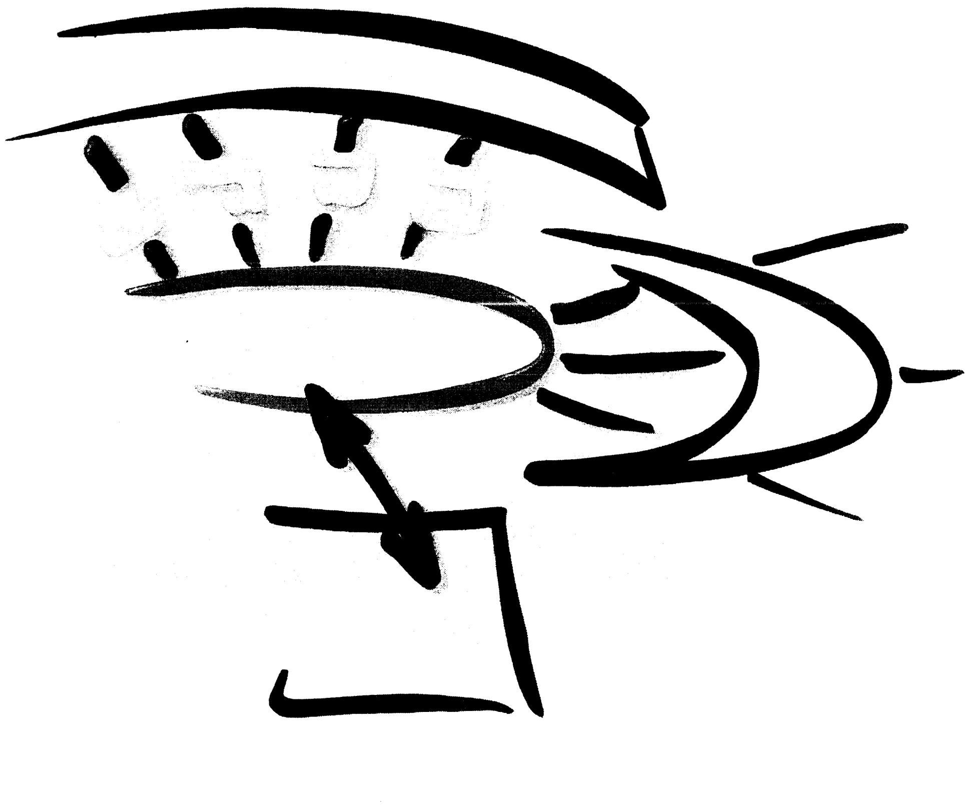
SPROC Signal Processor DATABOOK

STAR SEMICONDUCTOR



STAR—The Signal Processing Company

STAR Semiconductor Corp. Trademarks

SPROC, SPROCboard, SPROCbox, SPROCbuild, SPROCcells, SPROCdrive, SPROCfil, SPROClab, SPROClink, SPROCsim, SPROCview and "Sketch-and-Realize" are trademarks of STAR Semiconductor Corp.

Trademarks of Other Corporations

The following trademarks have been mentioned in this Data Book and are credited to their respective corporations.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation. OrCAD and DRAFT are registered trademarks of OrCAD Systems Corporation. VIEW*logic* and Workview are registered trademarks of VIEW*logic* Systems, Inc.

Disclaimer Notices and Copyrights

Information furnished by STAR Semiconductor Corp. is believed to be accurate and reliable. However, no responsibility is assumed by STAR Semiconductor Corp. for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent rights of STAR Semiconductor Corp.

Preliminary data are intended for guidance purposes in evaluating new products for equipment design. Such data are shown for types currently being designed for inclusion in our standard line of commercially available products. No obligations are assumed for notice of change of these devices. For current information on the status of preliminary data programs, please contact your local STAR Semiconductor Corp. sales office.

Life Support Policy

Except upon the prior express written approval of STAR Semiconductor Corp., no product shall be used as a component in any (i) life support device or system or (ii) any device intended for surgical implantation into the human body.

