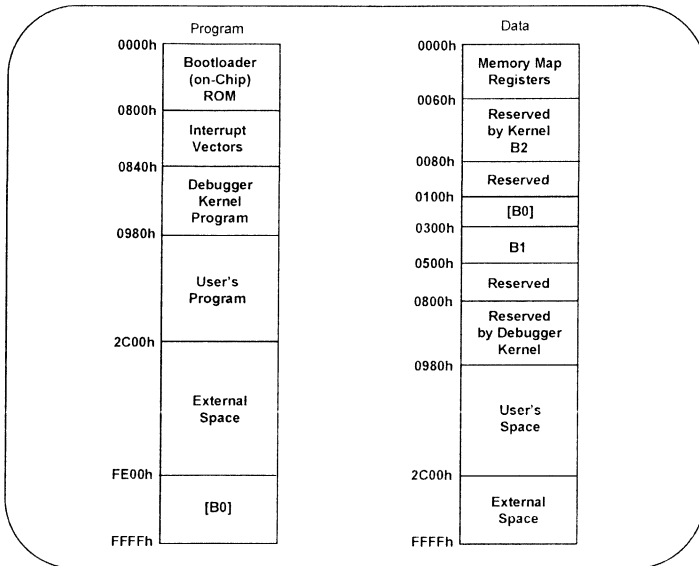


Figure 21: Memory Map of the C5x DSK

Memory map consists of two distinct separate areas: program and data. Program memory area starts with on-chip ROM which carries the bootloader. This area is masked with bootloader code during 'C50 chip manufacture. On DSK, 'C50's Single-Access RAM (SARAM) is mapped as program and data memory. RAM bit in PMST register should not be set to zero as this will map SARAM for data only. Interrupt vectors occupy up to 0840h. Debugger kernel program is loaded on start-up into the memory area from 0840h to 0980h. The user can use SARAM from 0980h to 2C00h.

Dual-Access RAM (DARAM) consists of three blocks. DARAM block B2 is reserved by Kernel and is used as buffer space for status registers. DARAM block B1 exists in data memory space at 300h-500h. DARAM block B0 exists either in local data memory space at 100h-300h or in program memory space at FE00h-FFFFh depending on the setting of CNF bit in ST1 register.

1.10.2 DSK Block Diagram

The following figure shows the block diagram of the DSK. The Analogue Interface Circuit (AIC) is connected to the serial port. Communications with PC is achieved through the use of three other signals.

Name	Function
BIO	Branch control input
XF	External Flag output
RS	Rest

PROM carries the start up program and the communications kernel. It is used only once during start-up and is not functional any other time.

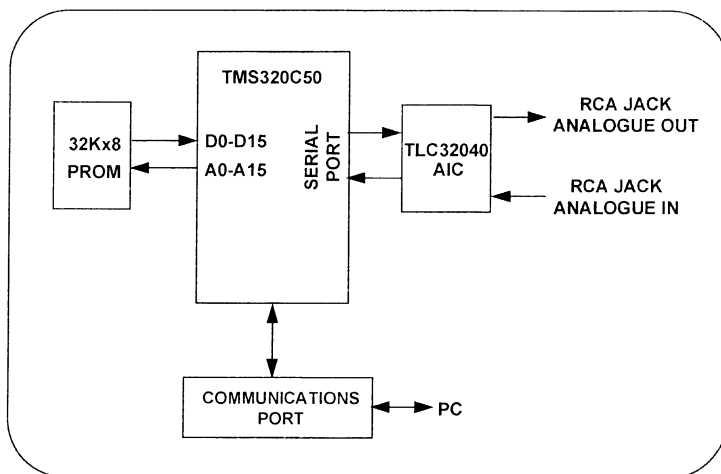


Figure 22: DSK Block diagram

1.10.3 The boot-up sequence of a DSK

The DSK boot-up sequence prepares both the 'C50 and the board for downloading information, monitoring and debugging program execution via the RS232 port. The kernel program, which performs these operations is stored in an external PROM and must be loaded into the internal RAM of the DSP.

A short program is held in the on-chip ROM (bootloader) of the 'C50 that downloads the contents of the external PROM into the 'C50's on-chip RAM. When the 'C50 is reset, the on-chip ROM drives the BR pin low and starts to load the kernel program from the external PROM on the DSK.

The DSK uses BIO and XF pins to communicate via an RS232 interface with the outside world, in this case a PC. When the BIO line goes low, this indicates that an RS232 transmission start bit has been sent by the host. Since the baud rate is unknown, the PC sends 0x80 data through RS232 and the 'C50 begins counting the number of elapsed CPU cycles until the BIO line goes high again. The baud rate is set by the width of the start bit plus 7 data bit, then divided by 8.

The kernel program consists of a communications kernel and a simple debugger. The communications kernel enables bi-directional RS232 communications using XF and BIO. The simple debugger uses the TRAP instruction and INT2. TRAP instruction is used for many features of the debugger - for example, single stepping. To single step, the debugger replaces the next instruction with a TRAP and saves the replaced instruction in a safe memory location. After the execution of the current instruction, the debugger restores the saved instruction. All this happens in the background and is transparent to the user.

The DSK module uses the INT2 line tied to the BIO input to sense the activity on the RS232 receive line. If you turn off the global interrupt enable bit or unmask INT2 the runtime halt feature will not work. Likewise, if you overwrite the vector table for either TRAP or INT2 the kernel will fail to operate correctly.

Summary of reserved memory areas, opcodes and register settings used by the debugger:

Memory areas

All of RAM block B2 (60h-80h)	used for CPU register storage
SARAM (800h-980h)	used by the kernel

Bit settings in registers

PMST OVLV & RAM bits	configures SARAM as program & data RAM
IMR=4 bit in IMR	sets interrupt 2 INT2

Opcodes

TRAP opcode is reserved for the debugger for the implementation of breakpoints and single step.

2. The Programming Process

2.1 The DSK Package

The DSK package* includes an assembler, a debugger and a comprehensive application guide. The pack contains everything necessary to develop, run and debug 'C5x software.

3½" Disk 1: DSK assembler, loader, debugger and a number of application programs

3½" Disk 2: A number of application programs including Sine Wave generator, Musical Note Generator, Serial Communications, FIR filter, Amplitude modulator, Echo generator, Minuet player, Sound recorder, Function generator, DTMF generator, Spectrum analyser and Oscilloscope

Hardware: TMS320C50 based DSK board

Literature: TMS320C5x DSK User's Guide Introducing the DSK
DSK Applications Guide This guide
TMS320C5x User's Guide Comprehensive guide to the processor

2.2 Getting the system ready

The DSK must be prepared to run programs before it is useful as a development tool. We need to complete a series of steps to get the DSK up-and-running. These are documented in the Starter Kit's User's Guide (Chapter 2, steps 2.2 to 2.6).

2.3 Trouble-Shooting

If the DSK is not running, please check all the suggested steps in **Installation errors** section of Chapter 2 in DSK User's Guide.

If your DSK is still not functioning, the following are the most common faults:

- Your Power Supply Unit (PSU) may be regulated. It may not be supplying the right voltages to the DSK board. For DSK to function properly, you need a **9V AC** supply.
- RS232 cable may have the wrong connections. Check with a continuity tester that you have the connections shown in the table below.

Signal name	Pin Numbers on DB connectors	
	Computer	DSK
Transmit data	2	3
Receive data	3	2
Signal ground	7	5
Data terminal ready	20	4

If not, you will need to get a cable with the correct connections.

* If you purchased the DSK Applications Guide separately, you will only get the guide and Disk 2.

- Try changing the baud rate, starting with a lower one.

```
DSK5D b1200 ↵
```

Try also 2400 and 4800.

- Another common fault is the alteration of PATH statement. Check that your AUTOEXEC.BAT file has DSK directory listed as shown below.

```
PATH=c:\dos;c:\dsk
```

If you have installed DSK software on a drive different than that of the PATH statement, then that drive must be specified. For example if DSK directory is on your E drive, then the path statement should be:

```
PATH=c:\dos;e:\dsk
```

You will need to restart the computer to make these changes take effect.

- CONFIG.SYS file must have a line:

```
FILES=20
```

If the FILES line has a number specified which is higher than 20, do not reduce it, leave it as is.

- If your DSK is still not functional, reinstall the software supplied on Disk 1 onto your hard disk.
- It is also possible that your computer may have a fault. Try getting DSK going on a different computer.

Assuming that your DSK is now functional under DOS, you can try to use it under Windows 3.1™, Windows 95™ or Windows NT™. Using your DSK in a Windows™ environment has some advantages particularly if you are using a word processor such as Word 6™ to edit your programs. You can switch between a DOS window and word processor window using ALT-TAB key combination.

To have your DSK working under Windows, open a DOS window and enter

```
DSK5D c2 ↵
```

It is common that Windows™ uses serial port 1 for mouse communications. The command line above assumes serial port 2 is free for DSK. Use c3 or c4 if serial port 2 is used by some other device.

2.4 Using the Debugger

The DSK user's guide has detailed information on the use of debugger in Chapter 3. The interface is very intuitive. We shall use a few example commands here to get you started quickly. For a more comprehensive study, follow chapter 3 of DSK User's Guide.

DSK's debugger has a window oriented interface as shown in Figure 23. The top menu bar shows the commands. Each command also has a sub menu. Most commands consist of two letters. Those are the capital letters in menus. Commands are issued at the bottom of the window.

For example, to display an area of memory, Display menu is activated by 'D' and then a sub menu is displayed. Choosing Data 'D' in this sub menu displays contents of data memory in the lower portion of the screen.

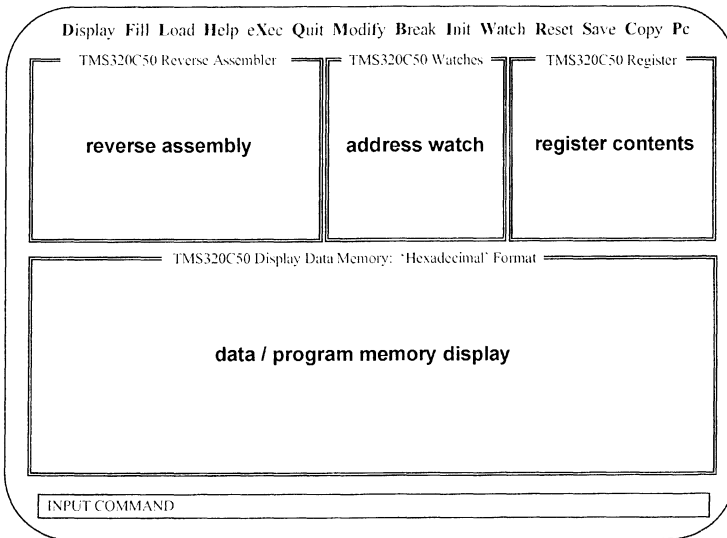


Figure 23: DSK debugger main screen

Let us now use a few commands.

Example 1: Filling data memory with a hex pattern

Enter FD 'Fill Data' followed by a carriage return in the INPUT COMMAND line. A dialogue box appears. Reply to prompts as shown in Figure 24. Each entry must be followed by a carriage return. Once the 'Fill Data Memory finished' message appears, press 'ESC' to remove the dialogue box. The lower window now displays the filled memory section. 127 (7f) memory locations of a word length (16 bits) have been filled with the hex pattern 'abcd'.

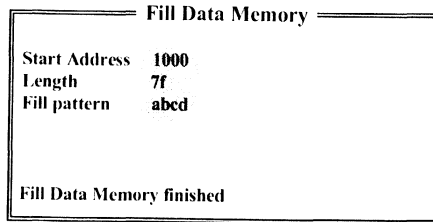


Figure 24: Filling data memory in DSK debugger

Example 2 : Modifying the contents of a register

Enter MR **Modify Register** followed by a carriage return in the INPUT COMMAND line. A dialogue box appears as shown in Figure 25. On the 'Input' line in the box, enter the following

```
accu = 1f1f ↵
```

This will change the contents of the accumulator. Accumulator is 32 bits but we have only put in a 16-bit value.

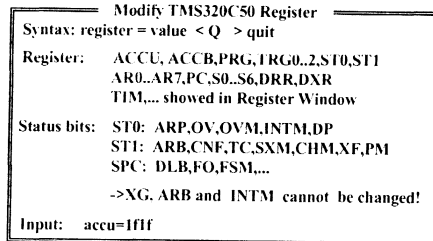


Figure 25: Modifying contents of a register

Observe the rightmost window where registers are displayed. The contents of 'ACC' has changed as shown in Figure 26.

```
TMS320C50 Register
ACC :00001f1f      C:0
ACCB :00000000     OV:1
PRG  :00000000     PM:0
TRG0 :0000      TRG1:0000
TRG2 :0000      DF: 0000
ST0  : 0600      ST1  :11fc
PC   : 0000      AR0: 0000
St0  : 0000      AR1: 8404
St1  : 0000      AR2: 0a00
St2  : 0000      AR3: 0000
St3  : 0000      AR4: 0000
St4  : 0000      AR5: 0000
St5  : 0000      AR6: 0000
St6  : 0000      AR7: 0000
DRR  :6c00      DXR  :0000
TIM  :0000      PRD  :0000
TMR  :0002      IFR  :001a
PMST :0834      INDX :0000
DBMR :0000      BMAR :0000
CWSR :0000      GRG  :ff00
SPCR :3CC8      TCR  :0400
```

Figure 26: Debugger register display

All registers in the register window have their standard names. St0 to St6 denote the hardware stack. You can observe the operation of the stack when a 'CALL' instruction is executed.

It is a good idea to try a number of menu commands before proceeding to the other sections. For example, use 'MD' to modify the contents of memory locations. You can see the changed contents of the memory locations in the data display window. You can also add watches 'WA' to these memory locations and observe the changes in the 'Watches' window.

2.5 Suggested Hardware Configuration

A suggested system configuration is shown in Figure 27. We have already covered power input and connection to the PC.

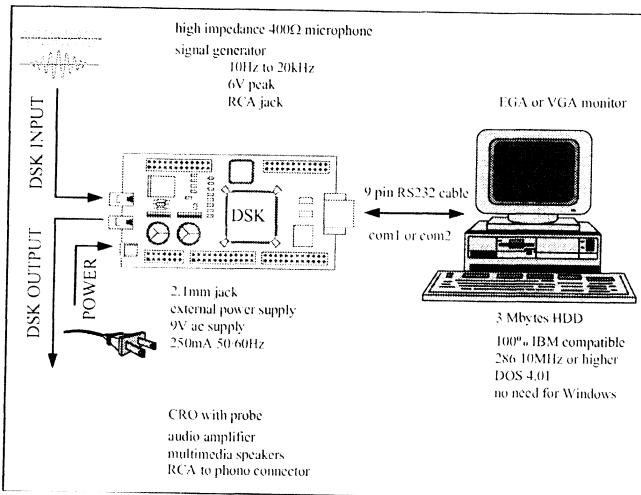


Figure 27: Suggested hardware configuration

Serial Input:

Ideally, a signal generator provides the best input to the DSK as the user has total control over input frequency and power. Alternatively, voice or sound can be used by connecting a microphone to the RCA jack or a sound source such as a cassette player.

Serial Output:

Monitoring the output can be achieved by connecting the RCA port to an amplified speaker. The user can then hear how the program manipulates data using a cathode ray oscilloscope (CRO) enables the designer to see the signal. Also the signals on DSK board's header pins can be tested using a CRO.

2.6 Installing Demonstration Software

Several software examples are contained on a 3½" floppy disk, which should be loaded onto the hard disk drive.

- insert the application disk into your floppy disk drive
- from the relevant hard disk drive move to the floppy :`\ a :`
- type `a:\install`
- follow the steps in the installation program

- remove the disk from the drive
- application software should be in DSK's home directory :**dsk**

The installation program creates sub directories under the DSK's home directory '**dsk**', then copies the files from the applications disk into appropriate directories. The DSK User's Guide calls this directory '**dsktools**'. However DSK installation program creates it as '**dsk**'. Do not be confused by this; just name the home directory '**dsk**'; it is simpler.

2.7 Assembling and running a program

The DSK is a programmable tool. On a cycle by cycle basis it must be told what to do and in what order. The set of rules by which the DSK processor executes instructions is known as the program. The program is totally defined by "programmer".

A program is written using a set of pre-defined instructions. We studied some assembly language instructions for 'C5x in the previous chapter. DSK User's Guide, Chapter 4 has comprehensive information on 'Using the DSK Assembler'. It is important that the conventions and defined protocols explained in DSK documentation are adhered to. This will make your programs understandable and easier to debug.

The TMS320C5x instruction set is very comprehensive, allowing the programmer great flexibility when writing the code. Once the program has been written, it needs to be assembled. The assembler is a software program that converts the designer's source code in assembly language into a format (machine code) that can be downloaded to the DSK.

To check whether your assembler is working correctly, type

```
C:/DSKTOOLS>dsk5a try1 ↵
```

File 'try1.asm' is supplied with DSK. You can use a different file if you wish.

This will assemble the file and create an 'object' file with 'dsk' extension which is down-loadable to DSK for execution. The assembled file is not Common Object File Format (COFF). However DSK debugger supports COFF files. You may need this feature if you use other 'C5x development tools.

Make sure that you are in DSKTOOLS directory. To download the program, invoke the debugger by typing:

```
C:/DSKTOOLS>dsk5d ↵
```

Now the assembled program can be downloaded from within the debugger by issuing Load Dsk 'LD' command.

INPUT COMMAND: LD

A dialogue box will appear as shown in Figure 28. Press carriage return and the contents of the dialogue box will change as shown in Figure 29. Reply yes 'y' to this dialogue box. Wait till download terminates and then press 'ESC'. This completes the download process and the top leftmost window of the debugger will now display the 'try1' program.

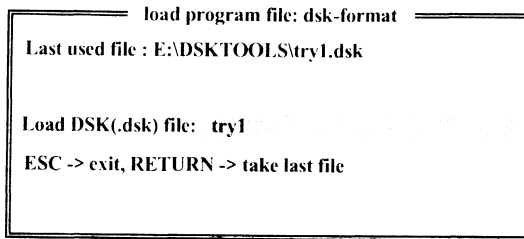


Figure 28: Loading a DSK file

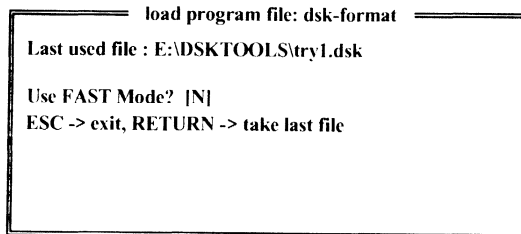


Figure 29: Choosing a download type

You can now run this program by issuing eXecute Go 'XG' command from the INPUT COMMAND line of the debugger. 'ESC' stops execution.

In DSKTOOLS directory, you will notice two files which are never mentioned anywhere; 'DSK5D.CFG' and 'DSK5D.PRM'. These two files are created by the debugger to keep its configuration parameters. You need never touch them or modify them.

Texas Instruments has provided a series of sample programs with the DSK. We shall start the programming process with one of these sample programs; PASS.ASM.

2.8 PASS.ASM

"PASS.ASM" is a simple program and it only **passes** information from the serial input port to the serial output port. On every interrupt, the input port of the DSK is read into the DSP and is written unmodified to the serial output port. Figure 30 demonstrates this functionality. In addition, the program also:

- Initialises the DSP processor
- Initialises the on-board AIC, setting a sampling rate of 8KHz
- Reads every sample from the serial port to the DSP's accumulator
- Writes the accumulator to the output serial port before the next sample arrives.

The program serves to illustrate the quality of a signal passed through a 16 bit processor and also shows how the board can be initialised. In addition, the software can be used as a template for expanding and developing different applications.

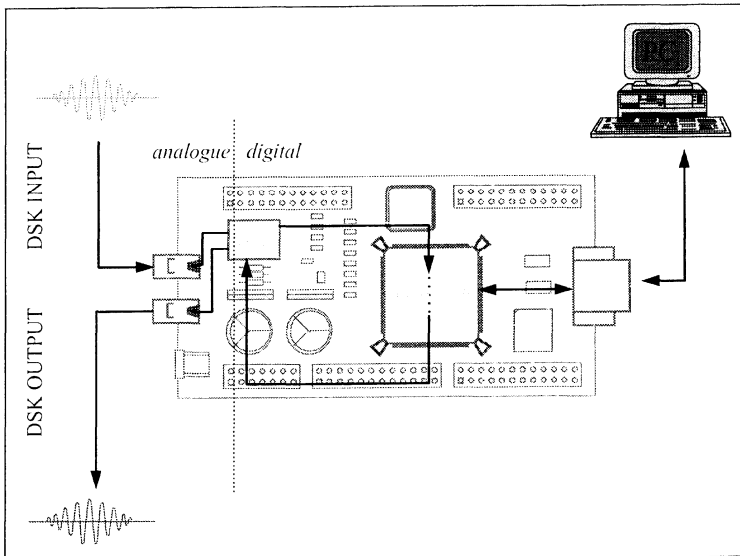


Figure 30: Functionality of PASS.ASM

Before starting the assembly process, connect a sound source to the input RCA jack of DSK. This could be a dynamic microphone, a cassette player or just a signal generator. Also connect an amplified speaker to the output RCA jack. This configures the hardware.

- 1) To assemble and run the software from the source code file PASS.ASM type:

```
:\dsktools\pass\DSK5A PASS.ASM -I ↵
```

(Chapter 4 of DSK User's Guide)

The assembler transforms the source code *.asm file into a *.dsk program that can be downloaded to the DSK. A list file is also produced because of '-I' option. The list file is helpful when debugging. The files produced are as follows:

PASS.LST	a list file of the assembled program
PASS.DSK	a DSK executable file

- 2) To download the software directly to the DSK type:

```
:\dsktools\pass\DSK5L PASS.DSK ↵
```

(Chapter 4 of DSK User's Guide)

The down-loader program takes the PASS.DSK program and loads it to the DSK via the RS232 cable. You should now hear the input sound source being reproduced on the output speaker.

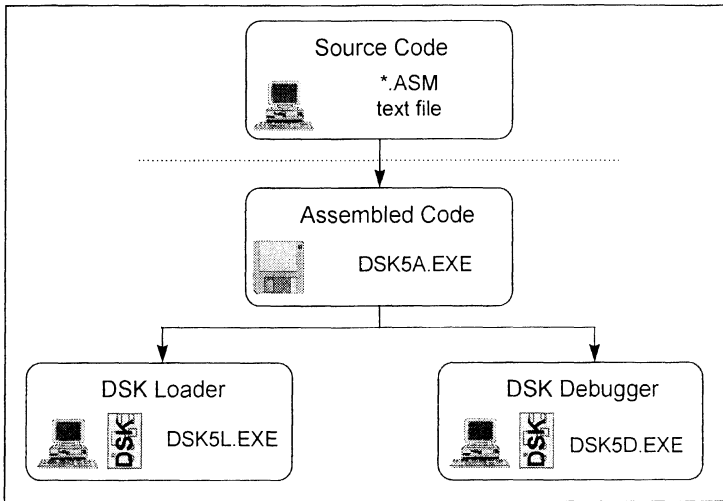
- 3) To download the software to the DSK using the debugger, type:

```
:\dsktools\pass\DSK5D ↵
```

(Chapter 3 & 5 of DSK User's Guide)

This initiates the debugger. Now from within the debugger, download the program:

LD	to load a DSK file type
PASS.DSK	file name
Y	fast load
↵	back to the debugger
XG	run the program



You should hear the input sound source being reproduced on the output speaker. Now let us examine the program.

2.9 The fundamentals of a DSK Program

2.9.1 Overview

Once the PASS.ASM program has been assembled, we can observe how it works. This practical approach illustrates the starting point to any DSK assembly program.

PASS program represents a very simple DSK program. It also illustrates the global concepts that are behind a DSK program and the software components that are necessary to ensure functionality. Any DSK program is split up into two primary parts; the instructions that are required to set up the board to run correctly and the instructions that executes the user's algorithm.

The first section is a necessary part of any DSK program which can be simple like PASS.ASM or complicated such as a digital mobile telephone application. In PASS.ASM this part consist of Settings, Interrupt Settings, Processor Initialisation and Board Initialisation. The second section simply depends on the application. In PASS.ASM second section consist of a Main Program and two Interrupt Service Routines (ISRs) - very simple code. Figure 31 demonstrates the code sections for a typical DSK application and PASS program.

Some DSK code only needs to be executed once, while others must be run continually on every interrupt. The initialisation and settings blocks in our example are run once while Main program and ISRs run continuously.

Let us briefly consider the function of each module;

1 - Settings

A section of code that assigns, allocates and reserves areas of data memory to hold constant values such as filter coefficients and also temporary results from algorithm execution. In PASS, this section stores the constants that determine the AIC's characteristics.

2 - Interrupt Settings

A section of code that tells the processor the type of interrupts to expect and what to do on receiving an interrupt. In PASS, this section holds the addresses of ISRs.

3 - Processor initialisation

Code which initialises the processor, transferring the known RESET state to the required set-up. This includes masking registers, enabling interrupts and setting operation modes. This is one of the very important sections of the code. If not executed, the other code sections may not execute correctly.

In PASS, this section sets PMST register, data page pointer (DP), processor wait states and appropriate interrupt masks.

4 - Board initialisation

Code that initialises the DSK board, particularly the AIC's sampling rates and anti-aliasing filters.

5 - Main program

The main directives to the processor conveying the user's intention. In PASS, this section is very simple and consist of a few instructions that wait for interrupts.