Printed in U.S.A. 12/91

DATA BOOK

TABLE OF CONTENTS

STAR
SEMICONDUCTOR

Section 1

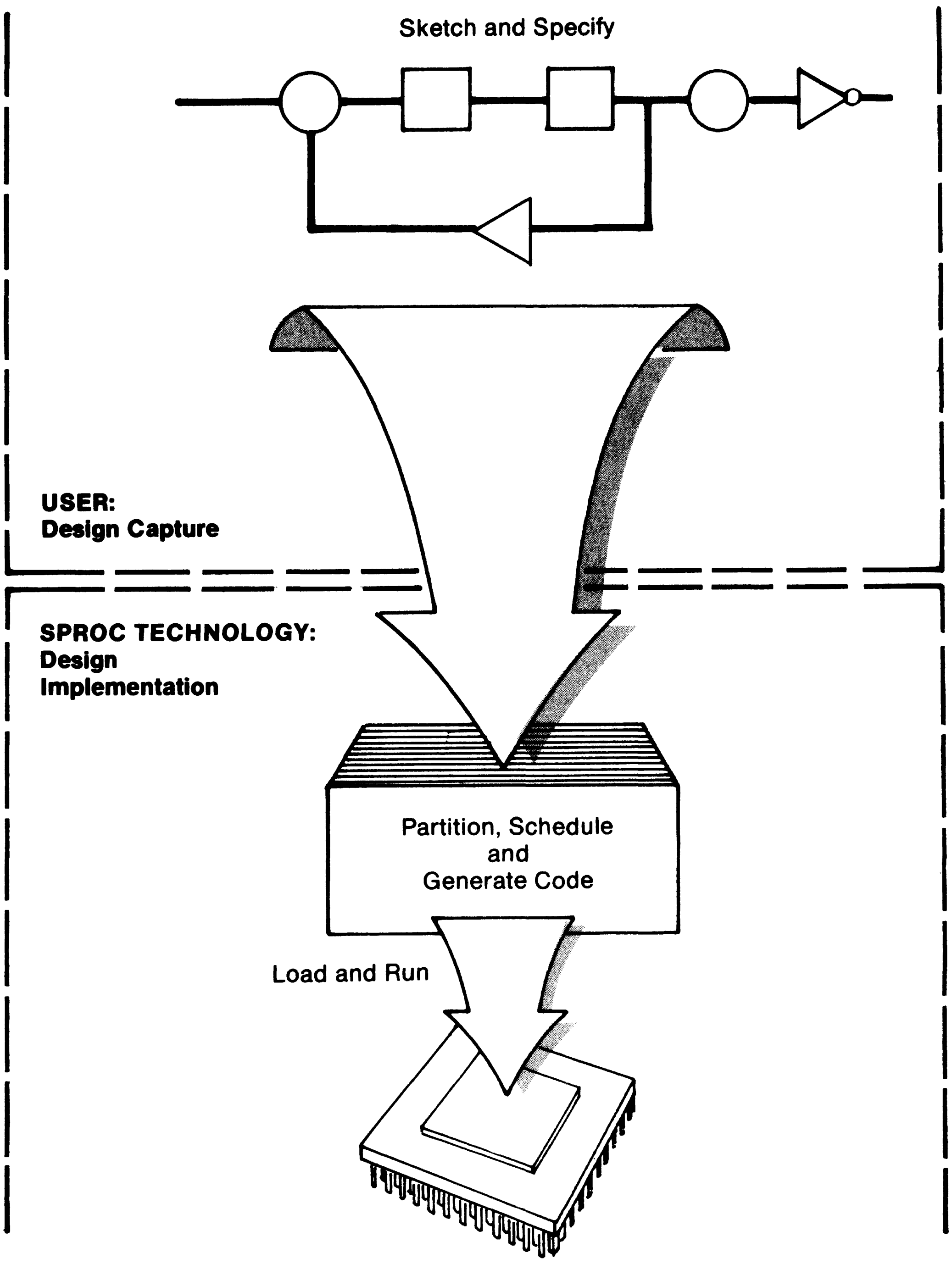
SPROC Signal Processing

STAR - The Signal Processing Company

STAR Semiconductor was founded in 1987 by an experienced team of signal processing experts and integrated circuit designers. Their mission was to develop and market easy to use, high performance programmable signal processors complete with support development system hardware and software. This objective has been achieved. STAR's integrated circuits and development systems are now expanding the application of signal processing by delivering long awaited breakthroughs in performance and ease of use to both analog and digital designers. The company's talent and resources are dedicated to continue making it easier than ever to design and implement high-performance, real-time signal processing systems.

STAR's SPROC-1000 series of programmable integrated circuits is the first to incorporate multiple signal processors on a single chip. The company's patented Central Memory Architecture (CMA), optimized for real-time signal processing applications, provides for concurrent processing of multiple data streams.

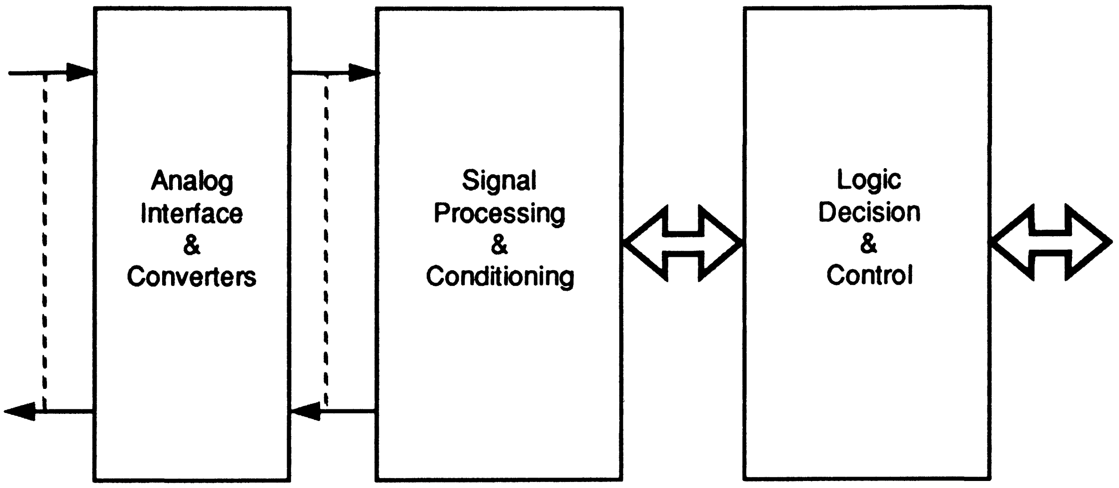
STAR's SPROClab development system delivers the first "Sketch and Realize" capability to designers of signal processing systems, eliminating manual coding and reducing time-to-market from months to days. By creating optimized systems directly from block diagrams, STAR's unique products let both novice and experienced DSP users improve the performance of their designs while dramatically reducing development time and cost.