6 - The Interrupt Service Routines (ISRs)

The ISRs contain instructions that the processor executes on receiving an interrupt. In PASS, there are two ISRs allowing reception and transmission of data.

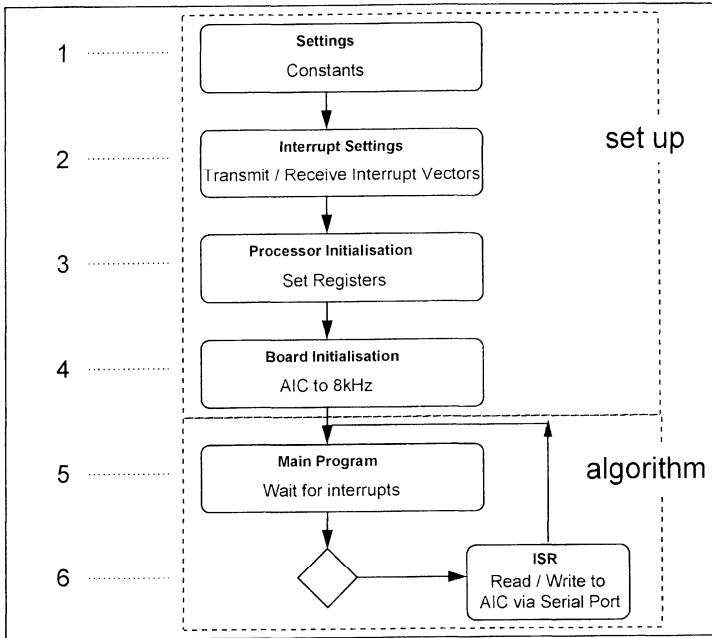


Figure 31: PASS.ASM Flow Chart

The DSK assembler is simple and requires no linker stage. With the DSK we declare the memory sections within the source code itself.

The recommended memory segmentation is shown Figure 32. The memory segmentation in the program code is done with assembler directives such as 'mmregs', 'ds' and 'ps'. DSK User's Guide has more information on assembler directives in Chapter 5. The memory allocation issued with these directives must agree with the processor and the system memory map. For example in PASS, data values are stored starting from 0F00 hex. Checking this with the memory map of DSK explained in Chapter 1, it falls into 'User's Space'.

.mmregs		Defines memory mapped registers such as ARx's
.ds	(address)	Assemble to data memory
	<i>data values</i>	0F00h within PASS.DSK
.ps	(address)	Assemble to program memory
	<i>interrupt vectors</i>	080Ah within PASS.DSK
.ps	(address)	Assemble to program memory
.entry		Marks execution start point of program
	<i>program code</i>	0A00h with PASS.DSK
.end		Marks end of program

Figure 32: Memory Addresses

Now let us examine each of these fundamental program blocks for PASS. While you are reading this text you may use a word processor to view the PASS program.

2.9.2 Settings

The first assembler directive '**.mmregs**' defines all the memory mapped registers with their standard names that were listed in Chapter 1.

The AIC initialisation routine uses pre-set values to determine the sampling rates and filter cut-off frequencies of the transmit and receive channels of the AIC. These constants are stored at the top of the code. The assembler directive '**.ds**' causes these constants to be listed in data memory from address 0F00h. The data values listed here will be used to set the AIC sampling frequency and gain. Constants are given the names of registers in the AIC to make them more understandable. For example the constant to be written into Transmit Counter Register A is named TA, Receive Counter Register A is named RA and so on.

The following table lists these names

Transmit Counter A	TA
Receive Counter A	RA
Transmit Counter B	TB
Receive Counter B	RB
Control Register	AIC_CTR.

The meaning of comment lines will become clearer in the explanation of AIC and AIC2ND subroutines later on.

```

;=====;
; .mmregs
;=====;
;=SETTINGS
;=====;
        .ds      0f00h      ; Assemble to data memory
TA      .word   17         ; TA AIC value - Fcut = 8 kHz
RA      .word   17         ; RA AIC value - Fcut = 8 kHz
TB      .word   37         ; TB AIC value - Fs = 2*Fcut
RB      .word   37         ; RB AIC value - Fs = 2*Fcut
AIC_CTR .word   28h        ; |LP xx G1 G0 | SY AX LB BP|
;-----+-----+
; |      GAIN      | | | +.. BP Filter
; |      Synch    --+ | +----- Loopback
; |      Auxin    -+---+
; + (sinx)/x filter
;=====;

```

Figure 33: PASS.ASM Settings

Note that the values are stored in decimal format except the AIC_CTR which is hexadecimal format (28h).

```

name .word value ; description, is a 16 bit data integer
name .set  value ; description, assigns a name to a value
; which is substituted during the assembly
; process
        .space bits ; description, reserves areas of memory for
; data of a certain number of bits.

```

DSK User's Guide has more information on additional assembly directives in Chapter 5.

2.9.3 Interrupt Settings

The Interrupt Settings section informs the processor the type of interrupts to expect and on receiving an interrupt, where to find the ISR. These addresses are called Interrupt Vectors. On recognising the interrupt, the processor goes to the interrupt vector location. Interrupt vector location contains a further branch to a the ISR. In PASS, location 080a hex contains a branch instruction and the next location contains the actual address of receive ISR. The next two locations contain the branch instruction and the address of the transmit ISR.

```

;=====;
; INTERRUPT SETTINGS
;=====;
        .ps      080Ah      ; Assemble to program memory
RINT    b       RX         ; Branch to RX on receive interrupt
XINT    b       TX         ; Branch to TX on transmit interrupt
;=====;

```

Figure 34: PASS.ASM Interrupts

You can verify this from within debugger;
 Load PASS in the debugger and then
 INPUT COMMAND: DPA ↵

New Address: 080a .J

This command will display the contents of all program memory locations starting from 080a. The contents of 080a and 080b should be '7980' and '0a16' which means branch to address '0a16'. Now return to main disassembly window and check the actual location of the receive ISR.

2.9.4 Processor Initialisation

Before any processing starts, the processor must be set to a known state. The TMS320C50 has a number of configuration registers. This program section sets these registers.

The initialisation instructions may need modifying for more complicated programs.

```

;=====;
;-PROCESSOR INITIALISATION                               =;
;=====;
        .ps 0A00h           ; Assemble to program memory
        .entry              ; Mark program entry point
        .text               ; Text
START setc    INTM          ; Disable interrupts
        ldp    #0           ; Set data page pointer to page zero
        splk  #0834h,PMST   ; Write 16 bit pattern to PMST register
        lacc  #0           ; Load accumulator with number zero
;-----;
        samm  CWSR         ; Set software wait state to zero
        samm  PDWSR        ; Set software wait state to zero
        splk  #022h,IMR    ; Using XINT syn TX & RX
        call  AIC          ; initialise AIC and enable interrupts
;-----;
        clrc  OVM          ; Overflow mode is set to zero
        spm   0            ; Product shift mode is set to zero
        splk  #012h,IMR    ; Mask interrupts
        clrc  INTM        ; Enable interrupts
;=====;

```

Figure 35: PASS.ASM Initialisation

- Before commencing the processor initialisation, the interrupts which could interfere with the program flow are temporarily disabled. This is performed by writing directly to the status register (ST0) which contains the interrupt "on/off switch". The SETC command sets the location within ST0 to 1. This location is represented by a defined label 'INTM'. This instruction disables all interrupts. This setting does not effect the operation of the RESET interrupt.
- The TMS320C5x processor splits memory up into a series of manageable regions which are termed as "pages". We discussed this in Chapter 1, Direct Addressing Mode' section. When addressing memory, the program instruction contains the lower 7 bits of the 16 bit memory address. The higher nine bits are referred to by the data page, thereby making up the whole address. Thus the data page points to one of the 512 pages while the instruction part of the address refers to an offset on each page. Therefore, it is possible to address memory using only one 16 bit program instruction. 'LDP #0' instruction sets data page to zero.

- SPLK instruction writes a 16 bit word directly to a given register, in this case the Processor Mode Status Register 'PMST'. Each bit in this register affects a configuration in 'C50. Let us study the function of each bit:

Table 8: PMST Register

Bit	Symbol	Setting	
0	BRAF	0	is a flag that is automatically set when block repeat is active. In the case of PASS, it is not the case as these instruction in the processor initialisation need only be run once.
1	TRM	0	disables the use of multiple TREGs which sets the device in a TMS320C2x compatible mode.
2	NDX	1	enables extra index register and configures the device to run in an enhanced TMS320C5x mode. INDX and ARCR are not affected by any C2xx compatible instruction
3	MP/IMC	0	disables / enables on-chip ROM. The corresponding external pin is sampled at RESET and is reflected in this bit. In this case, on chip ROM is enabled.
4	RAM	1	RAM =1 sets the entire on-chip RAM into program space.
5	OVLY	1	RAM overlay bit. This bit works together with RAM bit to map on-chip RAM. Table 9 shows the affect of possible bit combinations.
6	zero	0	
7	AVIS	0	is address visibility. When this bit is set, internal program address is mirrored on external pins for address tracing and debugging.
8	zero	0	
9	zero	0	
10	zero	0	
11	IPTR	1	these five bits point to the 2K memory block where the interrupt vectors reside, which allows re-mapping. At reset these bits reside at 0000h in memory. The interrupt vector address is generated by concatenating these bits with the interrupt vector number. Figure 36 demonstrates the address generation of the receive interrupt in PASS.
12	IPTR	0	
13	IPTR	0	
14	IPTR	0	
15	IPTR	0	

PMST	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0
Hex	0				8				3				4			

Table 9: On-chip Single Access RAM configuration control

OVLY	RAM	On-chip SARAM configuration
0	0	Disabled
0	1	Mapped into program space
1	0	Mapped into data space
1	1	Mapped into both; program and data space

In summary, the settings of PMST register:

- extra index register is enabled and the processor is in enhanced 'C5x mode
- on-chip SARAM is mapped into both; program and data space
- interrupt vector page is set to 8.

Actual interrupt addresses are calculated by concatenating the IPTR defined page number with the interrupt vector number. Figure 36 shows the calculation of the receive interrupt vector when IPTR is set to 8.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vector	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0
Hex	0				8				0				a			
IPTR	0	0	0	0	1											
Interrupt Number									0	0	0	0	1	0	1	0

Figure 36: Interrupt vector address calculation

The instructions until now have completed the **processor** initialisation. The following instructions prepare the processor for **board** initialisation.

- 'LACC' instruction loads the number zero into the accumulator, overwriting all other values. This is in preparation for the subsequent instructions.
- 'SAMM' stores the value contained within the accumulator (in this case zero) to a memory mapped register. Both registers CWSR and PDWSR control software wait states that hold up the CPU while accessing slower off-chip memory. As the DSK has no off chip memory contained on the board, this function is not required and therefore is zeroed. Chapter 5 & 6 of TMS320C5x User's Guide has more information on the use of these registers.
- The Interrupt Mask Register (IMR) selects the interrupts to mask. The IMR is written to with the 'SPLK' instruction. The instruction works in conjunction with the 'SETC INTM' which is a global switch for all interrupts. The IMR register selects the interrupts to operate when interrupts are enabled. 022h turns off the receive interrupts and allows the transmit interrupts which is used within the AIC routine.
- 'CALL AIC' instruction starts executing code from the AIC label within the code. This is a 'CALL' to subroutine 'AIC'. After execution, the program returns to the next line after the subroutine call.

These instructions have now completed the board initialisation. We do not yet know what AIC subroutine has accomplished. We shall examine that in the next section. The following set of instructions prepare the processor to run the main program.

- 'CLR C OVM' clears the overflow mode bit. OVM bit resides in ST0. If overflow occurs during execution, the most significant bits are allowed to overflow normally.
- The PM status bits are two least significant (LSB) bits contained within the Status Register 1 (ST1). They control a predefined shift within some instruction set

commands such as a multiply and accumulate. As the program does not require such a shift, these bits are set to zero.

- Finally, the INTM bit of the status registers is cleared, allowing all masked interrupts to occur.

2.9.5 AIC Initialisation

In the previous section, reference was made to the initialisation of the AIC. The initialisation is completed by a subroutine. Since subroutines do not represent the main code and functionality, they are usually kept towards the end of the software. The main or initialisation code can then 'CALL' the subroutines.

The TLC320C40 is a general purpose codec (coder-decoder) which is incorporated on the DSK. It is connected to the serial port of the 'C50. The device and the serial port of 'C50 must be configured to operate at the correct sampling rates and filter cut-offs. For a detailed description of how to set sampling rates and other features of the AIC please refer to the appendix x. The operation of the serial port of 'C50 is explained in detail in Chapter 5 of TMS320C5x User's Guide.

The functionality of the AIC initialisation routine is relatively complicated. For the inexperienced user an overview of the functionality of the routine will be sufficient at first. It is possible to use the routine without fully appreciating how it works. In general:

- The processor timer is used to drive an external pin which clocks the AIC unit
- The Serial Port Control (SPC) register is set
- The AIC is reset
- Various registers are loaded
 - TA and RA
 - TB and RB
 - AIC_CTR
- Return to main program.

```

;=====;
;=BOARD INITIALISATION                               =;
;=====;
AIC splk  #20h,TCR          ; To generate 10 MHz from Tout
      splk  #01h,PRD        ; Load period counter with 1
      mar   *,AR0          ; Modify AR pointer (used with GREG)
      lacc  #0008h         ; Load acc with 08h
      sacl  SPC            ; Store acc in SPC
      lacc  #00C8h         ; Load acc with 0C8h
      sacl  SPC            ; Set and reset SPC register
;-----;

```

Figure 37: PASS.ASM Board Initialisation

Note the following:

TA	driven by master clock and determines D/A conversion timing
TB	
RA	driven by master clock and determines A/D conversion timing
RB	

Setting the timer and initialising the serial port of 'C50

The Timer on 'C50 is used to generate a clock pulse for AIC. The timer is a counter that is decremented by one at every CLKOUT1 cycle. CLKOUT1 is an internally generated clock and is half of the master clock (20 MHz). A timer interrupt is generated each time the counter decrements to zero. This is also mirrored on TOUT pin. This waveform can be used as clock for external devices such as AIC. The timer block of the processor consist of three registers; Timer Control Register (TCR), Period Register (PRD) and Timer Register (TIM). The frequency on TOUT pin is given by;

$$TOUT = \frac{CLKOUT1}{(TDDR + 1) \times (PRD + 1)}$$

TDDR is Timer Divide Down Ratio which is represented by 4 bits in *TCR*. *PRD* is the value in the Period register. Let us now set the *TOUT* to 10MHz.

- SPLK writes 020h to the or Timer Control Register (TCR).

Table 10: TCR Register

0	Timer divide down ratio	0
1	Timer divide down ratio	0
2	Timer divide down ratio	0
3	Timer divide down ratio	0
4	Stop/ Start timer (stop = 1)	0
5	Reload Timer with period 1	1
6	Prescaler	0
7	Prescaler	0
8	Prescaler	0
9	Prescaler	0

Setting bit 5 high resets the timer. Timer Divide Down Ratio (TDDR) bits are reloaded from the prescaler on every countdown. These are set to zero to select the smallest period.

- The value '1' is written to the PRD register. The value of the PRD register is loaded into the timer register, TIM, which is decremented on every clock cycle by one. When TIM reaches zero, the timer is reset by automatically writing bit 5 high once again.

- Now we can calculate the output frequency on TOUT by using our formula above;

$$TOUT = \frac{20\text{ MHz}}{(0+1) \times (1+1)} = 10\text{ MHz}$$

Chapter 5 of TMS320C5x User's Guide has more detailed information on the operation of its timer.

- 'MAR *, AR0' instruction sets ARP (auxiliary register pointer) to AR0 for use when accessing global memory.
- The accumulator is loaded, using the LACC command, with 1000b which is then used in the subsequent line.
- For configuring the serial port, SPC requires two consecutive writes. The first write must write '0' to bits 6 & 7 to reset the transmitter and the receiver. During the first write, bits 0-5 may be written to for configuration. We will set bit 3 high to indicate frame synchronization pulses are required for transmission and reception of each frame. The rest of the bits can be set to zero.

'LACC #0008h' Instruction loads the immediate value to the accumulator. 'SACL SPC' instruction writes this value to the SPC register resetting the transmitter and the receiver and setting bit 3 enabling frame synchronization pulses. The following two instructions perform the second write to SPC enabling the reset transmitter and receiver by writing '1's to bit 6 & 7. Frame synchronization bit is kept set at '1'. This completes the initialisation of the serial port. Chapter 5 of TMS320C5x User's Guide has more detailed information on the function of each bit in SPC.

When not running in continuous mode, AIC requires three signals for serial communication; the data line, a line to initialise the transfer and a line to clock the transfer. Frame synchronisation pulse is needed to initialise the transfer.

Resetting the AIC

```

lacc    #080h          ; Initialise 8000h-FFFFh global memory
sach    DXR            ; Store high acc to DXR
sac1    GREG           ; Store to global memory register
lar     AR0,#0FFFFh   ; Set AR0 register
rpt     #10000         ; Set repeat counter
lacc    *,0,AR0       ; Access global memory
sach    GREG           ; Disable global memory
;-----;

```

Figure 38: Reset AIC

- Once a clock has been applied to the AIC, we need to ensure that the device is placed into a known state by performing a reset operation. The !BR pin of the TMS320C50 is connected to the reset pin of the AIC. The !BR is driven low when external global memory is accessed.

- Global memory is an area which can be accessed by more than one processor. The TMS320C50's address space is divided up into local and global sections; the local is for the DSP processor and the global sections are for inter-processor communications. A memory mapped register, GREG, specifies the sections of memory that are to be defined as global.
- 'LACC #080h' instruction loads the accumulator with 80h, leaving the upper accumulator filled with zeros. Remember that the accumulator is a 32 bit register.
- The Data Transmit Register 'DXR' is flushed, by writing the zeroed upper accumulator to DXR.
- 'SACL GREG' instruction causes the contents of the low accumulator to be placed into the register GREG, indicating that memory space 8000h to FFFFh is defined as global memory.
- 'LAR AR0 , #0FFFFh' loads the auxiliary register, AR0 with the number 0FFFFh. We shall use this register to access the address 0FFFFh.
- Note that this instruction also sets ARP to point to AR0.
- 'RPT #10000'. The repeat counter is set to repeat the next line 10,000 times.
- 'LACC *, 0, AR0' The accumulator is loaded with the contents of the memory location pointed to AR0. i.e. global memory location FFFFh is accessed 10,000 times. This drives the !BR pin low sufficiently long to reset the AIC. (Clock cycle = 25ns BR pin is low for 25x10000 ns = 250µs.)
- 'SACH GREG'. The GREG register is then zeroed by loading the upper contents of the accumulator, turning global memory off.

Setting AIC registers

This section of the code loads the AIC registers with the values required to set the right sampling rates and filter cut-offs. A reference is made to a function 'AIC2ND'. This is listed separately in the following section.

To set the AIC registers, accumulator of 'C50 is loaded with correct values and then these values are transmitted to the AIC through the serial port of 'C50. Figure 39 shows a simplified diagram of AIC registers and its bit placements. This figure should help with understanding of the following code fragment.

Bit positions															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
X	X	TA Register					X	X	RA Register					0	0
X	X	TB Register					X	X	RB Register					1	0
X	X	X	X	X	X	X	X	Control Register					1	1	

Figure 39: AIC Registers

```

;-----;
ldp    #TA          ; Load data page TA
setc   SXM         ; Set SXM for sign extension mode
lacc   TA,9        ; Load accumulator with TA data
add    RA,2        ; Load accumulator with RB data
call   AIC2ND      ; Call function AIC2ND
;-----;
ldp    #TB          ; Load data page TB
lacc   TB,9        ; Load accumulator with TB data
add    RB,2        ; Load accumulator with RB data
add    #02h        ; Add 10b to set status bits
call   AIC2ND      ; Call function AIC2ND
;-----;
ldp    #AIC_CTR     ; Load data page AIC_CTR
lacc   AIC_CTR,2   ; Initialised control register
add    #03h        ; Add 11b to set status bits
call   AIC2ND      ; Call function AIC2ND
ret                    ; Return from call

```

Figure 40: Setting AIC Registers

- The first register to be set is the TA register. 'LDP #TA' instruction points the data page pointer to the data page where TA resides. This section starts at absolute location 0FFFFh which was set in the 'SETTINGS' section.
- Secondly, the SXM or sign extension mode is set. This instruction allows sign extension when accumulator is loaded.
- 'LACC TA,9'. The accumulator is loaded with TA. A shift of nine places left ensures that the data is stored in the correct place within the word and therefore appears in the correct location in the TA register when transmitted. (Figure 39) Sign extension ensures that all high bits are set to zero since the value of TA is 17. Figure 41 shows the contents of TA and accumulator upon completion of this instruction.

TA															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Low accumulator															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 41: Contents of TA and Accumulator

- 'ADD RA,2'. RA is added to the accumulator, again left shifted by 2 places to ensure that it is in the correct location. Figure 42 shows the contents of RA and contents of the accumulator after the execution of this instruction. The least two significant bits are left as zero. These two bits tell the AIC to load the data word into the A counter (TA).

RA															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

low accumulator															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0

Figure 42: Contents of RA and Accumulator

- This data packet is then sent to the AIC by calling AIC2ND.
- The second register to set is TB. The appropriate data page is selected by 'LDP #TB' instruction.
- The accumulator is loaded with TB. A shift of nine places left ensures that the data is stored in the correct place within the word. The following figure shows the contents of TB and the accumulator after the instruction is executed.

TB															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1

low accumulator															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0

Figure 43: Contents of TB and Accumulator

- 'ADD RB, 2'. RB is added to the accumulator, again left shifted by 2 places to ensure that it is in the correct location.
- 'ADD #02h'. The least two significant bits are changed to 10, indicating that the word is intended for the B counter (RB). The following figure shows the contents of RB and the accumulator after the execution of the two instructions.

RB															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1

Low accumulator															
d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0
0	1	0	0	1	0	1	0	1	0	0	1	0	1	1	0

Figure 44: Contents of RB and Accumulator

- This data packet is then sent to the AIC by calling AIC2ND.
- Writing to registers TA, RA, TB and RB sets the sampling frequency. The formula is:

$$SAMPLING \ FREQUENCY = \frac{MCLK}{TA \times 2 \times TB}$$

MCLK is tied to TOUT which has a frequency of 10MHz as calculated above. Remember:

$$TOUT = \frac{CLKOUT1}{(TDDR+1) \times (PRD+1)} = \frac{20 \text{ MHz}}{(0+1) \times (1+1)} = 10 \text{ MHz}$$

TA and TB were just loaded with values of 17 and 37 respectively. Therefore the actual sampling frequency is:

$$SAMPLING \ FREQUENCY = \frac{10 \text{ MHz}}{17 \times 2 \times 37} = 8 \text{ KHz}$$

- The last register that needs to be set is the AIC control register. The data is held on data page AIC_CTR which is accessed by LDP instruction.
- The accumulator is loaded with AIC_CTR value of 28h with a shift of 2. This sets the transmit and receive sections to synchronous, sets gain of the preamplifier to 4 and deletes the bandpass filter.
- The least two significant bits are changed to 11b by adding 03h. This indicates that the word is intended for the AIC control register. Figure 45 shows the final content of the control register of AIC.

Control Register							
d7	d6	d5	d4	d3	d2	d1	d0
1	0	1	0	0	0	1	1

d2 = 0 / 1 deletes / inserts bandpass filter

d3 = 0 / 1 disables / enables loopback

d4 = 0 / 1 disables / enables AUX IN+ and AUX IN- pins

d5 = 0 / 1 Asynchronous / synchronous transmit and receive sections

d6 = 0 / 1 gain control bits

d7 = 0 / 1 gain control bits

d7	d6	Gain
0	0	1
0	1	2
1	0	4
1	1	1

Figure 45: Bits of AIC control register

- The data packet is the sent to the AIC by calling AIC2ND
- RET returns control to the main body of the program.

Writing to AIC using AIC2ND

```

;-----;
AIC2ND  ldp    #0          ; Load data page zero
        sach   DXR        ; Store upper acc to DXR
        clrc   INTM       ; Enable interrupts
        idle   ; Idle until interrupt - TINT
        add    #6h,15     ; Set 2 LSBs of upper acc high
        sach   DXR        ; Store upper acc to DXR
        idle   ; Idle until interrupt - TINT
        sac1   DXR        ; Store lower acc to DXR
        idle   ; Idle until interrupt - TINT
        la1   #0          ; Store lower acc to DXR - flushing
        sac1   DXR        ; make sure the word got sent
        idle   ; Idle until interrupt - TINT
        setc   INTM       ; Disable interrupts
        ret     ; Return from call
;-----;

```

Figure 46: Writing to AIC

- AIC2ND is the name of the function that is called to send information held in the accumulator of 'C50 to the relevant AIC registers. The routine is called three times in total, once for the A register (TA, RA), once for the B register (TB, RB) and also once for the AIC control register.
- 'LDP #0' sets the data page to zero.
- 'SACH DXR' writes to the serial port register (DXR) which is located on data page zero. The upper half of the accumulator is always zero with the lower half containing the information to be loaded to the counters. The higher word is written to DXR to flush the register.
- Interrupts are then enabled by clearing the INTM bit which is necessary when writing to the serial port.
- 'IDLE' then waits for the TINT interrupt.
- On TINT: DXR is written to XSR which directly passes to the AIC.
- 'ADD #6h, 15' adds 6h to the accumulator, followed by a shift of 15 steps to the left. This sets the least significant bits of the upper half of the accumulator to 11. When this is sent to AIC, it prepares for register alteration. AIC then uses the next data packet it receives to alter the appropriate register.
- 'IDLE' ensures that the processors waits until the data packet containing '11' is sent before writing to the serial port register again.
- The low half of the accumulator is then sent to DXR, the half which contains the data to be written to the registers. Note that the two least significant bits tell the AIC in which register to store the data. (Figure 39).
- To finish, zero is written to the DXR and sent on the next interrupt.
- Interrupts are disabled before return.
- RET returns the execution to the AIC routine.

2.9.6 Main Program

The main program only performs one function; that is to wait for the next sample to arrive. The AIC interrupts the main program which then branches to service the interrupt before returning to the main program to wait for another interrupt.

```

;=====;
;=MAIN PROGRAM                                     =;
;=====;
WAIT   idle           ; Idle until interrupt
      nop             ; No operation
      nop             ; No operation
      b      WAIT     ; Branch to WAIT
;=====;
    
```

Figure 47: PASS Main Program

- WAIT is a label as a reference to branch back. The idle command waits in a low power mode until an interrupt. As the interrupts are enabled, ISR is executed from this point.
- On returning from the interrupt service routine, the main program branches back to WAIT where it begins once again.

2.9.7 Interrupt Service Routines

The PASS program reads the serial input port of the DSK and writes the output to the serial output port. This reading and writing routine is performed at regular intervals within the program. The AIC controls the times when this takes place by its sampling rate, in this case 8,000 times a second.

```

;=====;
;= RECEIVE INTERRUPT SERVICE ROUTINE              =;
;=====;
RX  ldp   #0           ; Set data page pointer to zero
     lamm  DRR         ; Read data from serial input register
     and  #0FFFC      ; Clear lower two (status) bit from acc
     samm DXR         ; Write lower acc to DXR
     rete                    ; Return to main program
;=====;
;=TRANSMIT INTERRUPT SERVICE ROUTINE              =;
;=====;
TX  rete                    ; Return to main program
;=====;
    
```

Figure 48: PASS Interrupt Service Routine

On receiving an interrupt, the program branches to ISR called RX. This is defined within the interrupts section of the code. The ISR takes the sample from the input serial port, and outputs it once again to the output serial port. The contents of DXR are automatically transmitted. When DXR is empty, it generates a transmitter interrupt. On return from the receiver ISR to the main program, processor now goes into TX ISR and returns.

- RX is the label of the interrupt service routine.

- 'LDP #0' sets the current data page to zero which is required as both DRR and DXR as they are located on data page zero.
- 'LAMP DRR' loads the accumulator with the value held in DRR, the serial port input register. This 16-bit value is a digitized sample from the serial input of the DSK.
- The codec only gives a resolution of 14 bits, the two least significant bits are used as status bits and do not reflect the signal. By using the AND instruction, these two least significant bits can be zeroed before sending the sample to the output.
- 'SAMP' writes the lowest half of the 32 bit accumulator to DXR the serial port transmit register.
- 'RETE' returns control of the program to the main program with interrupts enabled.
- TX is the label of the transmit ISR. This ISR has only a 'RETE' instruction. So when a transmitter interrupt occurs, it automatically sends the contents of DXR via the serial port.

We have now completed the review of PASS. Now by modifying PASS, we shall write our first piece of real DSP code.

Before we proceed to the next section you could do a few experiments. Input a sinusoidal signal at 20kHz and observe the output. Has the program stopped functioning? (Hint: Consider the sampling frequency).

```

=====
;=INFORMATION          PROGRAM          ; pass through          =;
;=====              FILE NAME        ; pass.asm              =;
;=                     INCLUDE FILES    ; no include files      =;
;=                     PROCESSOR        ; TMS320C5x             =;
;=                     AUTHOR           ; Matt Byatt - Texas Instruments =;
;=                     DATE/VERSION     ; July 1996 / Ver 1.00   =;
;=====
;=DESCRIPTION         =;
;=====              =;
;=                   =;
;= Pass:              A program that initialises the TMS320C50 DSK processor =;
;=                   and AIC codec. At a sampling rate of 8kHz, the board's =;
;=                   serial input port is read to the accumulator.           =;
;=                   =;
;=                   Within the same ISR, the contents of the accumulator are =;
;=                   written to the serial output port without modification. =;
;=                   =;
;=                   The constant read and writing continually passes the =;
;=                   input signal to the output.                             =;
;=                   =;
;=                   =;
;=                   =;
;=====
;=SPECIFICATION       FREQUENCY         ; 8kHz                  =;
;=====              FILTER           ; not present           =;
;=                     MISC            ; TLC320C40 initialised =;
;=====
;=                   =;
;=                   .mmregs          =;
;=                   =;
;=SETTINGS            =;
;=====
;=                   =;
;=                   .ds              0f00h          ; Assemble to data memory
TA      .word    17                    ; TA AIC value - Fcut = 8 KHz
RA      .word    17                    ; RA AIC value - Fcut = 8 KHz
TB      .word    37                    ; TB AIC value - Fs = 2*Fcut

```

```

RB      .word    37          ; RB AIC value · Fs = 2*Fcut
AIC_CTR .word    28h        ; [LP xx G1 G0 | SY AX LB BP|
; +-----+ +-----+
; |          GAIN      | | | +--- BP Filter
; |          Synch    +-+ | +----- Loopback
; |          Auxin    +---+
; |          + (sinx)/x filter
;=====
;=INTERRUPT SETTINGS
;=====
      .ps      080ah        ; Assemble to program memory
RINT   b      RX          ; Branch to RX on receive interrupt
XINT   b      TX          ; Branch to TX on transmit interrupt
;=====
;=PROCESSOR INITIALISATION
;=====
      .ps      0a00h        ; Assemble to program memory
      .entry                    ; Mark program entry point
      .text                    ; Text
START  setc    INTM        ; Disable interrupts
      ldp     #0             ; Set data page pointer to page zero
      splk   #0834h,PMST    ; Write 16 bit pattern to PMST register
      lacc   #0             ; Load accumulator with number zero
;-----
      samm   CWSR          ; Set software wait state to zero
      samm   PDWSR         ; Set software wait state to zero
      splk   #022h,IMR     ; Using XINT syn TX & RX
      call   AIC           ; Initialize AIC and enable interrupts
;-----
      clrc   OVM           ; Overflow mode is set to zero
      spm    0             ; Product shift mode is set to zero
      splk   #012h,IMR     ; Mask interrupts
      clrc   INTM         ; Enable interrupts
;=====
;=MAIN PROGRAM
;=====
WAIT   idle                    ; Idle until interrupt
      nop                      ; No operation
      nop                      ; No operation
      b      WAIT            ; Branch to WAIT
;---- end of main program ----; Comment line
;=====
;= RECEIVE INTERRUPT SERVICE ROUTINE
;=====
RX     ldp     #0             ; Set data page pointer to zero
      lamm   DRR           ; Read data from serial input register
      and   #0FFCh         ; Clear lower two (status) bit from acc
      samm   DXR          ; Write lower acc to DXR
      rete                    ; Return to main program
;=====
;=TRANSMIT INTERRUPT SERVICE ROUTINE
;=====
TX     rete                    ; Return to main program
;=====
;=BOARD INITIALISATION
;=====
AIC    splk   #20h,TCR      ; To generate 10 MHz from Tout
      splk   #01h,PRD      ; Load period counter with 1
      mar   *,AR0         ; Modify AR pointer (used with GREG)
      lacc   #0008h        ; Load acc with 08h
      sacl  SPC           ; Store acc in SPC
      lacc   #00C8h        ; Load acc with 0C8h
      sacl  SPC           ; Set and reset SPC register
;-----
      lacc   #080h        ; Initialise 8000h to FFFFh as global memory
      sach  DXR          ; Store high acc to DXR
      sacl  GREG         ; Store to global memory register

```

```

lar    ARO,#0FFFFh    ; Set ARO register
rpt    #10000         ; Set repeat counter
lacc   *,0,ARO        ; Access global memory
sach   GREG           ; Disable global memory
;-----;
ldp    #TA            ; Load data page TA
setc   SXM            ; Set SXM for sign extension mode
lacc   TA,9           ; Load accumulator with TA data
add    RA,2           ; Load accumulator with RB data
call   AIC2ND         ; Call function AIC2ND
;-----;
ldp    #TB            ; Load data page TB
lacc   TB,9           ; Load accumulator with TB data
add    RB,2           ; Load accumulator with RB data
add    #02h           ; Add 10b to set status bits
call   AIC2ND         ; Call function AIC2ND
;-----;
ldp    #AIC_CTR       ; Load data page AIC_CTR
lacc   AIC_CTR,2      ; Initialized control register
add    #03h           ; Add 11b to set status bits
call   AIC2ND         ; Call function AIC2ND
ret                                          ; Return from call
;-----;
AIC2ND ldp    #0        ; Load data page zero
sach   DXR            ; Store upper acc to DXR
clrc   INTM           ; Enable interrupts
idle   ; Idle until interrupt - TINT
add    #6h,15         ; Set 2 LSBs of upper acc high
sach   DXR            ; Store upper acc to DXR
idle   ; Idle until interrupt - TINT
sac1   DXR            ; Store lower acc to DXR
idle   ; Idle until interrupt - TINT
lac1   #0             ; Store lower acc to DXR - flushing
sac1   DXR            ; make sure the word got sent
idle   ; Idle until interrupt - TINT
setc   INTM           ; Disable interrupts
ret                                          ; Return from call
;-----;
.end                                         ; End of program

```

Figure 49: PASS through Program

2.10 A First Program

PASS will provide the basis for this section. We will modify various sections of PASS to incorporate constants, different sample rates and different functions. Our aim is to write a program that outputs a sine wave at about 800Hz. Let us firstly study some theory that will help us with this program.

2.10.1 Theory

There are a number of ways to generate sine waves using DSPs. A look up table could be used which would pre-store the sample values of a sine wave. Another method is to design a digital oscillator. This is done by designing an unstable Infinite Impulse Response (IIR) filter. Although we will use this method, we shall take an equation which represents such an oscillator and use it. We shall not design it from scratch. We shall call this method **'the equation method'**. Equation method is quite accurate and results in good quality sine waves. Our sine wave equation is:

$$y_{(n)} = A \cdot y_{(n-1)} + B \cdot y_{(n-2)} + C \cdot x_{(n-1)}$$

This is a second order difference equation whose unit impulse response is $\sin(n\omega T)$.

The unit impulse is $x(n-1) = 1$. Applying this unit impulse to the equation above.

$$n = 0 \quad y_{(0)} = A \cdot y_{(-1)} + B \cdot y_{(-2)} + 0 = 0.$$

$$n = 1 \quad y_{(1)} = A \cdot y_{(0)} + B \cdot y_{(-1)} + C = C$$

$$n = 2 \quad y_{(2)} = A \cdot y_{(1)} + B \cdot y_{(0)} + 0 = A \cdot y_{(1)}$$

$$n = 3 \quad y_{(3)} = A \cdot y_{(2)} + B \cdot y_{(1)}$$

$$n = k \quad y_{(k)} = A \cdot y_{(k-1)} + B \cdot y_{(k-2)}$$

Once we evaluate the numerical values for A , B and C , we can calculate sine wave samples. The following equations relate the coefficients to the frequency of the sine wave and the sampling frequency:

$$A = 2 \cos(\theta_{DEGREES})$$

$$\theta_{DEGREES} = \frac{f_{DESIRED}}{f_{SAMPLING}} \cdot 360$$

$$B = -1$$

$$C = \sin(\theta_{DEGREES})$$

It is important to calculate 'A' and 'C' correctly and accurately for a good quality sine wave.

θ is governed by the Nyquist criteria and limits $f_{DESIRED}$ to a maximum of half the sampling frequency. Representing a sinusoid at this limit would give a low resolution with significant amounts of harmonic distortion.

As θ decreases, then the quality of the signal increases. In effect, the more samples per period the greater the accuracy.

2.10.2 Calculations

The sampling frequency is already set at 8,000Hz. Let us decide on the frequency of the sine wave and calculate the coefficients:

Sine wave output: 800Hz

Sampling frequency: 8,000Hz

Compute θ first:

$$\theta_{DEGREES} = \frac{f_{DESIRED}}{f_{SAMPLING}} \cdot 360 = \frac{800}{8000} \cdot 360 = 36^\circ$$

Placing this value in the coefficients equation above for A:

$$A = 2 \cos(\theta_{DEGREES}) = 2 \cos(36^\circ)$$

$$A = 2 \times 0.809017$$

$$B = -1$$

$$C = \sin(\theta_{DEGREES}) = \sin(36^\circ)$$

$$C = 0.58778$$

Our equation now becomes:

$$Y_{(n)} = 2 \times 0.809017 Y_{(n-1)} - Y_{(n-2)}$$

with an initial C value of 0.58778

Now we know all the coefficients but how can we use these with our DSP? The TMS320C5x DSP is a fixed point device and therefore can only work using integer numbers. Our coefficients are fractional numbers. Using the conventional methods of implementation will not provide the accuracy required. Another means of representation must be found to ensure that we achieve the correct sine wave frequency.

If we assume that a full positive 16-bit register (7FFFh) equals 1 and negative 16 bit register equals -1 (FFFFh) fractional numbers can be represented. Therefore, +0.5 would be 4000h and so on. We can therefore represent our fractional coefficients by using this process, which is called Q15 format:

$$Q15 = 2^{15} \times fraction_{DECIMAL}$$

Q15 format gives us higher accuracy. Remember that we shall be using this conversion formula in our programs. Let us now convert our fractional coefficients to Q15 format:

	Fraction	Decimal Q15 (rounded)	Hexadecimal Q15
A ⇒ Q15	2 x 0.809017	2 x 26510	2 x 678Eh
C ⇒ Q15	0.5877853	19260	4B3Dh

* As our coefficient A is greater than 1, we must reduce it by an integer factor to prevent overflow. This must be compensated for later in the program.

2.10.3 Example of Q15 multiplication

Now that our sine wave coefficients are converted into Q15 format, we should be able to calculate the sample values. But for this we need to be able perform a multiplication of an integer with a Q 15 number. Let us evaluate:

$$A \times SAMPLE$$

in the sinusoidal equation in Q15 format. Multiply a sample value, say $Y_{(n-1)}$ by A.

Let us take an arbitrary sample value $y_{(n-1)} = 0.5$.

$$A \times \text{SAMPLE} = 2 \times 0.809017 \times 0.5$$

$$A \times \text{SAMPLE} = 0.809017$$

$$A \times \text{SAMPLE (in Q15)} = 0.809017 \times 2^{15} = 26510_{\text{DECIMAL}} = 678E_{\text{HEXADECIMAL}}$$

Now let us do the same calculation in Q 15 format. A and $y_{(n-1)}$ in Q 15 format:

$$A = 2 \times 678E_{\text{HEXADECIMAL}} = 2 \times 26510_{\text{DECIMAL}}$$

convert $y_{(n-1)}$ into Q15 format:

$$y_{(n-1)} = 0.5_{\text{DECIMAL}} \times 2^{15} = 16384_{\text{DECIMAL}}$$

$$y_{(n-1)} = 4000_{\text{HEXADECIMAL}}$$

To prevent overflow, multiply the sample only with the fractional Q15 representation of A first:

$$\text{multiplication} = 26510_{\text{DECIMAL}} \times 16384_{\text{DECIMAL}} = 434\ 339\ 840_{\text{DECIMAL}}$$

$$\text{multiplication} = 678E_{\text{HEXADECIMAL}} \times 4000_{\text{HEXADECIMAL}} = 19E3\ 8000_{\text{HEXADECIMAL}}$$

Now the result can be scaled to full A by multiplying it by 2

$$\text{multiplication} = 2_{\text{DECIMAL}} \times 434\ 339\ 840_{\text{DECIMAL}} = 868\ 679\ 680_{\text{DECIMAL}}$$

$$\text{multiplication} = 2_{\text{HEXADECIMAL}} \times 19E3\ 8000_{\text{HEXADECIMAL}} = 33C7\ 0000_{\text{HEXADECIMAL}}$$

The following table shows the result in binary form as it would exist in a register:

Hex	Binary							
33C7 0000	0011	0011	1100	0111	0000	0000	0000	0000

Left shift by one bit to normalise: On completing a fixed point multiplication, the two most significant bits of the result are sign bits. Because the multiplication is done using two's complement numbers, we need only one sign bit. Left shift by one bit gets rid of one extra sign bit, leaving only one.

Hex	Binary							
678E 0000	0110	0111	1000	1110	0000	0000	0000	0000

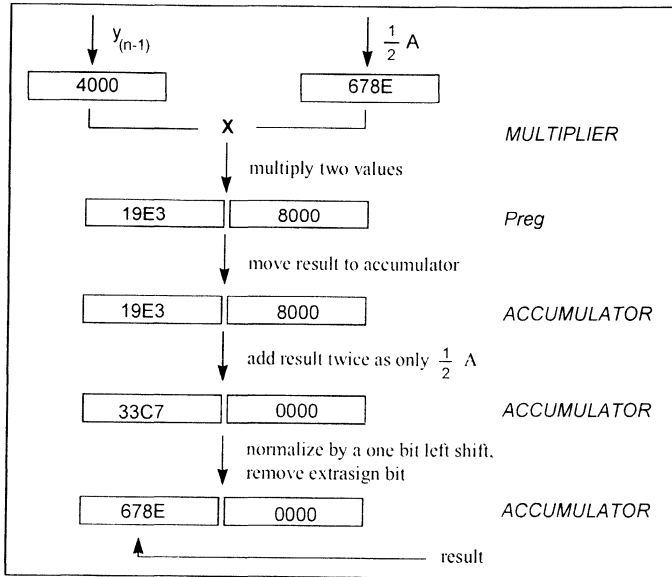


Figure 50: Multiplier Example

Time should be taken in verifying the details for yourself to understand this multiplication process.

2.10.4 The Program

The program should be designed to calculate the samples of the sine wave. This is straightforward since we already know the equation, initial value 'C' and coefficients 'A' and 'B'. Figure 51 shows how the samples are calculated using the equation. It is important to note that to compute the first sample $y(n)$, $y(n-1)$ is equal to 'C' and $y(n-2)=0$.

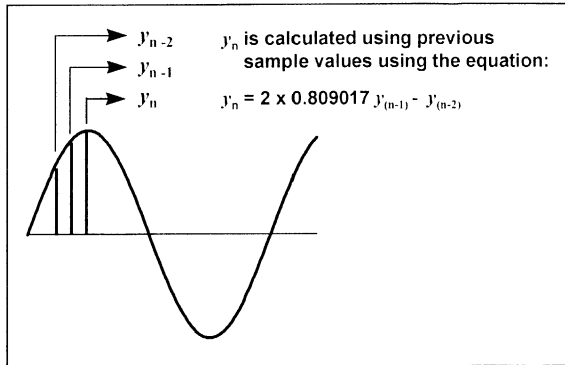


Figure 51: Sine wave generation

The main program is simple and only waits for interrupts. Transmit ISR calculates the new sine wave sample value and outputs it to AIC. Figure 52 shows a detailed flowchart of the program. Let us now make modifications to PASS to convert it to SINE.

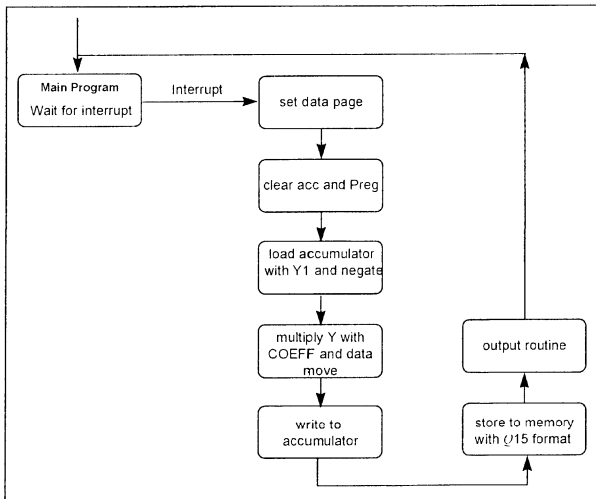


Figure 52: Sine Wave Program

2.10.5 From PASS to SINE

Let us examine the modifications section by section.

```

;=====;
; .mmregs
;=====;
;=SETTINGS
;=====;
        .ds      0f00h      ; Assemble to data memory
        .data
TA       .word   17        ; TA AIC value - Fcut = 8 kHz
RA       .word   17        ; RA AIC value - Fcut = 8 kHz
TB       .word   37        ; TB AIC value - Fs = 2*Fcut
RB       .word   37        ; RB AIC value - Fs = 2*Fcut
AIC_CTR .word   8h        ; |LP xx G1 GO | SY AX LB BP|
; +-----+
; |         GAIN      | | | +-+ BP Filter
; |         Synch    +-+ | +----- Loopback
; |         Auxin    +-----+
; |         + (sinx)/x filter
; +-----+
Y        .word   9630      ; 19260/2=C/2 Initial Y value, Sin()
Y1       .word   0000h     ; Secondary Y value
; +-----+
        .ds      01000h    ; Assemble to data memory
COEFF    .word   0678Eh    ; Sine wave frequency coefficient(A/2)Cos()
;=====;

```

Figure 53: Sine Wave Generator Settings

- In PASS the coefficients assembled into data memory were only used to initialise the DSK's AIC circuitry. In SINE we shall store some more data values and reserve data memory for storage.
- 'Y' is an initial sample value to start off the sine calculation. ($Y = C = y_{(1)}$). After the first calculation, it is used as storage space for the same sample value. 'Y1' is a storage space for secondary sample value. ($Y1 = y_{(n+2)}$). Note that one-half of C is used and this will be compensated for later.
- 'COEFF' is the value of 'A/2' coefficient in the sine equation. It determines the frequency of the sine wave. Again, this is one-half of the actual 'A' and will be compensated for in the calculations later. The reason for using half of the values is to prevent overflow.

```

;=====;
;=PROCESSOR INITIALISATION
;=====;
        .ps 0A00h      ; Assemble to program memory
        .entry        ; Mark program entry point
        .text        ; Text
START  setc   INTM     ; Disable interrupts
      ldp    #0        ; Set data page pointer to page zero
      spk   #0834h,PMST ; Write 16 bit pattern to PMST register
      lacc  #0        ; Load accumulator with number zero
; +-----+
      samm  CWSR      ; Set software wait state to zero
      samm  PDWSR     ; Set software wait state to zero
      spk   #022h,IMR ; Using XINT syn TX & RX
      call  AIC       ; Initialise AIC and enable interrupts
; +-----+
      clrc  OVM       ; Overflow mode is set to zero

```

```

spm      0          ; Product shift mode is set to zero
splk     #022h,IMR ; Mask interrupts
clrc     INTM      ; Enable interrupts
;=====;

```

Figure 54: Sine Wave Processor Initialisation

- Processor is initialised as in PASS.
 - We are using the transmit interrupt, not the receive. This is because we are generating the sine wave and transmitting the samples to the AIC. Therefore the interrupt mask is changed to mask the receive interrupts and to enable the transmit interrupts.
- 'SPLK #022h, IMR' writes 22h to the IMR register enabling the transmit interrupts.

```

;=====;
;=MAIN PROGRAM                                     =;
;=====;
WAIT  idle          ; Wait until interrupt
      nop           ; No operation
      nop           ; No operation
      b            WAIT ; Branch to WAIT
;---- end of main program ----; Comment line
;=====;

```

Figure 55: Main program

- Main program does nothing else but to wait for interrupts.

```

;=====;
;= TRANSMIT INTERRUPT SERVICE ROUTINE             =;
;=====;
TX
      ldp          #Y          ; Load data page Y
      zap          ; Clear acc and product register
;-----;
      lacc         Y1,15      ; Load acc with Y1 with shift of 15
      neg         ; Negate accumulator
      macd        COEFF,Y    ; Multiply with data move (Y * COEFF)
      apac        ; Double the result
      apac        ; since A/2 and C/2
      sach        Y,1        ; remove sign bit
;-----;
      lacc         Y          ; Load Y into lower acc
      ldp          #0         ; load data page zero
      and         #0FFFCh    ; Clear lower two (status) bit from acc
      samm        DXR        ; Write lower acc to DXR
      rete        ; Return to main program
;=====;
;=RECEIVE INTERRUPT SERVICE ROUTINE             =;
;=====;
RX
      rete        ; Return to main program
;=====;

```

Figure 56: Sine Wave ISR

- The ISR generates a sine wave using three values stored in memory;

$$Y = Y_{(n-1)} \quad \text{ISR result calculated last time, initial value } C/2$$

$$Y1 = Y_{(n-2)} \quad \text{ISR result calculated time before last}$$

$$\text{COEFF} = A/2 \quad \text{coefficient in the sine equation.}$$

You can follow the calculations below from Figure 57 which gives a pictorial representation of register events.

- Firstly, as we are using the data page system of addressing, we select the data page that contains our information - data page Y .
- We then clear the accumulator and the product register of previous values using 'ZAP'. This is essential in order to ensure that the MACD instruction used later does not use the previous multiplication result.
- $Y1$ is loaded into the accumulator with a shift of 15 places to the left. This places $Y1$ in upper accumulator and converts it into $Q15$ format. (Remember $Y1$ in $Q15$ format = $2^{15} \times Y1$).
- 'NEG' negates the contents of the accumulator to give $-Y1$ since $B = -1$.
- This is a complex instruction, demonstrating what the 'C5x can achieve in a single cycle. 'MACD COEFF, Y' multiplies the COEFFicient by the Y value with a **data move**. The result of the multiplication is written to the product register. The **data move** copies the contents of Y into $Y1$ before the multiplication (the next memory location). Thus $Y1$ becomes Y , ready for next time round.
- The product is added to the accumulator twice. This compensates for the fact that our COEFF and 'C' values are only halves of actual values ($t = 2 \times 678Eh$ & $C = 2 \times 9630$).
- This instruction again demonstrates the power of the 'C5x. 'SACH Y, 1' instruction copies the accumulator into a temporary register. Left shifts the entire register by one bit and then copies the upper 16-bits into data memory location 'Y'. The process removes the extra sign bit gained in two's complement multiplication.
- Finally, the sine wave output value is loaded into the lower accumulator before the lowest two bits are cleared and the result is written to the DXR.