SPROC "Sketch and Realize" Design Process Flow

SPROC Technology

Typical signal processing applications include tasks with data-dependent execution time and the ability to interface with other processors.


Analog Interface & Converters
Signal Processing & Conditioning
Logic Decision & Control

A Generic Signal Processing Based System

Mixing decision-oriented, data-dependent tasks with computationally intensive, highly deterministic tasks makes the development of a scheduler both difficult and inefficient. Separating the schedulers for these activities into signal processing and logic processing sections simplifies creating the application, results in a simpler, more flexible, higher performance design, and provides for a much more powerful development environment.

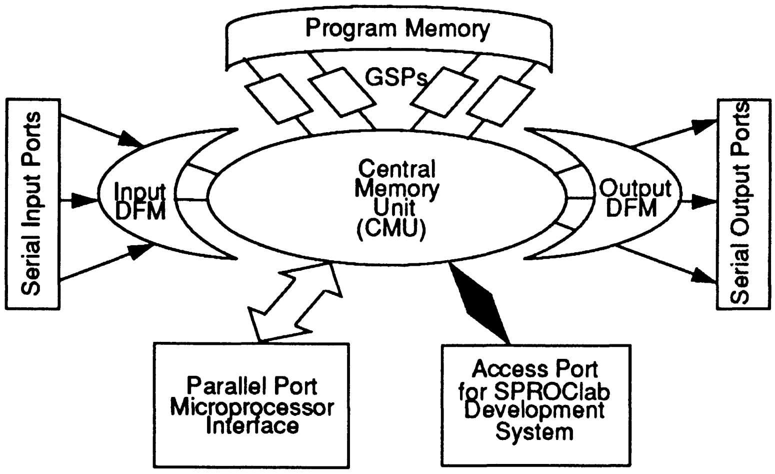
STAR Semiconductor's product line has been developed recognizing that real-time signal processing is very different from logic processing. Based on this insight, the company invented the SPROC signal processing architecture, which embodies a rethinking of both signal processing hardware and software application development tools. SPROC is the world's first general-purpose signal-processing solution optimized for real-time applications.

STAR's SPROC technology is fundamentally differentiated from first generation DSP technology by three factors:

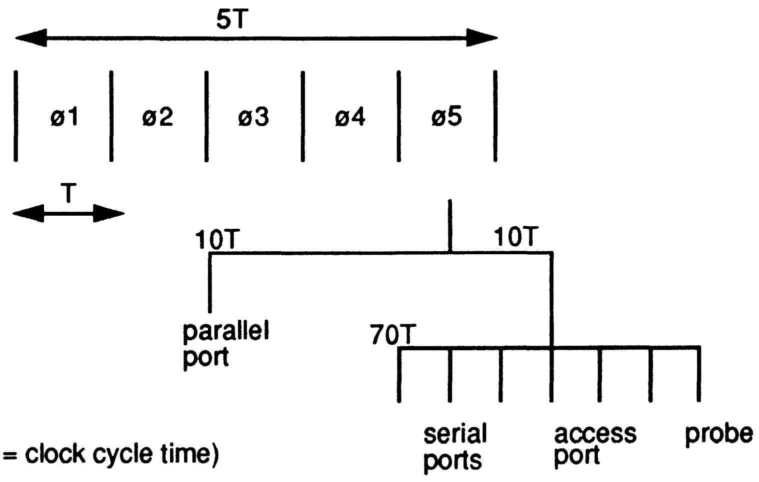
- Ground up rethinking of the processing and data-flow requirements of real-time signal processing
- Concurrent development of chip architectures and scheduler techniques designed to support high-level development and automated partitioning of complex problems to multiple SPROC chips
- System architecture development fully exploiting the integration capability and performance of modern semiconductor processing.

SPROC Architecture

The Central Memory Architecture (CMA) in STAR's SPROC signal processor product family includes the Central Memory Unit (CMU) and three principal logic elements: General Signal Processors (GSPs) for computation, input-output Data Flow Managers (DFMs) to coordinate simultaneous data streams, and parallel interfaces to external processors.