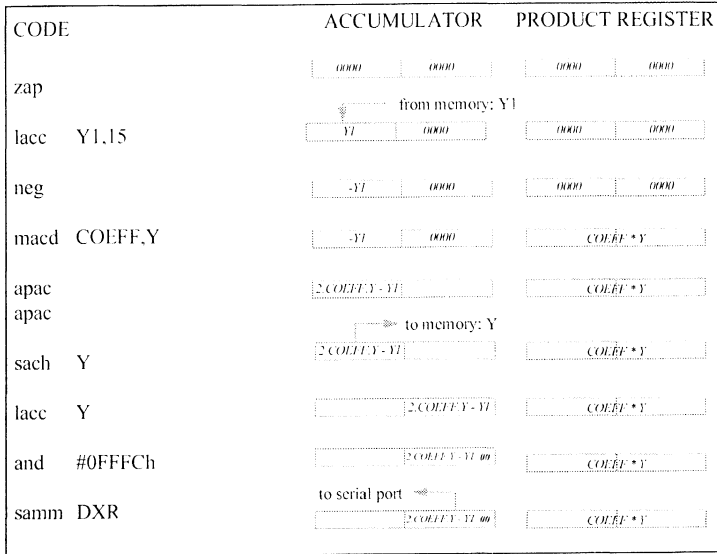


Figure 57: Register Based MAC

2.10.6 Running the program

With a completed program, assemble the SINE.ASM code and then run the program using either the loader or the debugger as with PASS.ASM.

A sine wave of approximately 800Hz should be created at the output of the AIC. If your modifications have not quite worked, we have included a working version of the software on the disk. Compare your modifications against this program and spot the mistakes.

```

;=====
;=INFORMATION          PROGRAM          ; sine wave generator          =
;=====              FILE NAME        ; sine.asm                    =
;                      INCLUDE FILES    ; no include files           =
;                      PROCESSOR        ; TMS320C5x                  =
;                      AUTHOR           ; Matt Byatt - Texas Instruments =
;                      DATE/VERSION     ; July 1996 / Ver 1.00       =
;=====
;=DESCRIPTION
;=====
;
;= Sine:               A program that initialises the TMS320C50 DSK processor
;                      and AIC codec. At a sampling rate of 8kHz, the DSK
;                      performs a serial input port dummy read.
;
;                      An internal digital oscillator program generates a sine
;                      waveodification.
;=====

```



```

;=          The frequency of the sine wave can be altered by changing ;=
;=          the coefficient values within the data table.              ;=
;=          ;=
;=          ;=
;=          ;=
;=====
;=SPECIFICATION          FREQUENCY          ; 8kHz                      ;=
;=====          FILTER          ; not present                  ;=
;=          MISC          ; TLC320C40 initialised                ;=
;=====
        .mmregs
;=====
;=SETTINGS
;=====
        .ds      0f00h          ; Assemble to data memory
        .data
TA      .word    17             ; TA AIC value - Fcut = 8 KHz
RA      .word    17             ; RA AIC value - Fcut = 8 KHz
TB      .word    37             ; TB AIC value - Fs = 2*Fcut
RB      .word    37             ; RB AIC value - Fs = 2*Fcut
AIC_CTR .word    8h             ; [LP xx G1 G0 | SY AX LB BP|
; +-----+-----+
; |         GAIN         | | | +--- BP Filter
; |         Synch  ---+  | | +----- Loopback
; |         Auxin  ----+  |
; +-----+ (sinx)/x filter
;-----+
Y      .word    9630            ; (19260/2) Initial Y value, sin()/2
Y1     .word    0000h          ; Secondary Y value
;-----+
        .ds      01000h        ; Assemble to data memory
COEFF   .word    0678Dh        ; Sine wave frequency coefficient A/2 = Cos()
;=====
;=INTERRUPT SETTINGS
;=====
        .ps      080ah          ; Assemble to program memory
RINT    b        RX            ; Branch to RX on receive interrupt
XINT    b        TX            ; Branch to TX on transmit interrupt
;=====
;=PROCESSOR INITIALISATION
;=====
        .ps      0a00h          ; Assemble to program memory
        .entry           ; Mark program entry point
        .text           ; Text
START   setc      INTM         ; Disable interrupts
        ldp      #0          ; Set data page pointer to page zero
        splk    #0834h,PMST    ; Write 16 bit pattern to PMST register
        lacc    #0          ; Load accumulator with number zero
;-----+
        samm    CWSR         ; Set software wait state to zero
        samm    PDWSR        ; Set software wait state to zero
        splk    #022h,IMR     ; Using XINT syn TX & RX
        call   AIC           ; Initialize AIC and enable interrupts
;-----+
        clrc    OVM          ; Overflow mode is set to zero
        spm     0            ; Product shift mode is set to zero
        splk    #022h,IMR     ; Mask interrupts
        clrc    INTM         ; Enable interrupts
;=====
;=MAIN PROGRAM
;=====
WAIT    nop          ; Wait until interrupt
        nop          ; No operation
        nop          ; No operation
        b        WAIT  ; Branch to WAIT
;---- end of main program ---- ; Comment line
;=====

```

```

;= TRANSMIT INTERRUPT SERVICE ROUTINE                                     =;
;=====
TX
    ldp    #Y                ; Load data page Y
    zap    ; Clear acc and product register
    ;-----
    lacc   Y1,15            ; Load acc with Y1 with shift of 15
    neg    ; Negate accumulator
    macd   COEFF,Y          ; Multiply with data move (Y * COEFF)
    apac   ; Double the result
    apac   ; Since A/2 and C/2
    sach   Y,1              ; Out of Q15 format
    ;-----
    lacc   Y                ; Load Y into lower acc
    ldp    #0                ; load data page zero
    and    #0FFFC           ; Clear lower two (status) bit from acc
    samm   DXR              ; Write lower acc to DXR
    rete   ; Return to main program
;=====
;=RECEIVE INTERRUPT SERVICE ROUTINE                                     =;
;=====
RX
    rete   ; Return to main program
;=====
;=BOARD INITIALISATION                                               =;
;=====
AIC
    splk   #20h,TCR          ; To generate 10 MHz from Tout
    splk   #01h,PRD          ; Load period counter with 1
    mar    *,AR0             ; Modify AR pointer (used with GREG)
    lacc   #0008h            ; Load acc with 08h
    sacl   SPC               ; Store acc in SPC
    lacc   #00C8h            ; Load acc with 0C8h
    sacl   SPC               ; Set and reset SPC register
    lacc   #080h             ; Initialise 8000h to FFFFh as global memory
    sach   DXR               ; Store high acc to DXR
    sacl   GREG              ; Store to global memory register
    lar    AR0,#0FFFFh      ; Set AR0 register
    rpt    #10000            ; Set repeat counter
    lacc   *,0,AR0           ; Access global memory
    sach   GREG              ; Disable global memory
    ;-----
    ldp    #TA                ; Load data page TA
    setc   SXM               ; Set SXM for sign extension mode
    lacc   TA,9              ; Load accumulator with TA data
    add    RA,2              ; Load accumulator with RB data
    call   AIC2ND            ; Call function AIC2ND
    ;-----
    ldp    #TB                ; Load data page TB
    lacc   TB,9              ; Load accumulator with TB data
    add    RB,2              ; Load accumulator with RB data
    add    #02h              ; Add 10b to set status bits
    call   AIC2ND            ; Call function AIC2ND
    ;-----
    ldp    #AIC_CTR          ; Load data page AIC_CTR
    lacc   AIC_CTR,2         ; Initialized control register
    add    #03h              ; Add 11b to set status bits
    call   AIC2ND            ; Call function AIC2ND
    ret    ; Return from call
;-----
AIC2ND
    ldp    #0                ; Load data page zero
    sach   DXR               ; Store upper acc to DXR
    clrc   INTM              ; Enable interrupts
    idle   ; Idle until interrupt - TINT
    add    #6h,15            ; Set 2 LSBs of upper acc high
    sach   DXR               ; Store upper acc to DXR
    idle   ; Idle until interrupt - TINT
    sacl   DXR               ; Store upper acc to DXR

```

```

idle      ; Idle until interrupt - TINT
lacl     #0      ; Store lower acc to DXR - flushing
sac1     DXR     ; make sure the word got sent
idle     ; Idle until interrupt - TINT
setc     INTM    ; Disable interrupts
ret      ; Return from call
;-----;
.end     ; End of program

```

2.11 From Sine to Musical Notes

Musical notes are a combination of sine waves at different frequencies. Since we can now generate sine waves at practically any frequency, we should be able to generate musical notes.

2.11.1 Background

A musical note consist of a fundamental frequency and a number of harmonics. First harmonic frequency is twice the fundamental frequency and one-half the amplitude of the fundamental. The second harmonic is three times the frequency and one-third of the amplitude of the fundamental and so on. Higher harmonics above the seventh harmonic are negligible. Figure 58 shows a musical note with its seven harmonics.

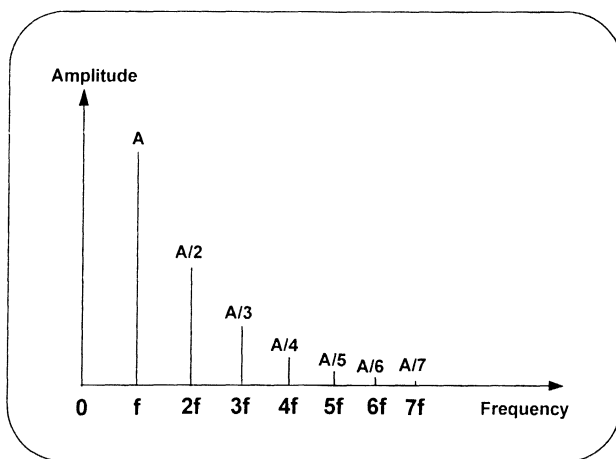


Figure 58: A musical note

The quality of a musical synthesiser relies heavily on the number and quality of harmonics generated to make up a musical note. In fact by modifying the way that the harmonics decay away, one can generate musical notes closely imitating various musical instruments such as a piano or a guitar.

Musical notes are grouped into octaves. An octave consists of seven or more musical notes 'Do, Re, Mi, Fa, Sol, La, Si'. Figure 59 shows the fundamental frequencies of musical notes grouped into octaves.

OCTAVE NOTE FREQUENCIES (Hz)											
NOTE	0	1	2	3	4	5	6	7	8	9	10
DO	13.7	27.5	55.0	110.0	220.0	440.0	880.0	1760.0	3520.0	7040.0	14080.0
RE	15.4	30.9	61.7	123.5	247.0	493.9	987.8	1975.7	3951.3	7902.7	15805.3
MI	16.4	32.7	65.4	130.8	261.6	523.3	1046.6	2093.2	4186.5	8372.9	16745.8
FA	18.4	36.7	73.4	146.8	293.7	587.4	1174.8	2349.7	4699.5	9398.9	18797.8
SOL	20.6	41.2	82.4	164.9	329.7	659.4	1318.8	2637.7	5275.3	10550.6	21101.3
LA	21.8	43.7	87.3	174.7	349.3	698.7	1397.3	2794.6	5589.2	11178.4	22356.8
SI	24.4	49.0	98.0	196.1	392.1	784.3	1568.2	3137.1	6274.1	12548.2	25096.4

Figure 59: Musical Note frequencies and octaves

2.11.2 Implementation

In our musical note generator, we shall implement all frequencies in Octave 6. For simplicity we shall use a single sine wave with no harmonics at each frequency. However, once we are successful in creating the notes, we can then enhance the program to generate musical notes with the related harmonics.

To generate sine waves at each of Octave 6 frequencies, we shall need to calculate the 'A' coefficient value and 'Y' the initial sample value for each frequency. The following table shows these values. Note that 'A' and 'C' values are halved and converted into Q15 ready to be used in our Octave 6 generator.

NOTES	FREQ Hz	COEF A/2	COEF Q15 DEC	Y C/2	Y Q15 DEC
DO	880	0.771	25264	0.319	10453
RE	987.8	0.714	23396	0.35	11469
MI	1046.6	0.681	22315	0.366	11993
FA	1174.8	0.604	19792	0.399	13074
SOL	1318.8	0.51	16712	0.43	14090
LA	1397.3	0.456	14942	0.445	14582
SI	1568.2	0.333	10912	0.472	15466

Figure 60: Coefficients for Octave 6 frequencies

Octave 6 program is designed to play all the musical notes in octave 6 sequentially. The program runs in a loop repeating the notes until stopped. Figure 61 shows the flow of the program. All the functionality of the program is in the transmitter interrupt service routine.

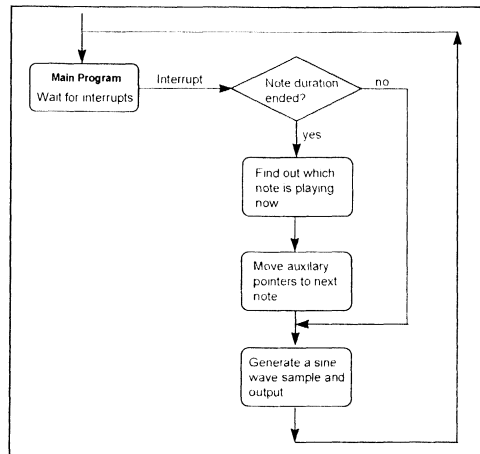


Figure 61: Octave 6 program flow

2.11.3 Octave 6

Let us now examine in detail the implementation of Octave 6 musical notes generator. The AIC settings and sampling frequency remain the same at 8,000Hz. The musical notes table places the constants required to generate each frequency in data memory with appropriate labels. These labels will be used later to access the data values. TIME defines the duration that a musical note is to be played. This is set at approximately half of one second.

```

;-----;
; note table
; fn = f note/f sampling
; Y/2 in Q15 format
; A/2 = cos(fn*360) in Q15 format
;-----;
do          .word 10453      ;880Hz
           .word 0          ;Initial Y = [sin(fn*360)/2] Q15
           ;Secondary Y value
docoef     .word 25264      ;cos(fn*360) in Q15
re         .word 11469      ;987Hz
           .word 0
recoef     .word 23396      ;coeff value
mi         .word 11993      ;1046Hz
           .word 0
micoef     .word 22315      ;coeff
fa         .word 13074      ;1174Hz
           .word 0
;

```

```

facoef      .word  19792      ;
sol         .word  14090      ;1318Hz
           .word  0          ;
solcoef     .word  16712      ;
la          .word  14582      ;1397Hz
           .word  0          ;
lacoef     .word  14942      ;
si          .word  15466      ;1568Hz
           .word  0          ;
sicoef     .word  10912      ;
;-----;
;-- end of note table-----;
;-----;
TIME       .word  1000h      ; initial time, 0.4 seconds
; to change time change this value and the reload value in TX INT
;-----;

```

Figure 62: Octave 6 musical notes data table

Interrupt settings and processor initialisation remain the same with the exception of initialisation of auxiliary registers AR1, AR2 and AR3.

```

;=====;
;=INTERRUPT SETTINGS                                     =;
;=====;
        .ps  080ah          ; Assemble to program memory
RINT   b    RX              ; Branch to RX on receive interrupt
XINT   b    TX              ; Branch to TX on transmit interrupt
;=====;
;=PROCESSOR INITIALISATION                             =;
;=====;
        .ps  0a00h          ; Assemble to program memory
        .entry              ; Mark program entry point
        .text               ; Text
START  setc  INTM           ; Disable interrupts
        ldp   #0             ; Set data page pointer to page zero
        splk #0836h,PMST     ; Write 16 bit pattern to PMST register
        lacc #0             ; Load accumulator with number zero
;-----;
        samm CWSR           ; Set software wait state to zero
        samm PDWSR          ; Set software wait state to zero
        splk #022h,IMR      ; Using XINT syn TX & RX
        call AIC             ; Initialise AIC and enable interrupts
;-----;
        clrc OVM            ; Overflow mode is set to zero
        spm  0              ; Product shift mode is set to zero
        clrc SXM
        lar  ar1,#do         ;point to first note
        lar  ar2,#dcoef      ;point to first coefficient
        lar  ar3,#TIME       ;point to note length

        splk #022h,IMR      ; Mask interrupts
        clrc INTM           ; Enable interrupts

```

Figure 63: Interrupt Settings and Processor Initialisation

- 'LAR AR1,#DO' instruction loads the address of 'DO' to AR1. This points AR1 to initial sample value Y of musical note 'DO'.
- 'LAR AR2,#DOCOEF' instruction loads the address of the coefficient (A/2) for the musical note 'DO' to AR2.
- 'LAR AR3,#TIME' instruction points AR3 to location TIME which holds a constant for the duration of a single note.

Loading of these auxiliary registers are in preparation for playing the first note.

Transmit interrupt service routine starts with a delay subroutine and a number of note selection branches. Delay subroutine ensures that a note is played for about half a second. We shall examine that later.

```

=====;
;= TRANSMIT INTERRUPT SERVICE ROUTINE =;
=====;
TX
  call  DELAY          ;time a note is played for
  sub   #1             ; end?
  bcnd  nochange,NEQ   ;if at end reload timer counter
  lacc  #1000h         ;reload timer counter
  mar   *,ar3
  sacl  *              ;store time for decrement

  lamm  ar2            ; check where we are
  sub   #doccoef,0    ; is it DO
  bcnd  setre,EQ      ; if DO set to RE

  lamm  ar2            ;is it RE
  sub   #recoef,0     ;is it RE
  bcnd  setmi,EQ      ;if RE set MI
  :
  :
  :
setdo
  lar   ar1,#do        ; if none of the above
  lar   ar2,#doccoef   ; set DO
  b     newfreq        ; start again

setre
  lar   ar1,#re        ; point to RE in note table
  lar   ar2,#recoef    ; point to RE coefficient in note table
  b     newfreq

setmi
  lar   ar1,#mi        ; point aux regs to MI
  lar   ar2,#micoef
  b     newfreq
  :
  :
  :

```

Figure 64: Note selection in transmit ISR

- 'SUB #1' and subsequent 'BCND nochange, NEQ' instructions check the parameter returned by DELAY subroutine to check whether a note change is necessary. If it is not, program branches to a location 'NOCHANGE' and previous

note is played. If time is up, the following instructions will ensure that a new note will be pointed to.

- 'LACC #1000h', 'MAR *,AR3' and 'SACL *' instructions ensure that the location TIME is loaded with a fresh count ready for next note. Remember that AR3 points to TIME. 'MAR instruction loads ARP with 3 and makes AR3 current.
- 'LAMB AR2' loads the accumulator with the contents of AR2. 'SUB #doccoef,0' compares it with the address of the note DO's coefficient. If AR2 is found to be pointing to this, the program branches to a location 'setre'. If AR2 is not pointing to this note's coefficients, following group of instructions check which note's coefficient it is pointing to. These instructions effectively implement the equivalent of the 'CASE' statement in C.
- Let us examine one group of 'setxx' instructions as the rest achieve the same functionality. 'LAR AR1,#re' points AR1 to a location 're' in the musical note table. 'LAR AR2.#recof' points AR2 to the coefficient of note 're'. 'B NEWFREQ' implements an unconditional branch to a location 'NEWFREQ'.

The following section of the code start with locations 'NEWFREQ' and 'NOCHANGE'. This section is a sine wave generator. This is the same sine wave generator code as in the previous section with just two exceptions. The first is that AR1 is expected to point to the sample value 'Y'. The second is that AR2 is expected to point to coefficient 'A'. This was the reason for loading AR1 and AR2 with the appropriate values for the note 'DO' in the processor initialisation section of the program.

```

nochange
newfreq
;-----;
; note generation
; ar1 --> Y, ar2 --> coef
; ar1 moves from Y to Y1
;-----;
zap                ; Clear acc and product register
;-----;
; acc = -B*Y1
;-----;
mar *,ar1          ;make ar1 current
mar **,ar1         ;point to next value Y1
lacc *- ,15,ar2    ;Load acc with Y1 with shift of 15,ar1->Y
neg                ;Negate accumulator (-B)
;-----;
; treg0 = coef
; ar1 --> Y
; preg = coef*Y
;-----;
lt *,ar1           ;load treg0=coef, make ar1 current
mpy *              ;coeff*Y & place in preg, ar1 current
dmov *            ;Y->Y1 for next time round
apac              ;Add product register to acc
apac              ;all coeffs halved
sach *,1          ;remove sign bit
;-----;
lacc *             ;load sample to send
ldp #0            ;load data page zero
and #0FFFC        ;Clear lower two (status) bit from acc
samm DXR          ;Write lower acc to DXR
rete              ;Return to main program

```

Figure 65: Musical note generation

Receiver interrupts are not used and not enabled. So the program has just an 'RETE' for receiver interrupt service routine just in case.

The length that a note is played is decided by DELAY subroutine. This subroutine expects AR3 to point to a memory location which contains a number to be counted down. It returns a '1' in accumulator if the count has finished, and returns a '2' in the accumulator if count has not yet finished. To keep the complexity of the program to a minimum, DELAY was implemented as a counter. This makes the actual delay in time dependant on the number of instructions that are executed. Ideally, timer interrupts should be used. But DELAY is sufficient for our purposes.

```

;=====;
;=TONE LENGTH ROUTINE                                     =;
;=====;
; time that a note is played, decrements TIME constant
; IN: AR3 --> TIME
; OUT: finished acc=1, not yet acc=2
DELAY
    mar *,ar3      ; make ar3 current
    lacc *         ; load count value ->acc
    sub #1        ; decrement one
    sac1 *        ; store
    bcnd carryon,GT ; if not finished, carry on

    lacc #1       ; finished acc=1
    b finish

carryon
    lacc #2       ; acc=2
finish    ret     ; acc=1 ;return with parameter in acc
;=====;

```

Figure 66: DELAY subroutine

- 'MAR *,AR3' and 'LACC *' make AR3 current and load accumulator with the contents of the memory location that AR3 points to respectively.
- 'SUB #1' and 'SACL *' decrement the count and store it back in its previous location.
- 'BCND carry on, GT' branches if the count has not yet reached '0'
- If the count has reached '0', the branch falls through to 'LACC #1' which places a '1' in the accumulator.
- 'B finish' is an unconditional branch out of the subroutine with ACC=1
- 'LACC #2' instruction ensures that acc=2 if the count has not finished.

2.11.4 Running the program

You can run 'ocatve6' from the debugger or by directly loading it using DSK5L loader. It will play the notes in octave 6 sequentially. The program provides a good basis for improvements.

The first improvement could be to add the harmonics to make the notes sound richer. For this you will need to calculate the appropriate frequencies and the necessary coefficients. 'NOTES6.XLS' is a spreadsheet in MS-EXCEL which has all the

necessary formulas for coefficient calculations. Once these calculations are done, you can create a new table for harmonics. To generate the sine waves requires some care and thought. Add just one harmonic first. Then the same method can be extended to all the harmonics.

To add just one harmonic to the fundamental frequency, before the calculated fundamental sample is output, point AR1 and AR2 to the appropriate memory location and run the sine sample calculation again, add the previous and this result and output it. Do not forget to reset AR1 & 2 back to their original locations.

A sensible way forward after the first harmonic could be to make the sine calculation into a subroutine. In any case these changes should provide you with several rewarding hours.

2.12 A PC Controlled Tune Generator

We now know how to generate a sine wave and how to add functionality to the basic sine wave program to produce a tune generator. However in the current version of the musical note generator we can not change the notes unless we re-assemble the program.

In this section we shall develop a program which allows us to control the note being played from a PC. This necessitates writing two programs; One which runs on the DSK which receives information and generates the desired sine wave and one on the PC which downloads the program to the DSK and allows the user to send information to the DSK.

We shall first closely examine how DSK communicates with a PC via RS232 port. Building on this we shall develop our tune generator.

2.12.1 Background

The PC communicates with the DSK via the RS232 serial port. The DSK does not use a its own on-chip serial port for communication but simulates the RS232 serial transmission by using the BIO (Branch Control Input) for the RX line and the XF (External Flag) pin for the TX line. The DSK also uses the DTR (Data Terminal Ready) line to reset the DSK. The connections are as follows:

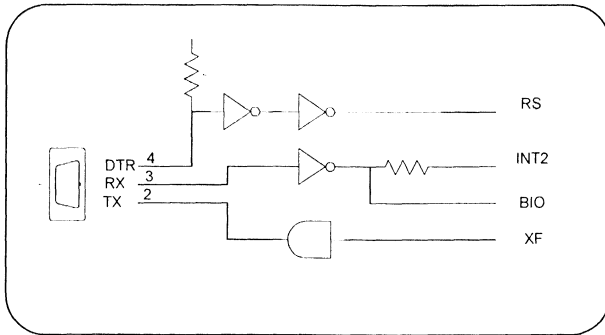


Figure 67: 'C5X DSK RS232 connections

It should be noted that the RX pin also connects to both BIO and INT2. This allows an RS232 to be interrupt driven.

Kernel Loading

The DSK is reset by driving DTR low which in turn drives RS low. After a short period of time the DTR line is raised to allow the 'C50 to start loading the communication kernel. Once the communication kernel is loaded it waits for the BIO (RX) pin to go low, indicating the first incoming start bit. On boot-up, the DSK does not know at what baud rate the PC is going to communicate. This problem is solved by counting the number of cycles that elapse over the length of this first start bit. Knowing this allows the DSK to calculate the Baud Rate. Once the DSK has received this first character it will send an ESC (27) character back to the PC at the correct baud rate to indicate that it has successfully synchronised with the PC. At this point the communication kernel is ready to receive commands.

Kernel Communication

The kernel accepts the following commands from the PC:

Mnemonic	Value Sent	Description
DD	00h	Dump Data
DP	01h	Dump Program
LD	02h	Load Data
LP	03h	Load Program
XCH_PGM	04h	Exchange Program
XG	05h	Execute/Go

The kernel echoes all data sent back to the PC. This allows software on the PC to verify if the data has been received correctly by the DSK. The writer of the PC software has the choice of re-sending an instruction or aborting. When the DSK transmits data it does not expect the PC to echo the data back to it. When the PC needs to transmit a word (2 bytes) the low byte is sent first followed by the high byte.

The PC needs to transmit words for addresses, data, and opcodes. The DSK will still echo back each byte it receives.

The procedure for receiving a word from the DSK is slightly more complicated. The PC needs to first send a sync byte (0) to the DSK to initiate the transfer. The DSK will then send the most significant byte and wait for another sync byte from the PC and will then send the low byte of the word.

DD - Dump Data

This command is used for receiving data from the DSK. The PC sends the command DD (00h) to the DSK followed by the starting address and the number of words to be retrieved. The DSK will then start transmitting the data on receiving a sync(0) character from the PC.

PC	DSK
DD	echo DD
Address low byte	echo Address low byte
Address high byte	echo Address low byte
Length low byte	echo Length low byte
Length High byte	echo Length high byte
null (initiate receive)	Data high byte (first word)
null	Data low byte
.....
.....
.....
.....
null (initiate receive)	Data high byte (last word)
null	Data low byte

DP - Dump Program

Similar to the last command except that it is used for receiving program data from the DSK. The PC sends the command DP (01h) to the DSK followed by the starting address and the number of words to be retrieved. The DSK will then start transmitting the data on receiving a sync(0) character from the PC.

PC	DSK
DP	echo DP
Address low byte	echo Address low byte
Address high byte	echo Address low byte
Length low byte	echo Length low byte
Length High byte	echo Length high byte
null (initiate receive)	Opcode high byte (first word)
null	Opcode low byte
.....
.....
null (initiate receive)	Opcode high byte (last word)
null	Opcode low byte

LD - Load Data

This command is used for loading data into the DSK. The command can also be used for initiating memory mapped registers. The PC sends the command LD (02h) to the DSK followed by the starting address and the number of words to be sent. The PC then transmits the words.

PC	DSK
LD	echo LD
Address low byte	echo Address low byte
Address high byte	echo Address low byte
Length low byte	echo Length low byte
Length High byte	echo Length high byte
Data low byte (first word)	echo low byte
Data high byte	echo high byte
.....
.....
.....
.....
Data low byte (last word)	echo low byte
Data high byte	echo high byte

LP - Load Program

This command is used for loading program code into the DSK. The PC sends the command LD (03h) to the DSK followed by the starting address and the number of words to be sent. The PC then transmits the words.

PC	DSK
LP	echo LP
Address low byte	echo Address low byte
Address high byte	echo Address low byte
Length low byte	echo Length low byte
Length High byte	echo Length high byte
opcode low byte (first word)	echo low byte
opcode high byte	echo high byte
.....
.....
.....
.....
opcode low byte (last word)	echo low byte
opcode high byte	echo high byte

XCH_PGM - Exchange Program

This command is only really of use if you are writing your own debugger software. The command allows the user to substitute a new program word at a specific address. The

original program opcode is returned. The primary use of this is to set break points in the software.

PC	DSK
XCH_PGM	echo XCH_PGM
Address low byte	echo Address low byte
Address high byte	echo Address low byte
opcode low byte	echo opcode low byte
opcode High byte	echo opcode high byte
null (initiate receive)	original opcode high byte
null	original opcode low byte

XG - Execute/Go

This command is used to initiate program execution on the DSK. The PC sends the command XG(05h) followed by the starting address. The PC then sends a sync(0) to start the DSK code running from the given address. Once the DSK is running, any further characters from the PC will cause the currently running program to halt.

PC	DSK
XG	echo XG
Address low byte	echo Address low byte
Address high byte	echo Address low byte
null (start DSK running)	

2.12.2 Using RS232 communications in assembly programs

The previous section described how to send and receive data using the communication kernel on the DSK. However the above only applies whilst the DSK is not running the users code. If you want to transmit and receive data within your program then you need some assembly code to handle the transmission and the reception. Transmission will be described first as this is simpler.

The method used to implement serial transmission is to place the required character in the accumulator. The character is then rotated through carry, one bit at a time and the external flag (XF) is set depending on the value of carry. Since XF is connected to TX line of RS232, after eight bit rotations the entire character will have been sent.

In entering the subroutine the carry flag is cleared, as this is used to determine if a one or zero should be transmitted to the RS232 port. The counter is now set up and defines how many times we should go around the inner loop of the subroutine to transmit the bits of a character. The section from **nextbit01** to **sbit01** is where we decide to transmit a one or zero. As we have already cleared the carry flag we will transmit a zero. This is the start bit which indicates the start of an RS232 character. At **sbit00** we cause the correct delay by repeating the instruction '**mar *, ar5**'.

We only actually need to execute this instruction once but repeatedly executing it does not present any problems and saves the use of a nop instruction. Once we have executed the delay the accumulator is rotated right through the carry flag. This means

that the value which was in the LSB of the accumulator is now in the Carry flag. Using this we can test the value of each bit in the character to be sent. Now that we have the value in the carry flag we branch back to **nextbit01** to test its' value. **banz nextbit01,*-** instruction causes a branch back unless the count on auxiliary register 5 (ar5) has reached zero. Once we have transmitted the 8 bits the loop exits and the two stop bits are sent by setting the pin (XF) high and delaying twice. The subroutine then returns.

```

TXBYTE:
        clrc c           ;start bit = 0
        lar ar5,#8      ;counter = 1 startbit + 8 data + 2 stop
nextbit01:
        bcnd sbit01,nc  ;if c=1 send 1 else send 0
        setc xf
        b          sbit00
sbit01:
        clrc xf
sbit00:
        rpt #BITLEN     ; delay for data bit
        mar *,ar5
        ror
        banz nextbit01,*-
        setc xf        ; send stop bits
        rpt #BITLEN
        nop
        rpt #BITLEN
        nop
        ret

```

Figure 68: Serial transmission routine

The critical part of the code is the part setting the delay between sending bits as this determines the baud rate. The following is a table of suitable values for delays.

Table 11: Delay values for serial communications

Baud Rate	BITLEN	BITLEN2	BITLEN (interrupts)	BITLEN2 (interrupts)
57600	345	169	310	155
19200	1039	518	920	460
9600	2078	1039	1860	960
4800	4156	2078	4156	2078

A few things are worth noting. The constant BITLEN2 is half the length of BITLEN and is used in the receive routine. The values of BITLEN assume that that this code is not interrupted by any other code. If your program is using interrupts such as reading and processing data from AIC, then you will need to use smaller delay values. Initial starting values are shown in the last two columns BITLEN (interrupts) and BITLEN2 (interrupts). The alternative is to mask any interrupts. However this may not be an acceptable option.

Receiving data is slightly more complicated. If a character is sent, then the DSK will immediately halt and enter the DSK communication kernel. The two options are to disable the interrupt that the DSK uses or install a new interrupt. Disabling the

interrupt is very simple. Most of the example code for the DSK will have code similar to the following:

```
SPLK    #022h,IMR        ; Using XINT
```

which enables INT2 for the kernel software and the serial port transmit interrupt. To disable the communication kernel change this line to:

```
SPLK    #020h,IMR        ; Using XINT
```

INT 2 has now been disabled. It is very important to realise that once this has been done it is no longer possible to use the debugger to control or debug your program. The following code can be used for receiving a character:

```

RXBYTE:      rpt    #BITLEN
             nop
wait01:      bcndd  start01,bio  ; wait for start bit
             lar   ar5,#7      ; delayed branch filled
             lacl  #0
             b     wait01
start01:     rpt    #BITLEN2    ; wait for half bit length
             nop
             mar  *,AR5
wbit01:      sfr
             rpt  #BITLEN
             nop
             bcnd zbit01,bio
             add  #80h        ; set bit in accumulator if bio(rx) is one
zbit01:     banz  wbit01,*-    ; loop 8 times
             ret
    
```

Figure 69: Serial reception routine

The code waits at *wait01* for the start bit. It should be remembered that other interrupt driven code such as data processing of AIC input will still be able to run. Once a start bit is detected the code waits for half a bits' width before checking for the next bit. This is to ensure that detection is in the middle of the bit to allow for the maximum possible baud rate discrepancy between the PC and the DSK. The code then uses the value BIO to determine if the next bit is one or zero. It then either places and shifts a one or a zero into the accumulator. This is repeated until the entire character has been shifted into the lower 8 bits of the accumulator.

The alternative to disabling the INT2 interrupt is to install a new interrupt service routine (ISR) for INT2. This can be done with the following code :

```

.int2:      .ps   0804h        ; Int 2 used for RS232
            B     NEWCHAR      ;
    
```

This code should be placed just before any existing ISR vector definitions. Don't forget to make sure that INT2 mask in the Interrupt Mask Register (IMR) is enabled:

```
SPLK    #022h,IMR        ; Using XINT
```


You now need to add code at the label NEWCHAR to actually receive the incoming character. It is important to remember that the INT2 interrupt has a higher priority than the AIC serial port interrupts. This may be an advantage or disadvantage depending on your application. If it is important that the AIC communication is not disrupted significantly then you should not use the INT2 interrupt. However, if to gain immediate control of the DSK is more important, then using INT2 maybe advantageous.

Other Considerations

One of the important considerations when using serial communication is the choice of baud rate. The program uses the RPT (Repeat) instruction followed by a NOP to produce a delay. The 'C50 pipeline treats a repeat followed by the instruction to be repeated as a single instruction regardless of how many times the instruction is repeated. The consequence of this is that the RPT instruction pair can not be interrupted by any interrupt until the repeats are finished. This delay may cause problems in servicing interrupts in some programs. One solution is to always use the highest baud rate possible to reduce all delays to a minimum level. The alternative is to use the RTB (Repeat Block) instruction instead of RPT instruction. However this instruction requires at least three instructions in the block to work properly.

2.12.3 The Programs

There are two programs in our tune generator, one which runs on the PC called `pc_tune.c` and one which runs on the DSK called `pc_tune.asm`. We will not be covering the details of `pc_tune.c` but the 'C' source is included and is quite easy to understand.

The program `pc_tune.asm` is based on the previous program `octave6.asm`. The major difference is that instead of playing the 7 notes of the octave in a predefined order it decides which note to play by information sent over the RS232 port. We will now look at the changes and additions that have been made to `octave6.asm` to produce `pc_tune.asm`.

The following figure shows the general flow of the assembly program. Sine generation routine is the same as in 'Octave 6'.

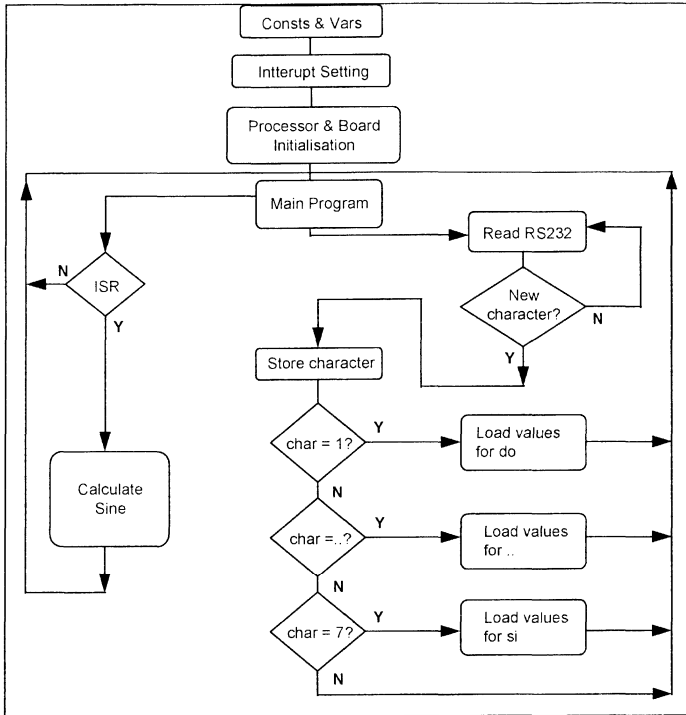


Figure 70: Flow of PC_TUNE assembly program

The first change is that pc_tune.asm includes references to two external files called com_init.asm and comm.asm. The contents of these files could have been placed directly in pc_tune.asm but it is good design practice to keep commonly used routines in separate files.

The first file is included in pc_tune.asm just after the AIC value definitions with the following line.

```
.include "com_init.asm"
```

com_init.asm contains the constants which determine what speed of RS232 communication we will be using, in this case 19200 BPS.

After the end of the note definition table the line

```
TEMP .word 0
```

has been added. This is used to provide a temporary storage space for any characters received from the RS232 port.

The next change is related to interrupt masking. In octave6.asm we have the line

```
splk #022h,IMR ; Using XINT syn TX & RX
```

In pc_tune.asm this is changed to:

```
splk #020h,IMR ; Using XINT syn TX & RX
```

The reason for this is to disable the kernel software on the DSK which communicates through INT2.

In the program octave6.asm the program reaches the main program label WAIT and constantly loops back to it whilst waiting for interrupts. In pc_tune.asm we use this loop to wait for an incoming character from the RS232 port. This is achieved with the following code:

```
;- Wait for a new character
   call  RXBYTE
;- and store it somewhere safe
   sac1  TEMP
```

The program branches off to the subroutine RXBYTE and waits for an incoming character. The operation of RXBYTE is covered in the previous section. Once a character has been received the subroutine returns control to the main program and stores the character at the location TEMP. The subroutine RXBYTE resides in the file comm.asm which is included at the end of pc_tune.asm with the line:

```
.include "comm.asm"
```

Once we have received a character from the RS232 port we need a method of acting on the data received. In our case we want to load a different set of note coefficients depending on the value sent. For example if we received the value '1', this would mean that we would have to load the coefficients for the note 'DO' if we received the value '2' we would have to load the values for the note 'RE' and so on. This can be achieved with a variety of methods but we use the following:

```
setdo   lacl  TEMP
        sub   #1,0
        bcnd  setre,NEQ
        lar   ar1,#do
        lar   ar2,#docoef ; set DO
        b     WAIT
```

This shows the code segment for checking if we should play the 'DO' note. The value of the character is loaded back into the accumulator. We should only play the note 'DO' when the received value is one. We test for this by subtracting one from the accumulator. If the result is zero then we load in the 'DO' coefficients into ar1 and ar2 and branch back to WAIT. If the value doesn't equal zero then we try the next note by branching to 'setre'. The code for 'setre' is

```
setre   lacl  TEMP
        sub   #2,0
        bcnd  setmi,NEQ
        lar   ar1,#re ; point to RE in note table
        lar   ar2,#recoef ; point to RE coefficient in note table
        b     WAIT
```

To select the other codes, the same code segment is used with different values.

The only other changes to the code are related to the TX interrupt service routine. We no longer need to use the DELAY function and therefore all references to it have been removed. Likewise, we no longer need to play a sequence of notes and therefore note selection code segments are removed. The remaining TX interrupt service routine is simply a sine wave generator.

2.12.4 Keyboard controlled notes

The program does not need to make use of the DSK5D debugger or DSK5L loader. To start the tune generator, type `pc_tune` at the DOS prompt. The program will automatically try communication port 1 and check for a DSK board. If you are not using communication port 1, you can select a different communication port using the C flag. For example, to select communication port 2, at the DOS prompt type:

```
pc_tune -c2 ↵
```

Once the program has found and reset the DSK it will try to download the program to the DSK. The program should show each line of DSK code as it is being downloaded. Once it has downloaded successfully, the PC will tell the program on the DSK to start running. You can now control the note being played within octave 6 by pressing the keys '1' to '7' on the PC's keyboard.

One simple modification idea could be to alter the program to echo back any data sent to it. Simply add the following line into the `pc_tune.asm` file :

```
;- Wait for a new character
    call  RXBYTE
;- and store it somewhere safe
    sac1  TEMP
    call  TXBYTE
```

Re-assemble the file `pc_tune.asm` and re-run the program `pc_tune`. This time you should see the data being sent back from the DSK thus demonstrating that two way communications can be implemented. This modification also demonstrates how assembly code modules can be reused in many programs.

3. Application Software

3.1 Introduction

This section examines typical DSP applications suited for the DSK. Some theoretical background is given which is followed by program details. Any special programming techniques employed are explained.

The application software examples build on the knowledge gained in the previous sections. These software examples further enhance your understanding of DSP and the DSK. They may also be useful in providing the templates for your own applications.

3.2 FIR Filter

3.2.1 Overview

Filters are used in signal processing to pass a group of frequencies and attenuate the rest. In this section we shall implement a digital low-pass filter. In the same directory (\DSKTOOLS\FIR) there are some more software examples that implement high-pass, bandpass and bandstop filters. You can experiment with these.

By connecting the AIC input to a sound source or a signal generator and the AIC output to a speaker and an oscilloscope, it should be possible to hear and see how the low pass filter algorithm removes higher frequencies. This particular digital filter attenuates all frequencies above 1KHz. If you input a music signal, the degradation in quality should be audible.

3.2.2 Files and Equipment

DIRECTORY	FILES	FUNCTION
\DSK\FIR	lopass.asm	source code file
\DSK\FIR	lopass.dsk	DSK output file

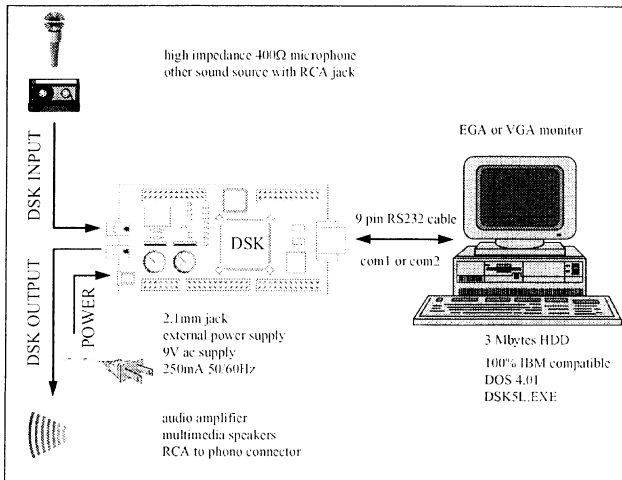


Figure 71: FIR Filter Equipment

3.2.3 Operation

Initiation

```
\DSK\FIR\    dsk5l lopass  c1 ↵ (com 1)
                                     c2 ↵ (com 2)
```

The effects of filtering on a signal can be observed on an oscilloscope. As the signal frequency moves above the filter cut-off frequency of 1kHz, the signal amplitude at the output of the filter should drop.

3.2.4 Background

Real signals combine a number of different frequencies. Filters allow certain frequencies to pass unaffected, while removing other frequencies. The spectrum of frequencies that are removed depends solely on the type of filter implemented. Filters are extremely important in signal processing where information at specific frequencies must be extracted or suppressed. For example, in audio applications the noise generated by 50Hz mains supply (mains hum) needs to be removed. This can be filtered out.

Primarily, there are four types of filter:

Low pass	allows only lower frequencies to pass
High pass	allows only higher frequencies to pass
Band pass	allows only a band of frequencies to pass, filtering out lower and higher frequencies outside the band.
Band stop	stops a band of frequencies, allowing the lower and the higher frequencies to pass outside the band.

The following figure shows an analogue high pass filter. It is easier to observe the effect on the signal in the frequency domain. The frequency at which the signal falls to half of its original amplitude is called 'the cut-off' frequency. This is also known as -3dB point ($20\log_{10}(1/2)$).

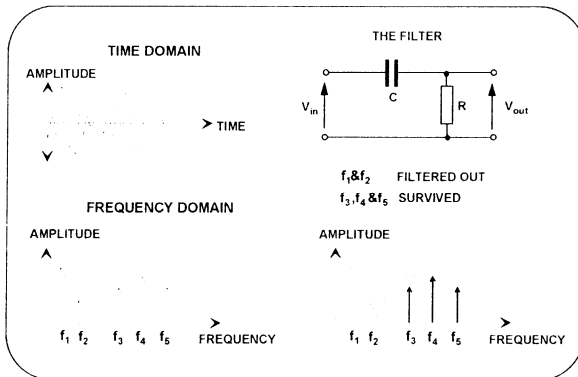


Figure 72: A high pass filter and its effect on a signal

Filters frequently affect another important property of the signal; it's phase. The following figure shows a sinusoid with a phase shift of 90 degrees. Phase is an important property of the signal. Most signal processing applications need to take into account phase response of various parts of the system. For filters it is desirable to have a linear phase response. A linear phase response means that all frequencies have the same amount of phase change introduced by the filter.

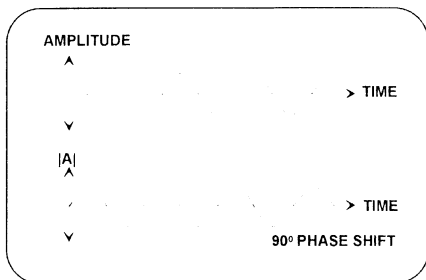


Figure 73: A sinusoid shifted by 90 degrees in phase

Filters can also be implemented digitally. Digital filters has distinct advantages over their analogue counterparts:

Predictability & Repeatability

In the digital domain there is no variance due to components and temperature as binary numbers are used to represent data. Every sample is always treated in the same way. There is no variance even if the signal is processed with different digital processors. Therefore, time and time again we can perform the same operation and achieve exactly the same characteristics.

Re-programmability

We can drastically alter gain and phase characteristics of a digital filter without needing to add or change any hardware. Digital filter characteristics may also be changed 'on-the-fly' in response to an input signal, allowing designers to build 'adaptive' systems.

Digital filters have traditionally been one of the most important of DSP applications. There are two types of digital filters:

- Finite Impulse Response (FIR),
- Infinite Impulse Response (IIR).

FIR filters respond to an impulse (a short signal) with a finite number of pulses. IIR filters respond to an impulse with an infinite number of pulses, hence the names. In practice, FIR filters are used where a linear phase response is desirable. IIR filters are used where the application requires a sharp cut-off with less processing. The subject of digital filtering is covered in a number of DSP text books¹. The first reference has a very simple illustration of how FIR filters work.

FIR filters are simpler to understand. The diagram below shows a simple three tap FIR filter, which involves three multiplications and two additions. Z^{-1} denotes a delay equivalent to a sample period. This is called a unit delay. In practice, the filter lengths

¹ A simple approach to DSP, C. Marven, G. Ewers, Texas Instruments, 1994.
Digital Signal Processing, A practical approach, E. C. Ifeachor, B. W. Jervis, Addison-Wesley 1993

are considerably longer than this, but the example will serve to illustrate the functionality of the filter.

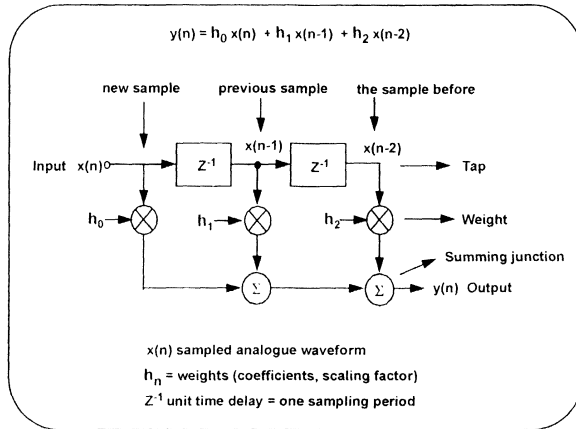


Figure 74: FIR Filter

- A new sample enters the filter from the left as $x(n)$ which is multiplied by coefficient h_0 .
- The previous sample, $x(n-1)$, is multiplied by h_1 , and the sample before that, $x(n-2)$, is multiplied by h_2 .
- All products are then added, yielding the final result which is:

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2).$$

- When the new sample arrives, all the previous samples are delayed and hence $x(n) \Rightarrow x(n-1)$
- When this routine is performed sufficient number of times with a digitized signal, unwanted frequencies are removed.
- The filter coefficients (weights) and the number of taps determine the characteristics of the filter.

FIR filters are popular as they can be implemented with ease and offer linear phase response. Linear phase response is essential in applications such as audio and data transmission.

Calculating the filter coefficients is a computationally intensive and repetitive task. This type of calculation is usually done using a software package. Filter design software takes the required response, roll-off rate, required number of taps and produces the filter weights and often simulates the response.

3.2.5 A FIR filter Implementation on DSK

The FIR filter we are going to implement is designed using a software filter design package. It is a relatively long filter with 80 taps. It has a cut-off frequency of 1kHz. The filter design package gives us the 80 coefficients. Let us now examine each section of the program that implements this filter.