Program Memory
GSPs
Serial Input Ports
Input DFM
Central Memory Unit (CMU)
Output DFM
Serial Output Ports
Parallel Port Microprocessor Interface
Access Port for SPROClab Development System

The CMU at the center of the device is a multi-ported program and data space. Unlike first-generation DSP architectures, where interrupts are used to provide concurrent activity, the CMU implements concurrency directly via time multiplex of memory. Resource access is controlled by allocating time slots to each GSP and to the I/O unit.



CMU Access Allocation

One SPROC signal processor machine cycle is multiplexed into five CMU access slots. Each slot is equivalent to one master clock cycle.

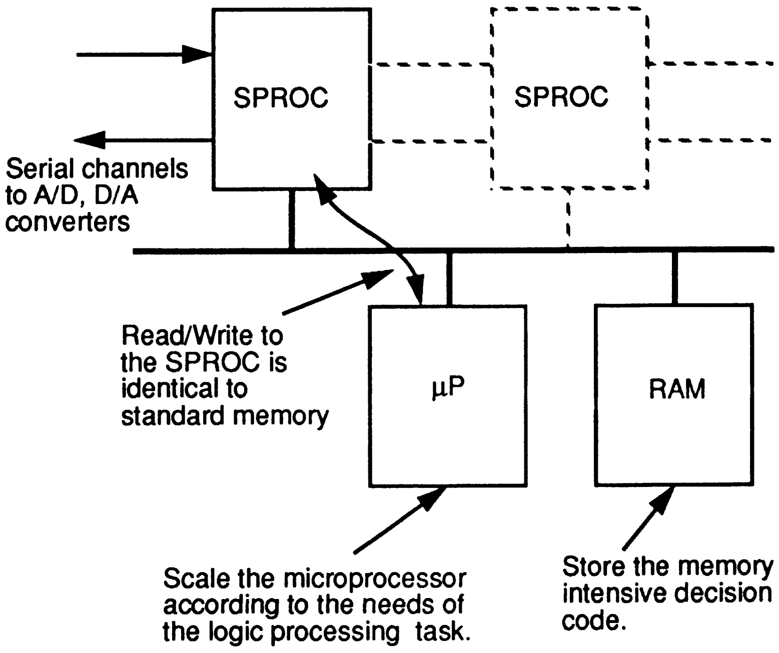
Slots ø1 through ø4 are used by the four GSPs. During these time slots the processor can either read from or write to the CMU.

Slot ø5 is the I/O slot, which is further divided. The parallel port is given this slot half of the time (10T), and thus can support signal or parametric data flow up to 1/10T words/sec without interrupting the processing on any of the GSPs. The other half of the time, the slot is further multiplexed into seven additional slots, each of which can read/write the CMU at 70T intervals. These seven additional time slots are used by the various serial ports, the access port, and a real-time probe.

The first implementation of the SPROC-1000 series contains four on-chip general signal processors optimized for real-time analog applications. The GSPs can be used either individually or collectively, depending on the applications's requirements. The CMU enables these 24-bit fixed-point processors to execute signal processing code concurrently and without interruption. This capability results in a major performance enhancement that improves with increasing sample rate.

The Data Flow Managers (DFMs) coordinate the filing of input and output data into or out of the CMU and handle I/O tasks with no impact on GSP performance. Because the CMU is a time division multiplexed (TDM) multi-ported memory, this activity can take place in parallel with the signal processing, enabling the device to service high sample rates with high efficiency. The DFMs set up either double buffers or vector FIFOs in the CMU. Direct hardware support of concurrency simplifies generation of code beyond what is possible with first-generation DSP architectures.

The parallel interface to other processors also optimizes the SPROC device for concurrent, real-time operations. Unlike first-generation DSP architectures, STAR's central memory architecture enables fast and straightforward communication with external logic processors. The architecture allows an external logic processor to read or write any part of the CMU completely asynchronously with any other ongoing activity. The SPROC device provides all the necessary interfacing signals to the supported logic processors, as well as any necessary word width adjustments required to interface with external logic processors.

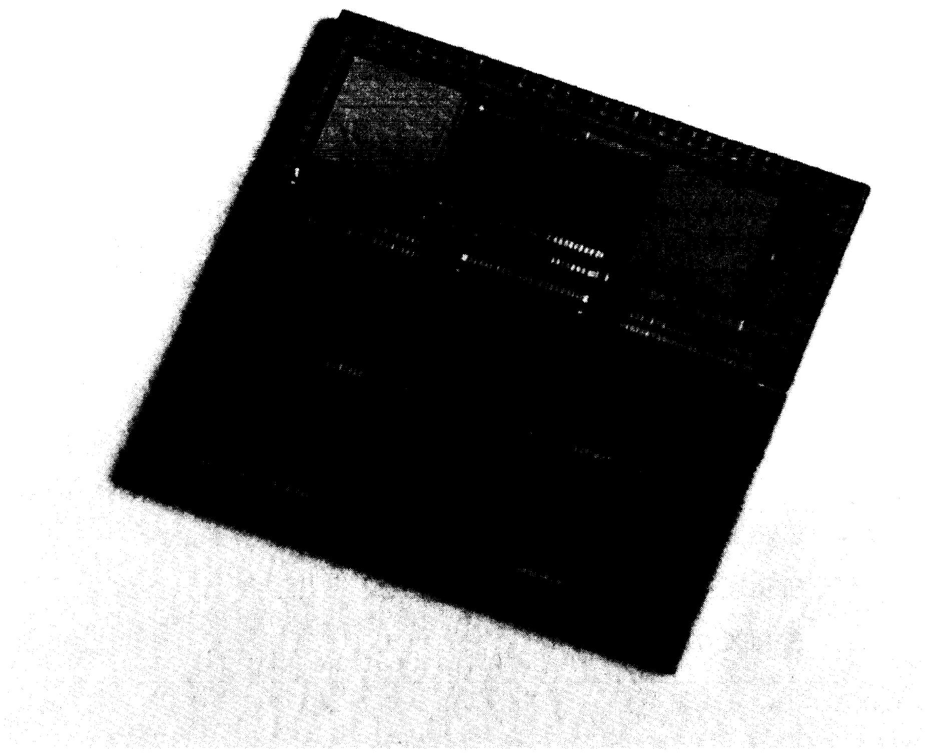

SPROC
SPROC
Serial channels to A/D, D/A converters
Read/Write to the SPROC is identical to standard memory
µP
RAM
Scale the microprocessor according to the needs of the logic processing task.
Store the memory intensive decision code.

These elements combine to provide an architecture with hardware support for concurrency and a memory system that permits easy and efficient coupling of the SPROC unit to other processor systems. The result is a technology that is optimized for a vast array of real-time applications easily programmable with high-level tools.

The SPROC-1000 Series of Programmable Signal Processors

The SPROC-1000 series of programmable, single-chip DSPs offers a variety of multiprocessor, memory, and interface configurations to suit applications with diverse function, performance, and cost requirements. Based on the unique central memory architecture, SPROC chips provide complete, integrated implementation of signal-processing subsystems.

Family	Number of On-chip General Signal Processors
SPROC-1400	4
SPROC-1200	2
SPROC-1100	1

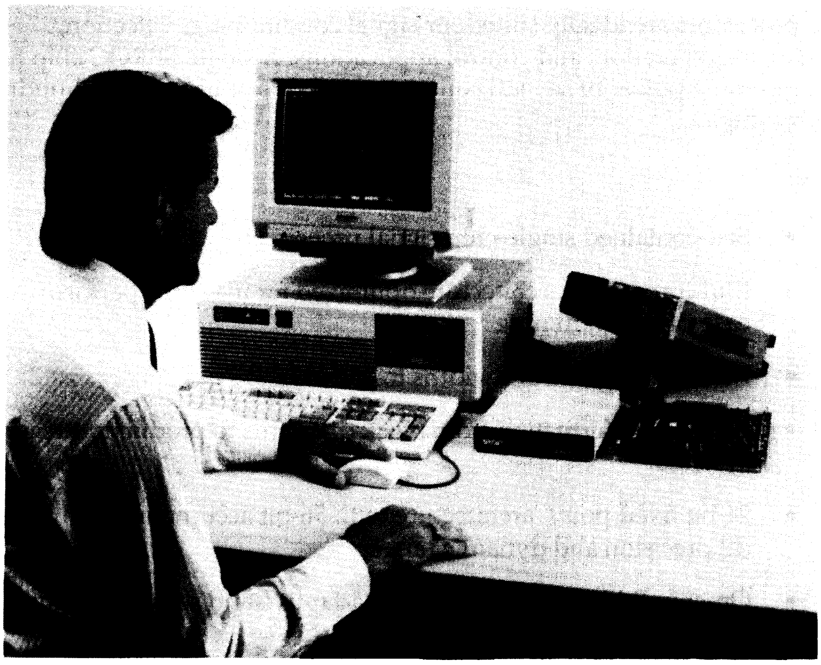


SPROC-1400 Family

The SPROC-1400 is the first in a family of easy-to-use high-performance digital signal processors for analog and digital applications. The chip contains all the necessary elements for efficient signal-processing system implementation. It is supported by the SPROClab development system, which provides direct transformation of designs from block diagrams to production-ready systems. SPROC processors are ideally suited for signal conditioning, detection, measurement, generation, and control applications. A single SPROC chip has the signal processing power of several conventional DSPs or as many as hundreds of analog op amps.

Features:

- Self-contained single-chip signal processor
- Multiprocessor architecture optimized for real-time performance and ease-of-programming
- Real-time signal bandwidths to 250 kHz
- Direct transformation from block diagrams to production-ready systems
- 24-bit fixed-point architecture with 56-bit accumulation giving 144 dB precision and dynamic range
- On-chip RAM for implementing adaptive and self-calibrating systems
- Initialization from 16K byte file in microprocessor or boot PROM
- Up to 64K external RAM accessible
- Four serial ports configurable for 8-, 12-, 16-, or 24-bit data
- 24-bit parallel port supporting Motorola and Intel microprocessor byte order interface protocols
- Software-directed built-in probe acting as an on-chip test point
- On-chip or external clock
- Fully static CMOS technology
- Single 5-volt power supply



SPROClab Development System

The SPROClab development system is a graphical programming environment for the SPROC-1000 series of high-performance digital signal processors. SPROClab supports STAR Semiconductor's unique "Sketch and Realize" approach to DSP design. With SPROClab, designers automatically transform signal-flow diagrams into production-ready high-performance signal-processing systems. SPROClab combines familiar analog system block diagram notation and interactive debugging capability with the power and flexibility of the programmable SPROC chip.