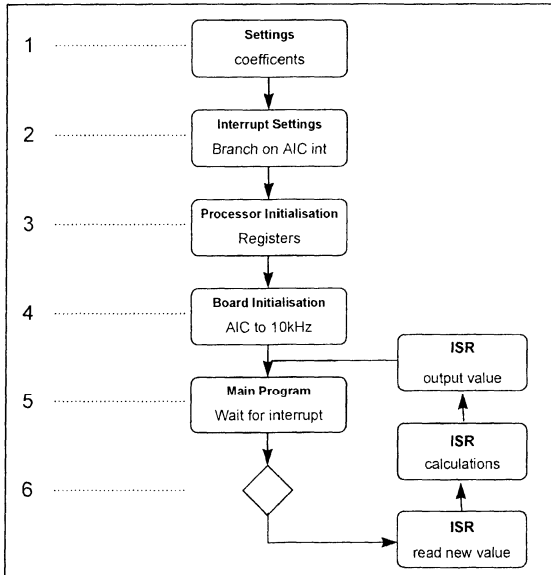


Figure 75: FIR Flow Chart

The FIR implementation is structurally similar to PASS.ASM. The coefficients are stored at the top of the source code.

```

        .mmregs          ; memory mapped registers
        .ds              0F00h ; data memory from address 0F00h
TA      .word           16 ; transmit A register value
RA      .word           16 ; receive A register value
TB      .word           31 ; transmit B register value
RB      .word           31 ; receive B register value
AIC_CTR .word           08h ; AIC control register value
OUTPUT  .word           0 ; temporary output storage
TEMP    .word           0 ; location of TEMPorary storage
TEMP1   .word           0 ;
seed    .word           07e6dh ; seed for random noise generator
;=====
;=Filter Coefficient Generator (fcut=1K) =
;=====
h0      .word           0 ; 40 0.0000
h1      .word          -157 ; 39 -0.0048
h2      .word          -261 ; 38 -0.0080
  
```

h3	.word	-268	;	37	-0.0082
h4	.word	-170	;	36	-0.0052
h5	.word	0	;	35	-0.0000
h6	.word	180	;	34	0.0055
.
.
h75	.word	-170	;	36	-0.0052
h76	.word	-268	;	37	-0.0082
h77	.word	-261	;	38	-0.0080
h78	.word	-157	;	39	-0.0048
h79	.word	0	;	40	0.0000
.
.
XN	.word	0,0,0,0,0,0,0,0,0,0	;	80	data locations
XN1	.word	0,0,0,0,0,0,0,0,0,0	;	stage delay line	
XN2	.word	0,0,0,0,0,0,0,0,0,0	;		
XN3	.word	0,0,0,0,0,0,0,0,0,0	;		
XN4	.word	0,0,0,0,0,0,0,0,0,0	;		
XN5	.word	0,0,0,0,0,0,0,0,0,0	;		
XN6	.word	0,0,0,0,0,0,0,0,0,0	;		
XN7	.word	0,0,0,0,0,0,0,0,0,0	;		
XNLAST	.word	0;			

Figure 76: FIR Coefficients

- The sampling rate is set at 10kHz by providing suitable values for TA, RA, TB and RB.

$$\text{SAMPLING FREQUENCY} = \frac{\text{MCLK}}{\text{TA} \times 2 \times \text{TB}}$$

By setting AIC_CTR to 8h, the transmit and receive sections of AIC are set to synchronous, the gain of the preamplifier is set to 1 and the bandpass filter is taken out.

- A temporary storage memory location is reserved, initialized and assigned the name OUTPUT. This location is used for storing the output sample from the filter.
- Storage areas TEMP, TEMP1 and a constant 'seed' are used by the random noise generator.
- The filter design package generates filter coefficients. These values are assembled into data memory, each with a label for user reference: i.e. h0 to h79. The coefficients are stored in Q15 format with their decimal value commented on the right of the semicolon. Notice that the coefficients are symmetrical around the middle.
- The history of input samples will be stored in memory. (x(n), x(n-1), x(n-2).....) The 80 locations are reserved and initialized. The last sample space is given the name XNLAST and the first sample space XN.

In the program following, sections until the interrupt service routines are similar to both PASS.ASM and SINE.ASM.

The filtering takes place in receive ISR. The interrupt service routine is executed every time a new analogue sample has been digitised by the AIC. After setting a sampling rate of 10,000Hz we can expect that the routine is serviced 10,000 times per second. The 80 coefficient filter equation is:

$$y(n) = h_0x(n) + h_1x(n-1) + h_2(x-2) \dots h_{78}(n-79) + h_{79}(n-79)$$

Receive ISR implements this equation.

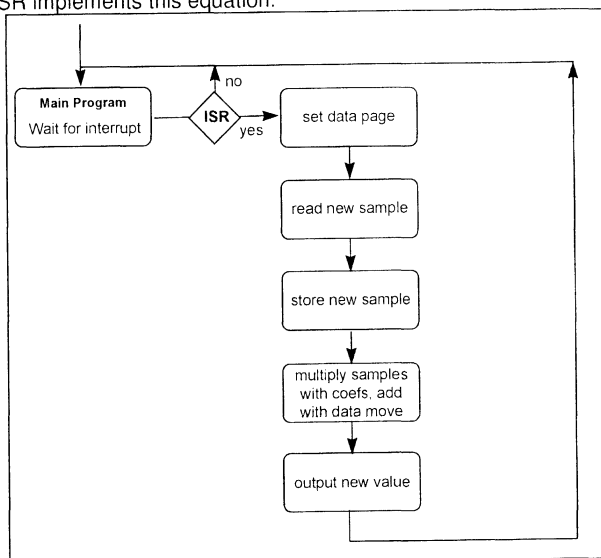


Figure 77: FIR ISR Flow Chart

3.3 FIR ISR

```

RX   ldp   #0           ; load data page zero
      clrc  INTM       ; enable interrupts for DSK debugging
      lamm  DRR        ; read serial input
      and  #0FFFCh    ; remove control bits
      ldp  #XN        ; load XN data page
      sacl XN         ; store received word to OUTPUT
      ;-----
      lar  AR0,#XNLAST ; AR0 = last delay element add
      zap  *,AR0      ; zero acc and product register
      mar  *,AR0      ; AR0 is the current AR register.
      ;-----
      rpt  #79        ; repeat next line 80 times
      macd #h0,*-    ; the complete coeff table.
      Apac          ; accumulate last product
      ;-----
      sach OUTPUT,1  ; remove extra sign bit
      lacc OUTPUT    ; load accumulator with output
      sfl          ; shift accumulator left
      and  #0FFFCh    ; two LSBs must be zero for AIC
      samm DXR       ; write acc to transmit register
      clrc  intm     ; enable interrupts
      ret            ; return to main program
  
```

Figure 78: FIR ISR

- 'LDP #0' sets the data page to zero to read the serial input port memory mapped register DRR.
- The most recent sample is read from DRR straight into the accumulator as a digital representation of the analogue waveform. This is our sample $x(n)$.
- '0FFFCh' is 'and'ed with the lower half of the accumulator to ensure that the two least significant bits are zeroed. This is necessary because the AIC has only 14 bits resolution.
- LDP changes the data page to the area of data memory that contains the history of samples.
- SACL stores the word to the location XN. This is the first sample location.
- The auxiliary register AR0 is loaded with the address of the data memory location which holds XNLAST. This will be used for indirect addressing during the multiply and accumulate phase.
- The accumulator and product register are cleared, removing $x(n)$ from the accumulator and the product generated from the previous operation. Performing this clearing is essential otherwise filter calculations will be wrong.
- Auxiliary register AR0 is then made active using the MAR command. This instruction changes ARP bits in ST0.
- RPT loads the repeat counter register with 79, indicating that the processor should repeat the following line a total of 80 times.
- MACD #h0,*- is the core of the filter.

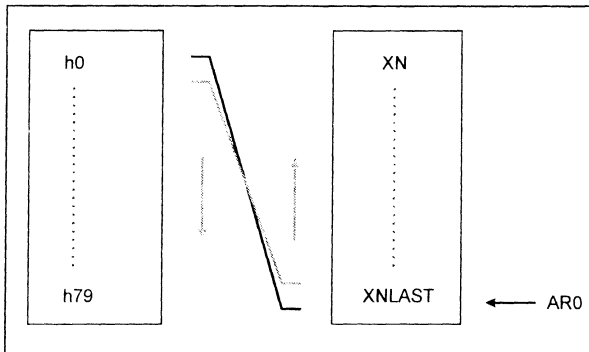


Figure 79: The Multiplication Process

The MACD instruction:

add the contents of the product register to the accumulator	<i>initially zero</i>
multiply the value pointed to by AR0 by hx	<i>initially AR0=XNLAST hx=h0</i>
copy location pointed to by AR0 to next location	<i>initially XNLAST into XNLAST+1 finally XN into XN+1</i>
decrement the address in AR0 by one	<i>as ^-</i>
increment the h0 address by 1	<i>as we are in a repeat loop</i>

All the samples are multiplied by their appropriate constant and added to the accumulator each time the instruction executes. The instruction also moves the samples by one memory location. This way the filter is ready for the next sample when it arrives which is the reason that AR0 points to the last sample rather than the first one. Such a manipulation would not be possible if the filter coefficients were not symmetric. If you are puzzled by the way the MACD instruction works, there is a program 'TESTMACD' in FIR directory which demonstrates the operation of the instruction on a simpler filter.

This instruction demonstrates the power of DSP filtering applications. An FIR filter is implemented effectively using a few instructions.

- The result of the successive multiply and accumulates is held within the accumulator, in Q30 format. The higher accumulator represents the integer part of the number which is stored with a left shift to a temporary location OUTPUT. Left shift eliminates the sign bit. The lower 16-bits are discarded. This has an effect on the accuracy of filtering but it is not significant.
- OUTPUT is reloaded to the lower accumulator.
- 'SFL' Left shifts accumulator by one bit normalizing the number out of Q15 format. Remember this operation is equivalent to dividing the number by 2^{15} .
- The two least significant bits are zeroed because the AIC is only 14 bits.
- The word is output to DXR register in the same way as in PASS.ASM. The filtered sample is then transmitted to the AIC.

Adding the random noise generator

The FIR filter as implemented will filter any signal that is fed into the input of AIC. LOPASS program also contains a random noise generator which is commented out. Comment these lines in by removing (;). There is one instruction just after the random noise generator; 'LAMB DRR'. This also needs to be commented out. Reassemble the program and run it. You will hear low pass filtered random noise. There are interesting experimental possibilities here. You could now comment out the filter and listen to the random noise only.

You can also change the characteristics of the random noise generator by using a different 'seed'.

In FIR directory, there are filter programs for a high-pass, a bandpass and a bandstop filter. These filters have 85 coefficients and all coefficients are contained in a separate file appropriately named as follows:

filter	coefficients file
high-pass	hipass.flr
bandpass	bandpass.flr
bandstop	bandstop.flr

These files are then included in the source file with an assembler directive:

```
.include "hipass.flr"
```

This directive causes the filter coefficients file to be included in the original source file before the actual assembly. This is good programming practice and keeps files tidy, modularised and separate. Examine the source files carefully. Is there any difference in the different filter source files?

3.4 Amplitude Modulator

3.4.1 Overview

This software modulates a signal from the serial port input onto a carrier wave. The output is sent directly to the serial output port which can be either heard or visualised.

3.4.2 Files and Equipment

DIRECTORY	FILES	FUNCTION
\DSK\AM_MOD\	am_mod.asm	source code file
\DSK\AM_MOD\	am_mod.dsk	DSK output file
\DSK\AM_MOD\	am_mod.ico	Windows icon

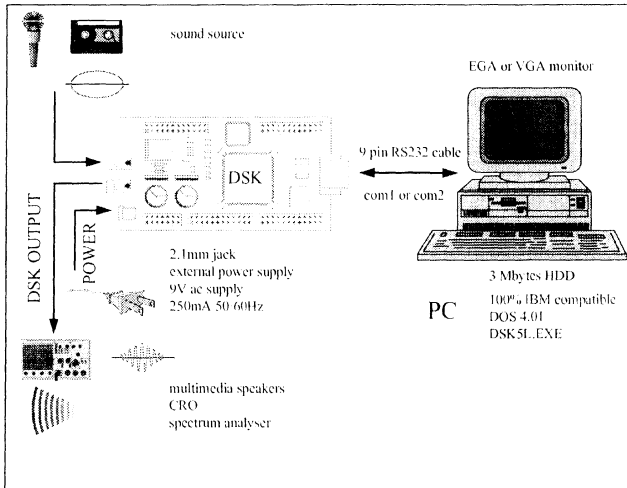


Figure 80: Amplitude Modulator Equipment

Initiation

```

\DSK \AM_MOD   dsk51 am_mod   c1 ↵ (com 1)
                                     c2 ↵ (com 2)
    
```

3.4.3 Operation

You can also run the program from the debugger. Apply an input to DSK and modulated signal can be observed at the output using an oscilloscope.

3.4.4 Background

Amplitude modulation techniques are used widely within radio applications where either voice or music is superimposed onto a carrier wave. Before the advent of frequency modulation, amplitude modulation was the standard technology.

In amplitude modulation the frequency of the carrier is kept constant. The amplitude however is varied in proportion to an input signal, thereby superimposing information onto the carrier wave. During demodulation, the carrier wave is removed, leaving the input signal.

Placing information on a carrier wave gives many advantages. The most significant is that we can have many different frequency carriers transmitting using audible frequencies with reduced interference. Hence we can have many radio stations broadcasting quality voice and music.

3.4.5 Theory

Amplitude modulation theory is very simple. The magnitude of the carrier sine wave should change proportionally with the information signal. The carrier usually is of higher frequency.

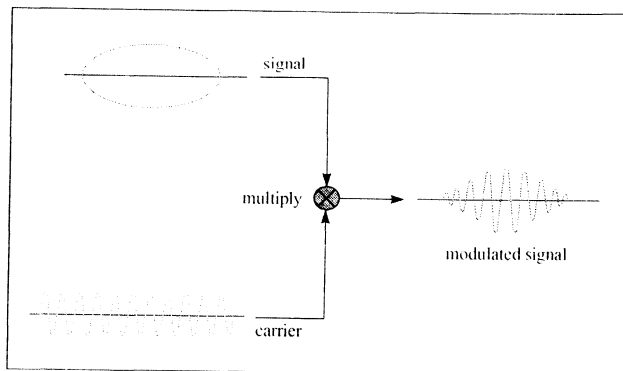


Figure 81: Operation of an AM

Modulating the carrier wave involves a multiplication as shown above. The result is a carrier with two side lobes; an upper and lower.

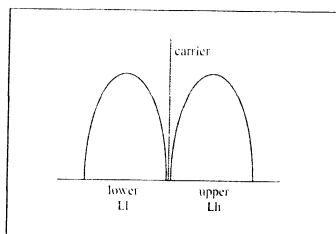


Figure 82: Frequency Spectrum

It is possible to see the modulation process using a spectrum analyser or a CRO and also to hear the carrier frequency and two side lobes.

$$F_L = F_C - B_W$$

$$F_H = F_C + B_W$$

3.4.6 Implementation

The implementation combines several concepts that we have already explored; sine wave generation, filtering and multiplication. The software serves to demonstrate that using a few simple building blocks we can start to build useful applications.

The carrier wave is generated using a digital oscillator which was discussed in the previous chapter. The frequency of the carrier is kept low to 2kHz. The input signal (modulating signal) is filtered using an FIR filter. This filter is similar to the FIR filter we implemented previously. However, for this application, coefficients are designed to give a bandpass filter between 200Hz and 1800Hz. The two signals are multiplied for every sample producing the modulated output. The modulated output signal can be observed on the output jack of DSK.

The following figure shows the general flow of the program. Let us now examine the program in more detail.

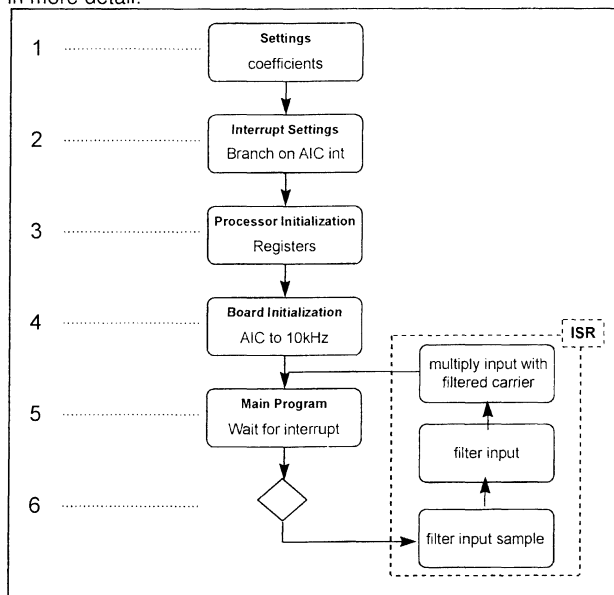


Figure 83: Amplitude Modulator Flow Chart

Settings consists of the data required to initialize the AIC, constants to generate a sine wave carrier and coefficients for a FIR bandpass filter. The filter ensures that the modulating signals are within the required range (200Hz - 1800Hz).

```

.mmregs          ; enable memory mapped registers
.ds              0f00h ; assemble to data memory
TA               .word 16 ; Transmit A register
RA               .word 16 ; Receive A register
    
```

```

TB      .word    31      ; Transmit B register
RB      .word    31      ; Receive B register
AIC_CTR .word    08h    ; AIC control register
coeff   .word    5219    ; coeff value
sinx    .word    15531   ; Y(n)
Yminus1 .word    02bfh   ; Y(n-1) value for carrier generation
;-----;
;Filter Coefficient Generator bandpass @ flcut=200hz fhcut= 1800hz
h0      .word    0      ; 40      0.0000
h1      .word    1      ; 39      0.0001
h2      .word    0      ; 38      0.0000
h3      .word    1      ; 37      0.0001
..      ..          ; ..      ..
h77     .word    0      ; 38      0.0000
h78     .word    1      ; 39      0.0001
h79     .word    0      ; 40      0.0000

XN      .word    0,0,0,0,0,0,0,0,0,0 ; 80 data locations for 80
XN1     .word    0,0,0,0,0,0,0,0,0,0 ; stage delay line
XN2     .word    0,0,0,0,0,0,0,0,0,0 ;
XN3     .word    0,0,0,0,0,0,0,0,0,0 ;
XN4     .word    0,0,0,0,0,0,0,0,0,0 ;
XN5     .word    0,0,0,0,0,0,0,0,0,0 ;
XN6     .word    0,0,0,0,0,0,0,0,0,0 ;
XN7     .word    0,0,0,0,0,0,0,0,0,0 ;
XNLAST  .word    0      ;
OUTPUT  .word    0

```

Figure 84: AM Filter coefficients

- A 10kHz sampling rate is set.
- Set serial port for synchronous operation.
- A 2kHz carrier is set using sinx, Yminus1 and coef f. All these values are in Q15 format.
- 80 tap FIR bandpass filter is set using Blackman windowing to generate 80 coefficients represented in Q15 format.
- 80 locations are reserved for the input samples.

The processor is initialized in the same way as in the previous programs. The main program is a 'WAIT' loop for interrupts.

All the necessary functions are implemented in the receive interrupt service routine. The ISR is executed on receiving an interrupt from the AIC. This is the input sample coming in. The routine performs the amplitude modulation in three stages. Firstly, a new sine wave output is calculated and stored in data memory. This represents the carrier. Secondly, an input is read from the serial input port and filtered by an 80 tap FIR bandpass filter with a pass band 200Hz to 1800Hz. Filtering removes unwanted frequencies which may be contained in the input signal. Both the filtering and sine wave generation routines are identical to FIR.ASM and SINE.ASM program's interrupt service routines respectively. Now let us look at the actual modulation function.

```

RECEIVE
; generate carrier          ; as with SINE.ASM
; filter input             ; as with FIR.ASM

;-----AM modulate bandpass on carrier
      ZAP          ; clear accumulator and product register
input  MAC   SINX,OUTPUT ; Multiply the carrier with the filtered
      APAC          ; Product -> ACC
      SACH  OUTPUT,1 ; Save and discard extra sign bit.
      LACC  OUTPUT,1 ; out of Q15 format
      AND   #0FFFCh ; Two LSBs must be zero for AIC
      SAMP  DXR      ; Send to transmit register
      RETE          ; return to main program

```

Figure 85: Amplitude Modulator ISR

- Firstly, clearing the accumulator and the product register prepares the CPU for the MAC operation.
- 'MAC SINX,OUTPUT' multiplies the magnitude of the carrier wave with the filtered result. Thus performing the modulation part of the program.
- Since accumulator was zeroed 'APAC' loads the modulated sample into the accumulator.
- 'SACH OUTPUT,1' stores the high accumulator into memory location with one left shift. Left shift discards the sign bit from the multiplication. Storing only high accumulator discards low accumulator. These are the least significant bits of the product.
- 'LACC OUTPUT,1' instruction normalizes the number out of Q15 format.
- 'AND #0FFFCh' clears the lowest two bits for AIC and the following instruction writes the sample to the transmit register.

There is some scope for experimentation with AM_MOD program. You could increase the carrier frequency to 3kHz. For this, the new coefficients need to be calculated. The formula for the coefficient is in previous chapter.

You also need some initial values for the sine wave. Use the same value for $\sin x$ and calculate a value for 'Yminus1' using the same formula. You should now have a program that modulates the incoming signal with a 3kHz carrier.

Another modification could be increasing the amplitude of the carrier. There is an instruction, the receive interrupt service routine, which reduces the carrier amplitude just after its generation. This needs to be changed.

You can observe the effects of these changes on an oscilloscope.

3.5 Reverberate

3.5.1 Overview

The reverberation software reads the serial input port and simulates echo by adding the delayed input signal to the new input signal. The echoed signal is output via the AIC to an amplified speaker.

3.5.2 Files and Equipment

DIRECTORY	FILES	FUNCTION
\\DSK\\REVERB	reverb.asm	source code file
\\DSK\\REVERB	reverb.dsk	DSK output file

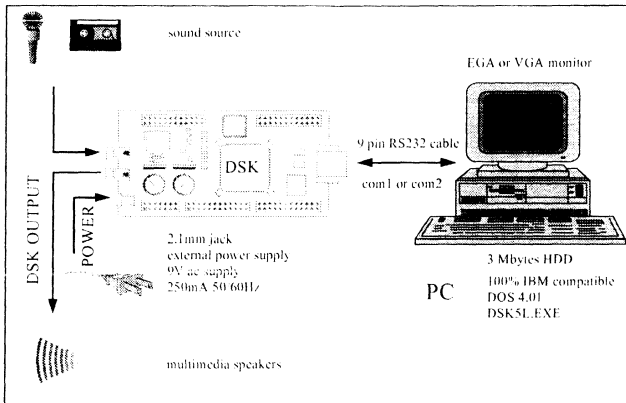


Figure 86: Reverberation Equipment

3.5.3 Operation

Initiation

\\DSK\\REVERB\\	dsk5l reverb	c1 ↵	(com 1)
		c2 ↵	(com 2)

The best way to observe the effect of the reverberation program is to apply an audio signal to DSK input and connect the DSK output to an amplified speaker.

With some microphone types, it may be difficult to experience the effect of reverb program. This is because the audio signal provided by the microphone is too small despite being amplified four times by the AIC. One solution is to use a microphone with higher output. Another solution is to build a pre-amplifier as suggested in Designer Note Pages Number 50 (DNP 50).

3.5.4 Background

Echo occurs when sound signal is reflected back to the listener. Echo in a small room can develop into reverberation which is a combination of a number of echoes. In a small room, as the signal bounces off the walls and other objects, echo gradually attenuates until it completely disappears.

It is possible to use the DSK for creation of reverberation effects. DSK stores sound samples of incoming signal and plays them back together with the new incoming signal at a slightly later time.

3.5.5 Theory

An echo can be heard if a person stands in a closed room and shouts out a short word and then listens. This is shown in the following Figure 87. Sound waves propagate from the speaker and bounce off the far wall (time 1) returning to the speaker a short time later. This gives the most intensive echo. The reflected signals continue and after several reflections they become too weak to be heard. In the example, three echoes are heard, equally spaced in time, each one significantly weaker than the last.

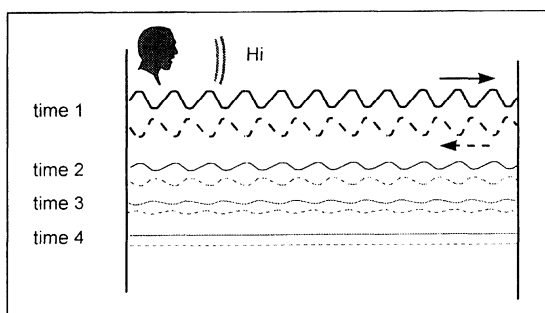


Figure 87: An Echo

Now, instead of shouting out a short word, consider the situation when the subject speaks out a five word sentence. What will the subject hear towards the end of the sentence?

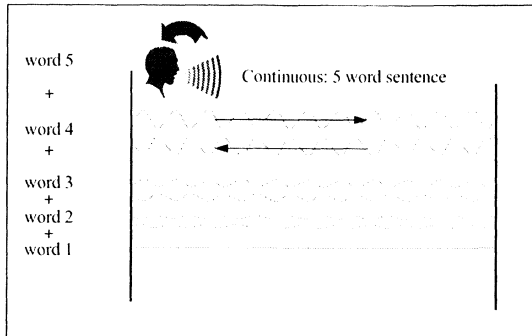


Figure 88: Continuous Echo

When speaking out the fourth word, the subject will hear a combination of word 4 together with echoes of previous words. The latest echo, echo of word 3 in this case, will be the strongest. (i.e. having the biggest amplitude). The weakest echo will originate from word 1.

3.5.6 Implementation

The reverberation effects described above can be created using the DSK. Echoes can be simulated by storing the sound samples for a defined time and then adding an attenuated version of these stored samples onto the incoming signal. A good reverberator needs to keep a history of samples. The older the sample, the more attenuated it should get. In DSP implementation, attenuation is achieved by multiplying the sample by a factor which is smaller than one.

The following figure shows a representative example of how a reverberator can be implemented on a DSP. In fact the diagram is strikingly similar to the diagram of an FIR filter.

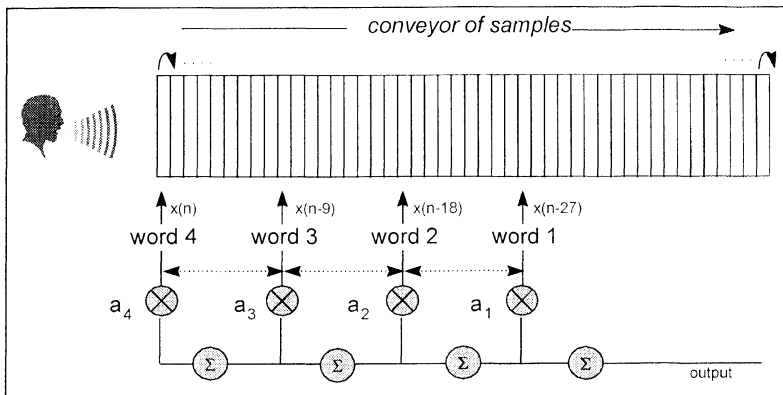


Figure 89: Echo Generator Implementation

DSP processors are ideal platforms for creating a variety of sound effects. Facilities for multiplication, addition and delay are very efficient. Additionally, most DSPs have circular buffers which make the implementation of delay easy and efficient.

The TMS320C50 has two circular buffers which work independently. A start and stop address defines the length of the buffer, while an auxiliary register points to a location in each buffer. When the pointer reaches the end of the buffer it automatically loops back to the start. This makes the implementation of delay easier. Furthermore, other auxiliary registers operating on the same buffer can be used to implement the taps.

Let us now implement a reverberator using a circular buffer. The following figure shows the overall flow of the program.

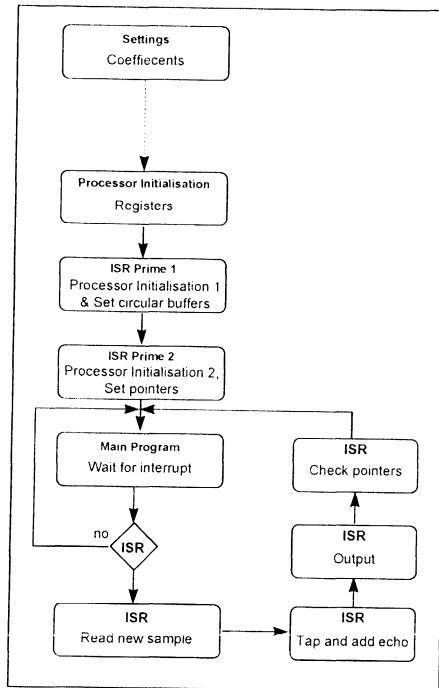


Figure 90: Reverberate Flow Chart

Let us now examine the reverberate program in detail. The settings section is similar to previous programs.

TA	.set	18	; Transmit A register value
RA	.set	18	; Receive A register value
TB	.set	36	; Transmit B register value
RB	.set	36	; Receive B register value
AIC_CTR	.set	29h	; AIC control register value
TAP	.set	500h	; Gap between echo

Figure 91: Reverberate Settings

- A sampling rate of 10kHz is set by loading TA, TB, RA and RB registers with the relevant values. The AIC_CTR sets synchronous operation for serial port, enables the band pass filter on the signal input and sets the input gain to 4.
- TAP represents the time gap between the echoes. The TAP sets number of memory locations which is proportional to the time between echoes.

Processor initialization is similar to previous programs except for the 'circular buffers' section. Let us now examine this in detail.

```

*****
* TMS32C05X INITIALIZATION
* *****
      .ps 0a00h
      .entry

START:
      SETC   INTM           ; Disable interrupts
      LDP    #0             ; Set data page pointer
      OPL    #0834h,PMST
      LACC   #0
      SAMP   CWSR          ; Set software wait state to 0
      SAMP   PDWSR        ;

;-----
; initialize circular buffers
      LACC   #0C00h        ; set up circular buffer pointers
      SACL   CBSR1        ; buffer1 = 0c00 -> 2bff
      SACL   CBSR2        ; buffer 2 = 0c00 -> 2bff
      LACC   #02bffh      ;
      SACL   CBER1        ; buffer top 2bff
      SACL   CBER2        ;
      LACC   #98h         ; buf1=AR0, buf2=AR1
      SACL   CBCR         ;

;-----
; initialize AIC
      SPLK   #022h,IMR    ; Using XINT syn TX & RX
      CALL   AICINIT      ; initialize AIC and enable interrupts

;-----
; initialize circular buffer pointers
      LAR    AR0,#2BFFh   ; AR0 --> top of CB1
      LAR    AR1,#1980h   ; AR1 -->1980h initial echo spacing
      LAMM   AR1
      SUB    #TAP,0
      SAMP   AR2          ;AR2--> 1980-TAP
      SUB    #TAP,0
      SAMP   AR3          ;AR3 -->1980-2xTAP
      SUB    #TAP,0
      SAMP   AR4          ;AR4 -->1980-3xTAP

;-----
;final processor initialization
      CLRC   OVM          ; OVM = 0
      SPM    0            ; PM = 0
      SPLK   #012h,IMR
      CLRC   INTM        ; enable
;-----

```

Figure 92: Reverberator Initialization

- Processor initialization is similar to previous programs.
- 'LACC #0C00h' starts a sub-section which initializes the circular buffers. Two circular buffers are created. Both reside in 0c00h to 2bffh. Effectively there is one circular buffer with two auto-pointers. Circular buffer one uses AR0 as a pointer; circular buffer two uses AR1.
- The value '0c00h' is stored to both circular buffer start registers.
- The following few instructions store the value '2bffh' to both circular buffer end registers.
- The accumulator is then loaded with the number '098h' which is loaded into the circular buffer control register. This register governs how the circular buffers

function. 098h assigns AR0 to circular buffer one and AR1 to circular buffer two and enables both buffers.

- The following two instructions initialize the AIC.
- Then a series of instructions set AR0 to AR4 to different positions in the circular buffer.
- The auxiliary registers are loaded with following values:

Register	Value
AR0	2bffh
AR1	1980h
AR2	1980h-500h
AR3	1980h-1000h
AR4	1980h-1500h

Let us now examine more closely the settings of auxiliary registers:

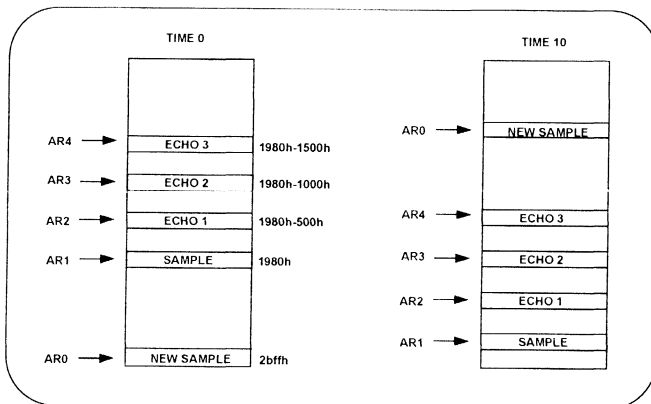


Figure 93: Circular Buffering

- The auxiliary register settings facilitate spacing the echoes. When a new sample arrives, it is written to a location in the circular buffer pointed to by AR0.
- AR0 then points to the next location for the next sample. But, because the buffer is circular, it will start from the beginning of the buffer (0c00h) if it has reached the top. Eventually, the oldest sample will be overwritten after AR0 has traversed the whole length of the buffer.
- AR1 points to the sample for which the echo will be generated. This sample is separated from the new sample by (2bffh-1980h) memory locations. AR2, AR3, AR4 point to previous samples which will be used to generate the other echoes. They are equally spaced by 500h (TAP) locations.

- Echo generation code needs to add a weighted portion of these samples to the sample pointed to by AR1.
- It is important to notice that the samples do NOT move. It is the pointers that traverse the circular buffer. New samples are written into the circular buffer as they arrive.
- Also note that AR0 and AR1 are maintained automatically by 'C50. This means that when AR0 and AR1 reach 2C00h, they are automatically reset to 0c00h. However for AR2, AR3 & AR4 this has performed manually.

All echo generation and buffer pointer maintenance takes place in the interrupt service routine. Let us now examine this in detail.

```

; RECEIVER INTERRUPT SERVICE ROUTINE
;
RECEIVE:
;-----
; core reverb routine
; stores received sample
; adds three echoes to delayed sample
; sample delay 2bfh-1980h
; echo delay 500h

        LACL   DRR           ; read data from DRR
        SACL   **+,0,AR1     ; store new sample
; and point to sample to be echoed
        LACC   **+,16,AR2    ; get&move sample to high acc
; start adding echoes
        ADD    **+,15,AR3    ; add (first echo/2) to sample
        ADD    **+,14,AR4    ; add (second echo/4) to sample
        ADD    **+,13,AR0    ; add (third echo/8) to sample
        AND    #0FFFCh,16    ; zero low two bits in high acc
        SACH   DXR          ; xmit
;-----
;pointer maintenance AR0 & AR1 maintained auto
; maintain AR2, AR3, AR4 manually
        LAMM   AR2           ;has AR2 reached top of buffer
        SUB    #2C00h,0
        BCND   CHK_3,NEQ
        LAR    AR2,#0C00h    ;set to bottom if it has
        B      GO
CHK_3:   LAMM   AR3           ; Is AR3 at top?
        SUB    #2C00h,0
        BCND   CHK_4,NEQ
        LAR    AR3,#0C00h    ; set to bottom if it has
        B      GO
CHK_4:   LAMM   AR4           ;is AR4 at top?
        SUB    #2C00h,0
        BCND   GO,NEQ
        LAR    AR4,#0C00h
GO:      RETE                ; go back program

```

Figure 94: Receive Source Code

- Note that the ARP initially points to AR0.
- 'LACL DRR' loads the fresh sample into accumulator.
- 'SACL **+, 0, AR1' stores the sample in the location AR0 is pointing to with no shift, increments AR0 and makes AR1 current (ARP=1). This completes the sample storage.

- 'LACC *+, 16, AR2' picks the sample to be processed. Notice that this is not the fresh sample but a previous sample away from the fresh sample by (2bffh-1980h) locations. This sample is moved to the high accumulator making the accumulator ready for echo addition. AR1 is incremented and AR2 is made current.
- 'ADD *+, 15, AR3' picks up a sample from the location AR2 is pointing to and shifts this sample by 15 bits. This is equivalent to halving the sample value. This value is then added to the value in the high accumulator.
- The following ADD instructions add smaller values of earlier samples to the sample value. The sample now has three echo values added to it with different weighting factors.
- The following two instructions 'AND and SACH' transmit the sample to AIC.
- The AR0 and AR1 are maintained automatically using the circular buffers. AR2, AR3 and AR4 must be checked manually to ensure that they do not go out of the range of circular buffer, and when they reach the top they loop back to the beginning. This is performed using a series of conditional branches.
- AR2 is checked first, loading the contents to the accumulator. Subtracting 2C00h represents the length of the buffer. If the value is less than or equal to zero, AR2 has not reached the end of the buffer and therefore the program continues to check AR3 and AR4.
- If the test finds that AR2 is at the top of the buffer, the start address is loaded into the register.
- Same tests are carried out for AR3 and AR4.

Now that you have gained sufficient experience with 'REVERB' program, a few interesting experiments could be performed. Firstly create a new file to experiment with and copy 'REVERB' into it. TAP constant could be changed to experiment with different size rooms. As TAP gets smaller, so does the size of the room. Another interesting experiment would be to change the echo weights. This is more complex but it will give you more programming experience. You need to change the shift factors in the core reverb instructions. Another more interesting and more complex change would be to use four echoes rather than three and add the echoes to the fresh sample as it arrives. This would eliminate the delay introduced to the fresh sample.

4. More Software Examples

4.1 Introduction

The application disk provided with this guide contains 6 more demonstration programs. Full source code listing files are included for self-study.

These programs demonstrate some of the more advanced capabilities of the DSK, while offering enjoyment. You should find on the application disk:

DIRECTORY	MAIN DSK FILENAME	FUNCTION
\DSK\MINUET\	minuet.dsk	outputs a tune
\DSK\RECORDER\	sampler.exe	short music recording program
\DSK\FUNCTION\	func.exe	noise and sine wave generator
\DSK\DTMF\	dialer.exe	a tone dialling PC interface
\DSK\SPEC\	spec50.exe	0-5kHz 256 radix 2 spectrum analyser
\DSK\OSCOPE\	oscope.exe	a virtual on-screen oscilloscope

After operating the programs, you can make modifications to them. Never modify the originals, in case you may need them again. Make a copy under a different name and then make your modifications.

4.2 Minuet Player

4.2.1 Overview

The DSK is used to play a minuet consisting of base and tune notes. The music repeats itself until the user stops the program.

4.2.2 Files

DIRECTORY	FILES	FUNCTION
\DSK\MINUET\	minuet.dsk	dsk file
	minuet.asm	dsk assembler source
	minuet.txt	tune tables
	sinsub.asm	sine wave generator

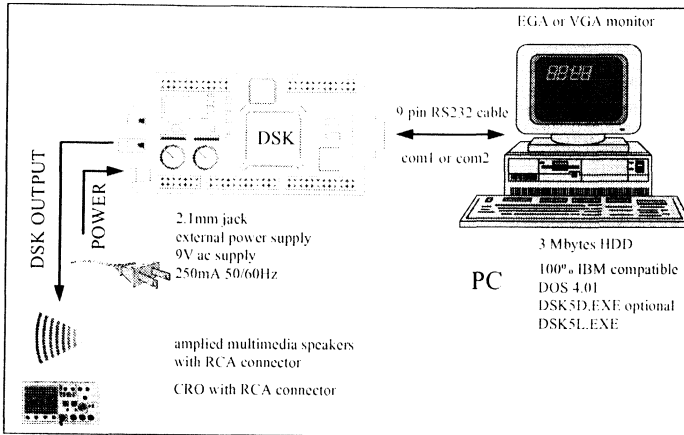


Figure 95: Music Player Equipment

4.2.3 Operation

Initiation

```

\DSK\MINUET\ dsk5l music -c1 ↵ (com 1)
                                     -c2 ↵ (com 2)
    
```

The music program generates a series of sine waves at base and tune frequencies with varying duration. Duration, base and the tune frequencies are contained in file called 'minuet.txt'. You can change the tune by changing the contents of this file. Try changing the duration values first. Once you are successful with these, you can proceed to change base and tune frequencies. The sine wave generation program is in the file called 'sinsub.asm'. This is used to calculate sine sample values for base and tune notes. 'Sinsub.asm' uses an equation to produce a sine wave output which is different from the one we used till now. You can run the minuet from within the debugger as well.

4.3 Sound Recorder

4.3.1 Overview

The application is a digital recording and playback facility. A simple graphical PC interface displays the record and playback status.

4.3.2 Files

DIRECTORY	FILES	FUNCTION
\DSK\RECORDER\	sampler.exe	host PC exe file
	sampler.dsk	dsk file
	sampler.asm	assembler source file
	sampler.ico	windows calling icon

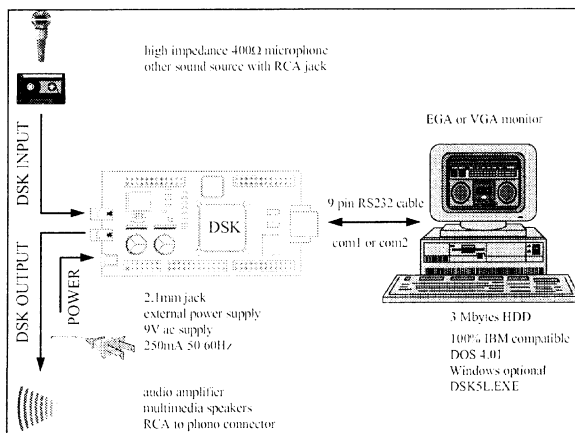


Figure 96: Sound Recorder Equipment

4.3.3 Operation

Initiation

\DSK\RECORDER\	sampler	-c1 ↵	(com 1)
		-c2 ↵	(com 2)

Command Keys

R	record
P	play

You can run the program from a DOS window under Windows 3.1, 3.11, 95 and NT. However, it will run slower. Use 'sampler.ico' file for icon definition when installing to operate under Windows.

'Sampler.exe' file loads the 'sampler.dsk' onto DSK ready for recording. Hitting 'R' key from the PC keyboard starts recording of the input of DSK. Just over one second's length of data is recorded. Hitting 'P' key starts the playback. The quality of the playback should be slightly degraded due to data packing but this may not be noticeable.

The record and playback program fills the DSK's data memory with information read from the AIC input port during record mode (R) and empties the memory locations to the AIC output port during playback mode (P). The host PC program controls the operation of the DSK by writing values to certain locations in DSK's memory. A simple PC interface informs the user when recording and playing have ceased.

The program packs two 8 bit words into one 16 bit memory location. Tightly packing in the data optimises the recording time but unfortunately means that the least significant 6 bits of the AIC's 14 bit word must be discarded. This reduces sound quality from 14 to 8 bits. In addition, the lower discarded bits represent lower volumes, and so quieter signals will not be detected.

With some microphones, it may be difficult to record the signal. This is because the audio signal provided by the microphone is too small despite being amplified four times by the AIC. One solution is to use a microphone with higher output. Another solution is to build a pre-amplifier as suggested in Designer Note Pages Number 50 (DNP 50). Ensure that the input is well grounded for high quality and low noise recording.

4.4 Host Driven Function Generator

4.4.1 Overview

The function generator program uses a host PC as an interface for displaying and modifying signals generated by the DSK. Both random noise and sine waves at different frequencies can be created.

4.4.2 Files

DIRECTORY	FILES	FUNCTION
\DSK\FUNCTION\	func.exe	host PC exe file
	func.dsk	dsk file
	func.asm	assembler source
	litt.chr	display fonts
	trip.chr	display fonts
	egavga.bgi	graphics drivers
	func.txt	operational guidance
	func.ico	windows calling icon

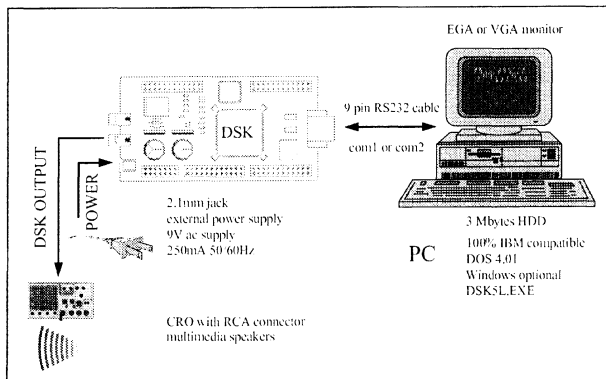


Figure 97: Function Generator Equipment

4.4.3 Operation

All the above files must be in the same directory. EXE program downloads the DSK file onto the 'C50 board, so you do not need to download the DSK file separately. If the mouse does not operate, use 'up, down' keys to move the frequency up and down by the amount 'DELTA=100Hz'. DELTA is adjusted by using the number keys:

Number	DELTA (Hz)
0	1
1	10
2	100
3	1000

Therefore frequency increments from 1Hz to 1000Hz in steps of (x10) are possible. Maximum operational frequency is 5,500Hz.

Initiation

```
\ DSK \FUNCTION\ func -c1 ↵ (com 1)
                        -c2 ↵ (com 2)
```

Command Keys

```
N                selects random noise
S                selects sine wave
↑↓              change sine wave frequency
```

'ESC' ends the program. You can run the program from a DOS window under Windows 3.1, 3.11, 95 and NT. However, it will run slower. The program can also be installed with an icon for windows operation.

Host PC communicates with DSK via its serial port. DSK program generates sine wave by reading certain constants from defined memory locations. The host program modifies these values directly which causes DSK to generate a sine wave at different frequencies. Examine the FUNC.ASM program. You will notice that there is no communication program with the PC Host. This is because, when the Host PC wants to communicate with DSK, it stops it and modifies the contents of certain memory locations affecting the changes requested by the user.

4.5 Host Driven DTMF Dialler

4.5.1 Overview

DSK TouchTone software uses a host PC to communicate dialled digits to the DSK. The DSK converts the digits into sound in compliance with the DTMF standard.

4.5.2 Files

DIRECTORY	FILES	FUNCTION
\DSK\DTMF\	dialer.exe	host PC exe file
	dtmf50.dsk	dsk file
	dtmf50.asm	assembler source
	phnumbrs.ato	display fonts
	dtmf.ico	windows calling icon

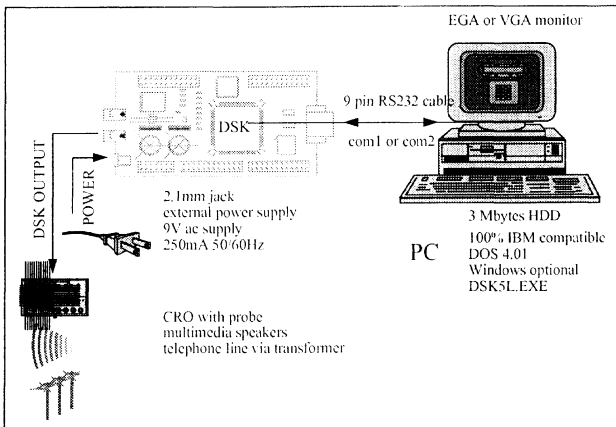


Figure 98: DTMF Equipment

4.5.3 Operation

Initiation

\DSK\DIALER\	dialer	-c1 ↵	(com 1)
		-c2 ↵	(com 2)

Command Keys

D	direct dialling
S	stored number dialling
↵ ↵	

The program operation is not so simple. A standard DTMF telephone keypad consists of a matrix where each row and column is assigned a unique frequency. Selecting a number is achieved by superimposing a column frequency with a row frequency, giving a distinctive tone.

When 'DIALER.EXE' is run, it loads 'DTMF50.DSK' to DSK. The host PC communicates the number to be dialled to the DSK. The DSP processor then calculates the two required sine waves and superimposes them. The DSK outputs the signal for a set period of time. This is repeated for all the digits, thus conveying the number. Telephone numbers can be dialled in two ways:

- by selecting a predefined number using the arrow keys
- dialling a number directly by typing D, the required number, followed by enter

Note: The phone log book can be altered using a ASCII text editor.

Note: "*" and "#" are valid keys for a distinct tone. ";" causes an insertion of 1 second pause.

4.6 Spectrum Analyser

4.6.1 Overview

The spectrum analyser transforms the DSK into a useful laboratory tool. A PC based display shows a graphical representation of the serial port input signal in the frequency domain.

4.6.2 Files

DIRECTORY	FILES	FUNCTION
\\DSK\\SPEC\\	spec50.exe	Host PC exe file
	spec50.c	C source for Host PC
	spec50.h	C header for Host PC
	spec50.asm	C5x Assembler source
	spec50.dsk	DSK executable
	spec50.txt	text explanation
	dsk_twid.asm	fft twiddle factor source
	egavga.bgi	graphics drivers

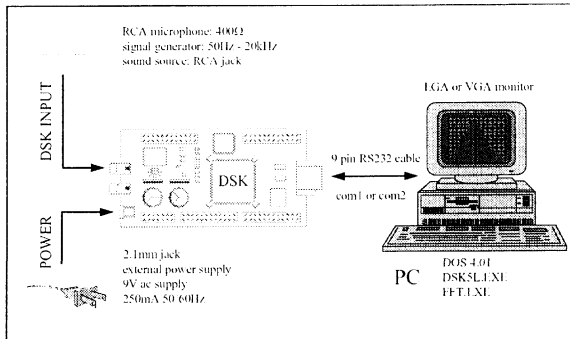


Figure 99: Spectrum Analyser Equipment

4.6.3 Operation

Initiation

\\DSK\\SPEC\\	spec50	-c1 ↵	(com 1)
		-c2 ↵	(com 2)

Command Keys

A	Averaging function 1,2,3,4
R	Reset DSK
Q	Return to MS-DOS

Before running the software, ground the DSK's serial in and out ports. Reconnect after initialization.

The DSK reads the input samples from the AIC into a buffer. When the buffer is full, it performs a 256 radix 2 FFT on the data, transforming it into the frequency domain. The DSK then signals to the PC that the data is ready. The real part of the FFT data is grabbed by the PC, loaded into memory and then displayed.

Pressing A performs an averaging feature, where the most recent data is averaged with previous values. Various averaging values can be selected: 1, 2, 4 & 8.

Red tops on the graphic display lag behind the signal, providing a display of previously viewed samples.

4.7 Oscilloscope

4.7.1 Overview

The oscilloscope software transforms the DSK and PC into a virtual oscilloscope. The DSK relays the serial port input signal's size and phase information to a host PC where it is displayed.

4.7.2 Files

DIRECTORY	FILES	FUNCTION
\\DSK\OSCOPE\	oscope.exe	Host PC exe file
	oscope.asm	C5x Assembler source
	oscope.dsk	DSK executable
	oscope.txt	Text explanation
	frame.dta	frame data
	oscope.ico	Windows icon
	egavga.bgi	graphics drivers

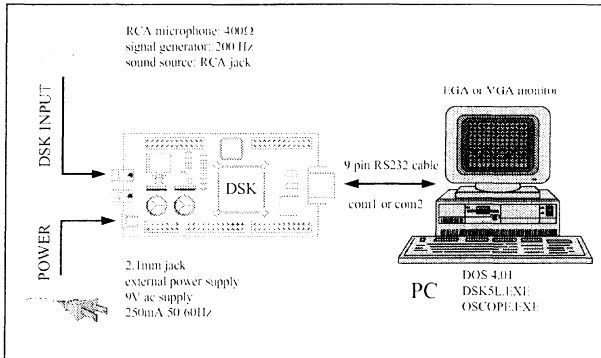


Figure 100: Oscilloscope Equipment

4.7.3 Operation

Initiation

\DSK\OSCOPE\	oscope	-c1 .J	(com 1)
		-c2 .J	(com 2)

Command Keys

ESC	exit - return to DOS
left - right arrow	increase - decrease samples
up - down arrow	increase - decrease volts per division
R	reset DSK
S	input how many samples
T	input trigger level
F	Frame-by-frame mode

Following commands operate only in Frame-by-Frame mode

space	process next frame
D	toggle display data mode
O	toggle SaveFrame to file

Entering trigger level disables PC interrupts which can create problems. It is best not to change this value. If the signal is unstable, use Frame-by-Frame mode. In this mode you can save sample values to a file and display them on the screen.

The DSK is responsible for reading the AIC and sending the information to the PC via the RS232 cable. The 14 bit resolution of the AIC is truncated to give an 8 bit word consisting of the 8 most significant bits of the AIC which are then transmitted to the PC.

The PC grabs the series of 8 bit words and stores them into a buffer. When the buffer is full, the PC looks for a triggering point which occurs when the signal crosses the x-axis and displays the signal accordingly.

5. Additional Information

5.1 Hints and Tips

5.1.1 DSK processor

- User program memory starts at memory location 0980h and is 8832 words long.
- Block B0 of memory can be configured to either program or data memory using the CNF bit. See the TMS320C5x USER'S GUIDE for details.

5.1.2 DSK software

- New versions of both the monitor debugger and assembly software are periodically released by Texas Instruments. Free upgrades can be found on TI's E-PIC BBS or via TI's Internet service.
- To quit the debugger program without terminating the program being run on your DSK type CTRL-BREAK
- View all the readme files on the TMS320 DSK tools disk.

<i>c:\DSK\DSK.DOC</i>	<i>the DSK User's Guide</i>
<i>c:\DSK\INSTALL.DOC</i>	<i>installation and technical reference</i>
<i>c:\DSK\README.1ST</i>	<i>read this file first - general information</i>
<i>c:\DSK\APPS\APPS.DOC</i>	<i>standard application description programs and how to use them</i>
<i>c:\DSK\ROM\ROM.DOC</i>	<i>creating a communications kernel code to program an EPROM</i>

- It is preferable to work using hexadecimal numbers when programming code. Hexadecimal is represented within the monitor/debugger program and therefore aids debugging.
- A list of current bugs for the TMS320C50 processor, assembler, debugger and DSK board are available on the BBS and Internet. Bug fixes are also usually included.
- In single step mode, the debugger is prone to change register values. This does not occur when executing to a specific address using the XA command.
- The latest release of the monitor debugger (version 1.02) performs serial port initialization of the TMS320C50.

5.1.3 DSK AIC

- The maximum sampling rate of the AIC is 19.2KHz.
- The AIC circuit needs to be initialized only once in between resets and power off. The TRY1.ASM program included with the DSK will run only when the AIC has been initialized. This can be achieved by first running a software program that contains the AIC initialization routine such as FUNC.DSK.
- Within the STARTER KIT'S USER GUIDE the input and output serial ports of the PCB have been unintentionally swapped. Please refer to the white silk-screen markings on the board for the correct input and outputs.

- The AIC's anti-aliasing filter on the A/D is a bandpass filter which can be bypassed within software. See PASS.ASM example in Section 2 for details. If a DC path is required, the anti-aliasing filter must be disabled and an external low-pass filter attached.
- The D/A low-pass filter is always active and cannot be disabled via software.
- The frequency roll-off of the bandpass filter and the output low-pass filter is determined by the switched capacitor filter clock frequency. This is set by software control, using typical characteristic values for the filters. For accuracy, these should be measured directly.
- The conversion frequencies for the A/D and D/A can be different. See Appendix of the DSK USER'S GUIDE for full details.
- The gain control which is implemented through software only is featured on the A/D side.
- A large selection table of AIC set-up configuration is available from the TI Internet or BBS E-PIC (DSK_PP26.EXE).
- PC to DSK communication baud rates can be varied to a maximum of 57,600 BPS.

5.2 AIC Settings

This report describes the steps required to initialize the AIC of the TMS320C50 DSK. The analogue interfacing circuit (AIC) is used to interface the analogue world to the DSK's internal world. Before this can occur, the AIC must be initialized and instructed how to perform. This takes place in four phases:

- Set the internal timer of the DSP to drive the master clock of the AIC
- Set the serial port of the DSP to receive and transmit to the AIC
- Place the AIC in a known state by performing a reset
- Load the AIC's internal registers with operational values.

A software based initialization routine is required to carry out the four phases as documented in Chapter 2. The four internal registers of the AIC are loaded with values that determine sampling rates, filtering characteristics and operational modes. The registers are as follows:

Register A	xx	TA reg	xx	RA reg	00
Register A' B'	x	TA' reg	x	RA' reg	01
Register B	x	TB reg	x	RB reg	10
AIC Control	xxxxxxxx		d7 d6 d5 d4 d3 d2	11	

The AIC registers work in conjunction with two counters A and B. These counters are loaded with the register values and are decremented every clock cycle. Once the counters are zero an interrupt is generated and the counters are reloaded from the registers.

Register A and B control the sampling and frequency characteristics of the AIC: T for the transmit and R for receive. The coder and decoder can therefore work at different frequencies. Registers A'B' hold values which are used to increase or retard the timing on the receive and transmit circuits. AIC control sets the functionality of the AIC:

- d2: deletes or inserts the input bandpass filter
- d3: disables or enables the loopback function (see DSK USER'S GUIDE page B-8 for details)
- d4: disable or enables the AUX IN+ and AUX IN- pins (see DSK USER'S GUIDE page B-8)
- d5: asynchronous/synchronous transmit receive sections (see DSK USER'S GUIDE page B-7)
- d6: gain on input
- d7: gain on input

Note that the gain is only on the input
 Note that the output filter is always on.

d6	d7	gain
0	0	1
0	1	2
1	0	4
1	1	1

5.2.1 Loading values

The AIC works by primary and secondary communications. In normal operation, only primary communications are used when reading and writing data to the AIC. When 14 bit data is sent to the AIC the two least significant bits are reserved for control. These two control bits tell the AIC to reload the counters from the registers in a particular way.

It is important to ensure that when sending data to the AIC the two least significant bits are set correctly (usually 00). If these bits are not controlled correctly, the AIC operation may be altered unintentionally.

Bits 0 1	load A Counter	load B Counter
00	TA : RA	TB : RB
01	TA+TA' : RA+RA'	TB : RB
10	TA-TA' : RA-RA'	TB : RB
11	TA : RA	TB : RB

* In the special case of 11 being sent, the AIC is told to expect a secondary transmission that is to change the registers values.

Bit 0 1	Register
00	A
01	AB'
10	B
11	AIC

The two least significant bits of the secondary transmission determine the register to be loaded.

5.2.2 TA, TB, RA, RB Values

$$\left. \begin{array}{l} TA \\ TB \end{array} \right\} \text{D/A conversion timing}$$

$$\left. \begin{array}{l} RA \\ RB \end{array} \right\} \text{A/D conversion timing}$$

Calculating the sampling rate means satisfying the following equations:

$$f_{MCLK} = \frac{1}{t_c \times (TDDR + 1) \times (PRD + 1)}$$

$$f_{SCF} = \frac{f_{MCLK}}{2 \times counter_A}$$

$$f_s = \frac{f_{SCF}}{counter_B} = \text{sampling rate}$$

f_{MCLK} : master clock frequency
 TDDR : DSP timing register
 PRD : DSP period register
 t_c : period of CLKOUT

f_{SCF} : switched capacitor frequency
 $counter_A$: TA or RA
 $counter_B$: TB or RB

T values and R values are identical for the same sampling rates.

MCLK frequency is usually set at the highest level: 10 MHz.
 The switch capacitor filter frequency should be set to 288,000 to give the typical filter values.

$$f_{BAND_LOWER} = \frac{f_{SCF} \times 300}{288000} \quad \text{3dB lower cut-off for bandpass input filter}$$

$$f_{BAND_UPPER} = \frac{f_{SCF} \times 3600}{288000} \quad \text{3dB upper cut-off for bandpass input filter}$$

$$f_{LOW_PASS} = \frac{f_{SCF} \times 3700}{288000} \quad \text{3dB cut-off for lowpass output filter}$$

Changing the value of the SCF to change the filter characteristics, this however will change the sampling rate.

The frequencies 300Hz, 3600Hz and 3700Hz are taken from the typical characteristics diagram of the bandpass and low-pass filter diagram in the data sheet. As these values are typical, the real values can only be measured.

Please consult appendix B within the DSK USER'S GUIDE for more information.

5.2.3 Typical Values

8kHz	
TA	17
RA	17
TB	37
RB	37

16kHz	
TA	9
RA	9
TB	36
RB	36

5.3 Questions and Answers

Why is the DSK only supplied with a 19.2kHz mono AIC?

Audio codecs that are supplied with low-end DSP development systems are not suitable for many general purpose applications that the TMS320C50 DSP processor can handle. The TMS320C50 is multi-functional and therefore the AIC was chosen to give the most flexibility and performance in the widest range of applications.

Audio AIC circuits are only suitable for high quality audio applications, whereas the AIC supplied with the DSK can be programmed for a wide range of sampling rates, incorporates filters and has 14 bit resolution.

Texas Instruments has designed an add-on board, which is available through distribution, to change the general purpose AIC for a 44kHz audio codec for specialised audio applications.

Why do some programs only run some of the time?

The DSK's AIC must be initialized after reset. If your program does not incorporate an AIC initialization routine the codec will not function. The routine need be executed only once, therefore running software that initializes the AIC before your program will solve the problem.

How do I set the AIC registers?

See chapter 2 and 5.

Is it possible to use an external clock with the TMS320C5x?

By connecting a x1 clock source to the CLKIN2 pin input and pulling CLK2MD pin low. The signal must be clean and cross-talk precautions taken.

The input to my DSK is not working

The DSK is designed to connect directly to an 8Ω microphone via an RCA jack. Sometimes, the output level of a microphone is not sufficient. It is suggested that an amplified microphone is used to ensure that the appropriate signal level is achieved.

5.4 Known Bugs

The following is a list of currently known bugs with current workarounds. For up-to-date information, watch Texas Instruments ftp and bbs sites

5.4.1 Handshake error

DSK after about 7 to 10 minutes of operation may produce a 'handshake error'. A red box will appear and no further communication will take place between the DSK and the Host. The only workaroud at the moment is to reset the DSK.

5.4.2 DTMF

When loaded with DIALER, after about 3 minutes of operation, tones produced becomes garbled. Reset DSK and re-run DIALER.

5.4.3 Function

5.4.3.1 Amplitude

When producing a 100Hz Sine wave, the output signal amplitude will vary if incremented in 10Hz steps. There is no workaround for this bug at the moment.

5.4.3.2 Random noise generation

If the multiplier buttons are selected when the random noise generator is operating, the screen will show a sine wave output. However the function generator will continue to produce noise.

5.4.4 Network Problems

If your PC is connected to the network, it may have network drivers resident in rhz lower portion of memory. DSK debugger and loader uses this section of memory and may overwrite these programs. DSK and application examples will operate correctly but when you exit the debugger, you may get the following message:

**general read failure on network drive
Abort, Retry, Fail?**

Come out of this message with an Abort. You will need to reset your PC to correct the problem.

Best solution to this problem is to disconnect from the network and do not load the network drivers when using DSK.

5.4.5 DSK Debugger loads

When in debugger, after about seven consecutive program loads, program counter of 'C50 may be set to a wrong point. This will give the impression that the program is not operating correctly. If you are doing consecutive program loads, before issuing XG command, check the program counter to see if it is set correctly.

6.1.3 BBS

E-PIC bulletin board service places a wide selection of DSK programs on a world-wide ftp server. By simply dialling the BBS number and logging-on via a modem you have access to a vast software resource that you can take freely and modify. Access to interesting and useful DSK programs allows you to see how others have tackled DSP design issues and provides a solid basis for application development. New releases of the DSK assembler and debugger are also available free of charge from the BBS.

Note:

The BBS covers not only DSK software but, in addition, there is a wide range of software for all of our fixed and floating point processors - also freely available.

The debugger can run COFF type files that are available from all TMS320C5x processors.

6.2 Internet

The number of subscribers to the world-wide Web is growing rapidly, providing on-line users with a vast database of information. The TI home page gives up-to-date information on all of our semiconductor products:

TI Home Page	http://www.ti.com
TI Semiconductors	http://www.ti.com/sc/docs/schome.htm
TI DSP Solutions	http://www.ti.com/sc/docs/dsps/dsphome.htm
DSP Hotline	http://www.ti.com/sc/docs/dsps/expsys.htm
DSP software co-operative	http://www.ti.com/sc/docs/dsps/softcoop/
DSP Designers Note Book Pages	http://www.ti.com/sc/docs/dsps/dnp/contents.htm
Training Workshop Dates	http://www.ti.com/sc/docs/training.htm
JTAG home page	http://www.ti.com/sc/docs/jtag/jtaghome.htm
TI microcontrollers	http://www.ti.com/sc/docs/micro/home.htm
TI&ME - custom information page	http://www.ti.com

The Texas Instruments TMS320 Third Party Program is the most extensive collection of Digital Signal Processing development support in the industry. Over 250 independent companies and consultants provide development boards, operating systems, software algorithms, function libraries and system consulting services. More information about Third Party support is available from either E-PIC or the TI web site.

6.2.1 TI&ME™ Internet Information Service.

TI&ME™ is your own custom web page on www.ti.com with direct hotlinks to new information from Texas Instruments on only those interests that you specify. You can request your own personalised weekly e-mail newsletter when anything new (within your specific interests) is added to www.ti.com. The e-mail contains the title and a brief abstract of the new information. This optional e-mail newsletter is an easy way to keep informed on new developments without spending your time searching.

Access to the TI on-line library of over 3000 semiconductor data sheets with an easy search front end. View these on-line in PDF or SGML or print a copy to your postscript printer. You are completely in control of the information flow: You select only what you are interested in.

- TI delivers only what you request.
- You may change your interest profile at any time.

6.3 TI Workshops

Texas Instruments' approved workshops give designers the opportunity to learn DSP programming principles from the experts. We understand the need for high quality training at an affordable price. We therefore offer a series of technical design workshops aimed specifically at DSP design engineers who wish to implement their designs more effectively.

Based on a structure of classroom learning and design laboratory sessions, the workshop participant quickly learns the major features of a processor and how to maximise its performance. Lessons learnt within the workshops can be used to improve your own designs.

The courses are targeted at both the experienced DSP programmer and the novice alike, covering the common issues that surround all DSP designs. Professional engineers, students, university lecturers and engineering project managers who want to improve their knowledge of DSP will find the courses useful. For companies who wish to train many engineers, TI offers an in-house training scheme that can prove more cost effective, timed for your convenience and tailored to your specific needs.

Workshops are available for all of our fixed and floating point DSP processors and also in several application areas such as digital motor control and active noise cancellation. The sessions are held in different locations within Europe and are taught in various languages. Texas Instruments publishes course dates on a quarterly basis. Please refer to regular mailings or E-PIC or the TI homepage for further details on fixed point and floating point workshops.

The TMS320C5x Workshop Agenda

<i>Development Tools:</i>	Software development process Using COFF tools Simulator
<i>Program Control:</i>	Branch instructions Subroutine calls Repeat instructions Block repeats
<i>Numerical Issues:</i>	Binary fractions IIR vs. FIR filters
<i>Algorithm Development:</i>	The algorithm
<i>Local Operations</i>	
<i>Multiprocessor Interfacing:</i>	TMS320C5x multiprocessing techniques Bit I/O interfacing Global memory interfacing

HOLD and HOLDA interfacing
DMA operation
Serial ports
TDM system operation

The course fee includes all study material, use of hardware and take-home literature.
(To ensure a place on a workshop, please book at least four weeks in advance).

6.4 Literature

6.4.1 The DSP Teaching Kit

The DTK or **DSP Teaching Kit** is a popular package that has already proved successful with universities, companies and individuals throughout Europe. Sold as a complete pack, the kit includes lecture foils, lecture notes along with a demonstration disk. An AC power supply and RS232 cable are supplied with the DTK - all the equipment needed to use the DTK as a teaching aid and to demonstrate software. The kit can be taught by anyone who has a basic knowledge of DSP as all the lecturing material is provided. In five lectures the fundamentals of DSP from sampling and filtering to more advanced topics such as bit reversed addressing in FFTs are covered.

For further information on the teaching kit please refer to E-PIC or distribution.

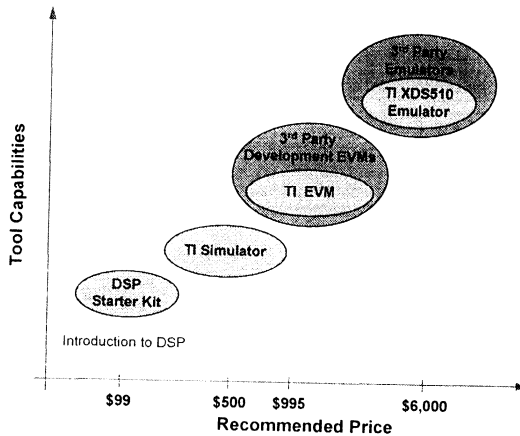
DSP Teaching Kit Part Number: TMDS320015X

6.4.2 TMS320C5x Related TI Books

TMS320C5x User's Guide	SPRU056B
TMS320C5x Fixed-Point DSP Product Bulletin	SPRT119A
TMS320C5x Power Dissipation Application Report	SPRA030
TMS320C5x DSP Seminar Workbook	SPRW017
TMS320C5x Data Sheet	SPRS030
Telecommunications Applications With 'C5x	SPRA033
TMS320WP010 Product Bulletin	SPRT124
TMS320WP010 Data Sheet	SPRS040
TC320IS54B Product Bulletin	SPRT114
TC320IS54B Chipset User's Guide (Preliminary)	SPRU128A
'C5x On-Chip Oscillator With External Resonator	SPRA054
'C5x application notes and DNP at: http://www.ti.com/sc/docs/psheets/apnote.htm	

6.5 Further Development

The DSK is the first step on the DSP development ladder. Texas Instruments provides a range of higher level development tools which provide additional features to the DSK.



6.5.1 Hardware

TMS320 Evaluation Modules (EVMs) are low-cost development boards used for device evaluation, benchmarking, and limited system debug. EVMs are available for the TMS320C16, 'C2x, 'C3x, and 'C5x. Common TMS320 EVM features include:

- Modification/display of memory and registers
- Assembler
- Software single-step and breakpoint capabilities
- On-board memory
- Host upload/download capabilities
- I/O capability

TMS320C5x Evaluation Module Part Number: TMDS3260050

The TMS320 Extended Development Systems (XDSs) are powerful, full-speed emulators used for system-level integration and debug. TI provides conventional in-circuit emulators as well as the world's first in-system scan-based emulators (XDS/22 and XDS510). The XDS510 emulator is currently available for TMS320C3x, 'C4x, 'C5x and 'C54x DSPs. Key features of the XDS510 include:

- Full-speed execution and device monitoring within your target system via a 14-pin target connector
- Global run/stop/breakpoint of parallel-processing DSPs
- High-level language debugging interface
- Software breakpoint/trace on all program and data addresses
- Single-step execution
- Windowed user interface similar to that of the 'C3x, 'C4x, or the 'C5x simulator

- Loading/inspecting/modification of all registers and memory
- Benchmarking of execution time of clock cycles

Full speed emulation and monitoring of the target system is performed serially via a 14-wire cable, which runs from the XDS510 to the target system.

TMS320C5x XDS Part Number: TMDS00510

6.5.2 Software

Speedy development and code maintenance over the life cycle of a product are concerns that all developers share. TI supports DSP developers with a family of optimising compilers for the TMS320 DSPs. These compilers translate ANSI-standard, C language files into highly efficient TMS320 assembly language source files, which are then input to a TMS320 assembler/linker.

TI offers C compilers that support the TMS320C2xx, 'C3x, 'C4x, 'C5x and 'C54x devices, and are complemented by the standard TMS320 programmer's interface for debugging C and assembly source code. The C compilers produce a rich set of debugging information, which allows source-level debugging in C. This enhances productivity and shortens the development cycle for both fixed-point and floating-point applications. Key features of the TMS320 C Compilers include:

- Highly efficient code, incorporating state-of-the-art generic and target-specific optimisations.
- ANSI standard runtime-support library
- ROM-able, relocatable, and re-entrant code
- The ability to link C programs with assembly language routines, allowing hand coding of time-critical functions in assembly language
- A C shell program that facilitates one-step translation from C source to executable code

TMS320C5x C Compiler Part Number: TMDS324L855-02

TMS320C5x Assembler/Linker Part Number: TMDS324L850-02

TMS320C5x Debugger Part Number: TMDS324L01

A TMS320 simulator is a software program that simulates the TMS320 DSPs for cost-effective code development and verification in non-real-time. Each simulator replicate allows the designer to verify operation and monitor the state of the TMS320. Simulators are available for the following devices: 'C2xx, 'C3x, 'C4x, 'C5x and 'C54x. Key features common to all TMS320 software simulators include:

- Execution of user-oriented DSP programs on a host computer
- Modification and inspection of registers
- Data and program memory modification and display:
 - Modification of an entire block at any time
 - Initialization of memory before a program is loaded
- Simulation of peripherals, caches, and pipelined timings
- Extraction of instruction cycle timing for device performance analysis

- Programmable breakpoints on:
 - Instruction acquisition
 - Memory reads and writes(data or program)
 - Data patterns on the data bus or the program bus
 - Error conditions
- Trace on:
 - Accumulator
 - Program counter
 - Auxiliary registers
- Single-stepping of instructions.

The simulators use TMS320 object code produced by the TMS320 macro assembler/linker or ANSI C compiler.

TMS320C5x Simulator Part Number: TMDS324L851-02

TI Sales Offices in Europe

BELGIUM

Avenue Jules Bordetlaan 11
1140 BRUXELLES/BRUSSELS
Tel : 32 (0)2 74 55 400 - Fax : 32 (0)2 74 55 410

FINLAND

Tekniikantie 12
02150 ESPOO
Tel : 358 (0)9 43 54 20 33 - Fax : 358 (0)9 46 73 23

FRANCE, MIDDLE-EAST & AFRICA

8-10 Avenue Morane-Saulnier - BP 67
78141 VELIZY-VILLACOUBLAY Cedex
Tel : 33 (0)1 30 70 10 01 - Fax : 33 (0)1 30 70 10 54

GERMANY

Haggertystraße 1
85356 FREISING
Tel : 49 (0)8161 800 - Fax : 49 (0)8161 80 45 16

Karl Wiechert Allee 74
30625 HANNOVER
Tel : 49 (0)5115 406 0 - Fax : 49 (0)5115 406 300

Maybachstraße 11
73760 OSTFILDERN (Stuttgart)
Tel : 49 (0)7113 40 30 - Fax : 49 (0)7113 40 32 57

HOLLAND

Buitenveldertselaan 3-5-7
1082 VA AMSTERDAM
Tel : 31 (0)20546 98 00 - Fax : 31 (0)20646 31 36

ITALY

Centro Direzionale Colleoni - Palazzo Perseo
Via Paracelso, 12
20041 AGRATE BRIANZA (MI)
Tel : 39 (0)39 684 21 - Fax : 39 (0)39 684 2912

REPUBLIC OF IRELAND

Europa House, Harcourt street
DUBLIN 2
Tel : +(353) 1 475 52 33 - Fax : +(353) 1 475 47 78

 **TEXAS
INSTRUMENTS**

DSP SUPPORT

Multilingual Technical Hotline

Deutsch	+49 - (0)8161-803311
English	+44 - (0)16 0466 3399
Français	+33 - (0)1 30 70 11 64
Italiano	+33 - (0)1 30 70 11 67
24 Hours FAXLINE	+33 - (0)1 30 70 10 32
E-Mail	epic@ti.com

Application Support (Via Modem)

BBS	+33 - (0)1 30 70 11 99
Internet	http://www.ti.com

SPAIN / PORTUGAL

C/ Gobelos, 43
Urbanización - La Florida
28023 MADRID
Tel : 34 (0)1 372 80 51 - Fax : 34 (0)1 372 80 87

SWEDEN

Halsinggegatan, 40 - Box 6706
11385 STOCKHOLM
Tel : 46 (0)8 587 555 00 - Fax : 46(0)8 587 555 90

UNITED KINGDOM

800 Pavilion Drive
Northampton Business Park
NORTHAMPTON NN4 7YL
Tel : 44 (0)1604 66 31 00 - Fax : 44 (0)1604 66 31 06

 **TEXAS
INSTRUMENTS**

Printed in France by Zimmerman
Duocom - October 1997