Key features include:

- Easy-to-use signal-flow diagram editor supporting a powerful graphical programming approach
- Library of commonly-used analog and signal-processing function blocks
- Powerful filter design tools
- Fully automatic scheduling and code generation, from signal-flow diagrams to SPROC executable code
- Interactive debugging tools to modify system parameters and observe internal signals - - while a design runs on the SPROC chip
- Automatic symbol table generation for easy interfacing with controlling microprocessors
- Interface to test equipment and the target system for rapid prototyping and debugging
- Support for arbitrarily complex designs, including multiple SPROC chip implementations
- PC and MS-DOS platform

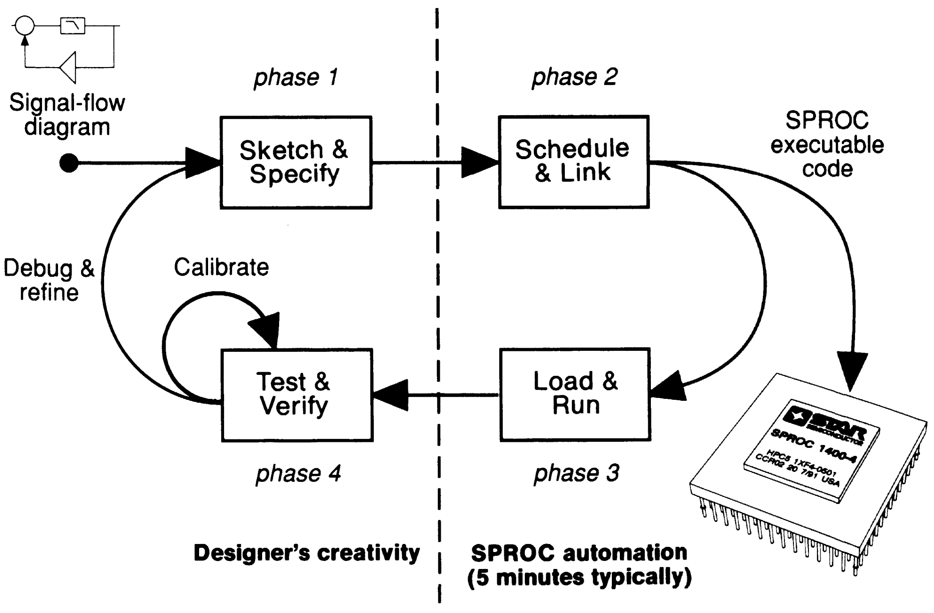
Programming Basics

SPROClab's unique "Sketch and Realize" design capability is made possible by the SPROC proprietary multiprocessor architecture. SPROC avoids the trade-off between performance and ease-of-programming plaguing traditional DSP architectures. SPROC-based designs are automatically partitioned among multiple internal processors. Concurrent computations are scheduled for optimal run-time performance. The SPROClab tools take advantage of the unique SPROC architecture to provide the greatest possible degree of end-product performance, development productivity, and design flexibility.

Although SPROC chips can be hand coded, this option usually does not yield significant performance improvement over graphical, block-diagram programming. However, designers who wish to specify custom design elements can create new function blocks and add them to the SPROCcells function library.

SPROClab incorporates many unique features to simplify and speed the development of complex systems. For example, all parameters can be calibrated while the SPROC chip is running in either the evaluation environment or the target system. The SPROC chip includes a software-directed probe for the observation of internal signals on an oscilloscope or other test instruments, under SPROClab control.

The SPROClab development process includes four phases:



Technical Support

Documentation

In addition to this Data Book, the following technical literature is delivered with every SPROClab development system:

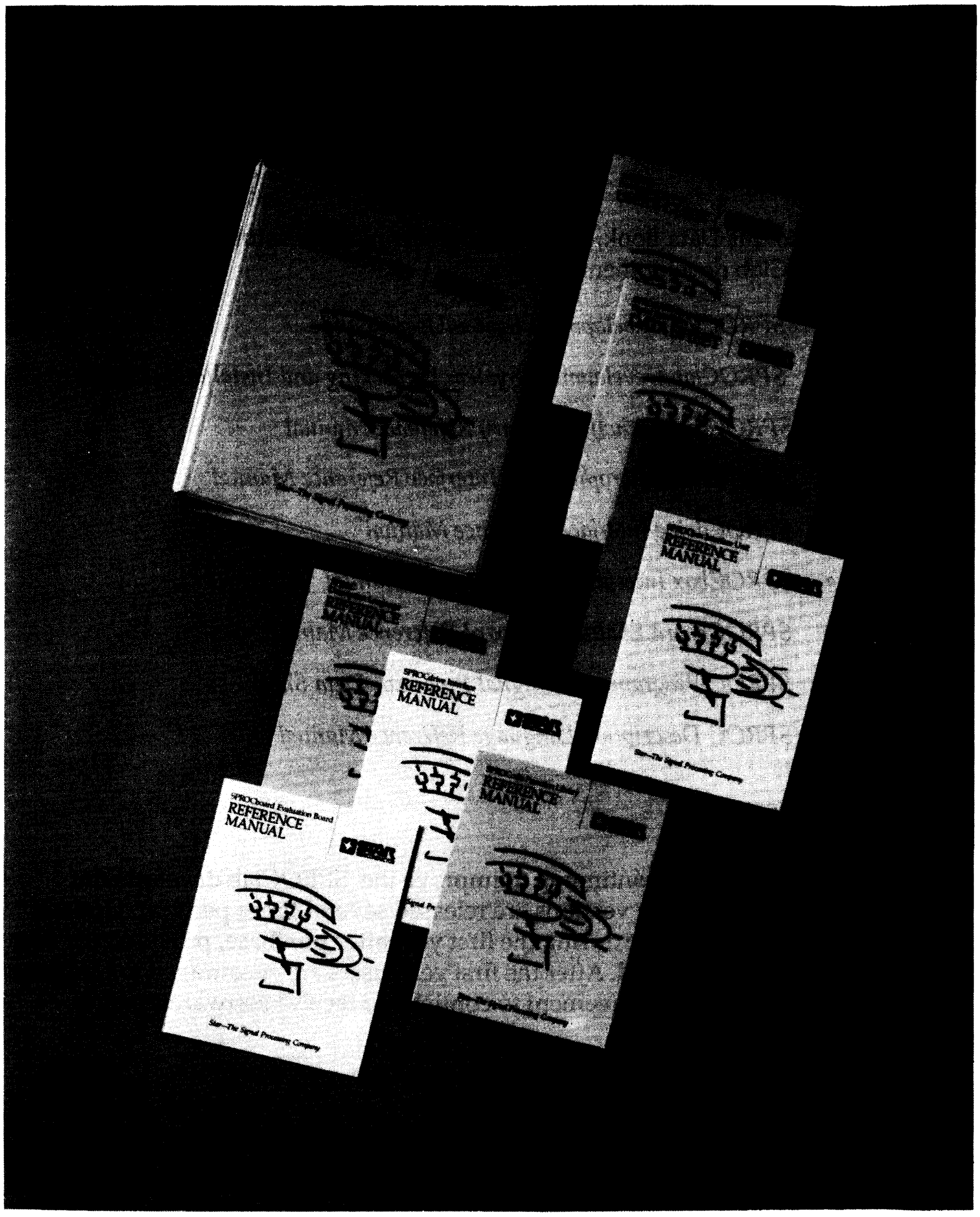
- *SPROClab Development SystemUser's Guide*
- *SPROClab Development System Unpacking and Installation Guide*
- *SPROCcells Function Library Reference Manual*
- *SPROClink Microprocessor Interface Reference Manual*
- *SPROCdrive Interface Reference Manual*
- *SPROCbox Interface Unit Reference Manual*
- *SPROCboard Evaluation Board Reference Manual*
- *SPROC Programmable Signal Processor Data Sheet*
- *SPROC Description Language Reference Manual*

Software Updates

STAR Semiconductor is continuing to improve the SPROClab development system software, and new versions are released several times per year. Updates are provided free of charge during the first year after purchase, provided the user returns the registration card. After the first year, users are encouraged to purchase a Software Maintenance Agreement to continue to receive software updates.

Field Technical Specialists

STAR Semiconductor provides local technical support to customers through a network of Field Technical Specialists (FTSs). For the name and phone number of the nearest STAR FTS, call one of the sales offices listed in Section 7, *Supplementary Information,* of this Data Book.


SPROCboard Evaluation Board
REFERENCE MANUAL
REFERENCE MANUAL
REFERENCE MANUAL
REFERENCE MANUAL
Star—The Signal Processing Company

Applications Assistance

STAR Semiconductor provides applications assistance to STAR customers. For applications assistance, call Applications Engineering on (908) 647-9400.

Bulletin Board

To provide customers with up-to-date information and an immediate response to questions, STAR Semiconductor provides 24-hour access to an electronic bulletin board.

STAR Semiconductor Technical Bulletin Board:

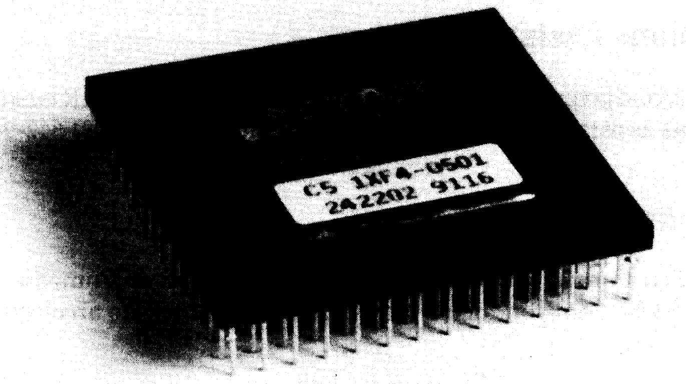
(908) 647-2505

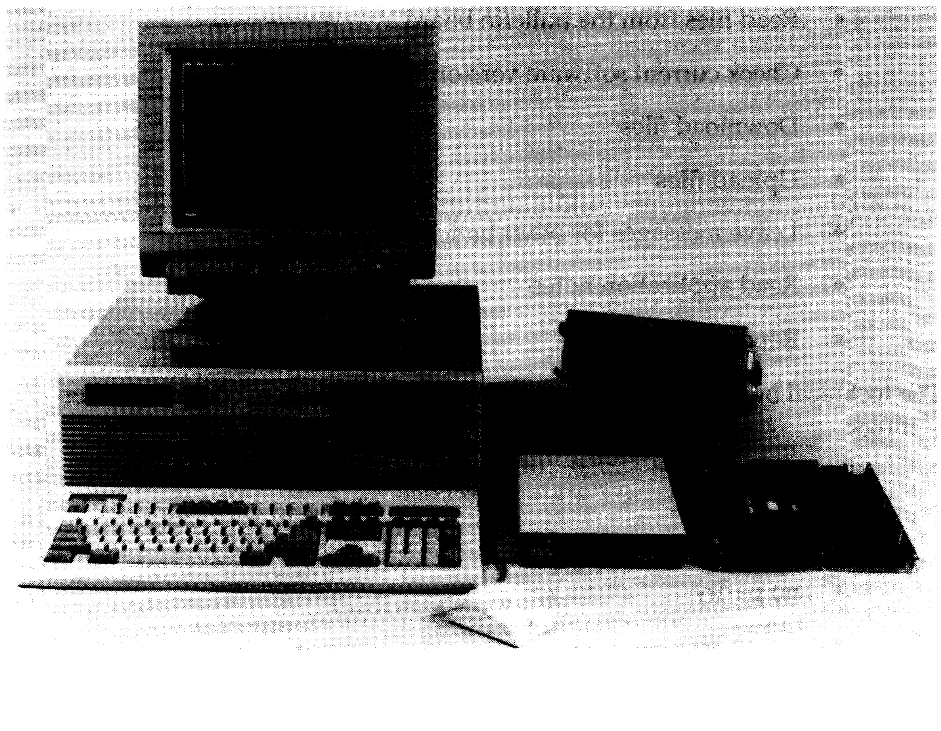
The technical bulletin board provides the following services to all registered STAR customers:

- Read files from the bulletin board
- Check current software version numbers
- Download files
- Upload files
- Leave messages for other bulletin board users
- Read application notes
- Report problems

The technical bulletin board requires the following modem communication settings:

- 2400 baud
- 8 bit
- no parity
- 1 stop bit


C5 1XF4-0501
242202 9116



Section 2
Guide to Products

Product Index

STAR Semiconductor's first generation of programmable, single-chip signal processors offers a variety of multiprocessor, memory, and interface configurations to suit applications with diverse function, performance, and cost requirements.

The SPROClab development system is a complete suite of hardware, software tools, and documentation for developing and debugging SPROC-based systems. SPROClab allows designers to implement complex signal processing systems in minutes and debug them with unprecedented ease. SPROClab's software tools run on IBM-compatible PCs and communicate with target systems via SPROClab's hardware modules.

The following table provides a reference listing of products offered by STAR Semiconductor. Products include hardware, software, and technical documentation.

STAR Semiconductor Product Listing

Product	Description	Part #
SPROClab Development System	Complete graphical programming environment for SPROC-1000 series of high-performance signal processors. Includes: 1 SPROClab Software Kit, SDS-1000-00 1 SPROCbox Interface Unit, SDH-1100-00 1 SPROCboard Evaluation Board, SDH-1200-00 1 Security Key, SDH-1400-00 1 Power Supply, SDH-1300-00 1 AC Power Cord, SDH-131x-00 1 Auxiliary Power Cable, SDH-1140-00 1 Access Port Cable, SDH-1130-00 1 PC Interface Cable, SDH-1410-00 1 SPROClab Development System Document Set, STI-1000-00	SDH-1000-00
SPROC-1000 Series Programmable Signal Processor	One of a series of single-chip, programmable digital signal processors. One chip is included in SPROCboard, SHD-1200-00	see explanation of naming convention following this table
SPROClab Hardware		
SPROCbox Interface Unit	Microprocessor-based interface from a SPROC target system to the development system. Included in the SPROClab Development System, SDH-1000-00.	SDH-1100-00

STAR Semiconductor Product Listing (Continued)

Product	Description	Part #
SPROCboard Evaluation Board	Demonstration board containing interface converters and logic. Includes: 1 SPROC-1000 Series Programmable Signal Processor. Included in the SPROClab Development System, SDH-1000-00.	SDH-1200-00
Security Key	Connects to user's PC parallel port to enable use of SPROClab software. Included in the SPROClab Development System, SDH-1000-00.	SDH-1400-00
Power Supply	+5V, 12V; 115/230 volts, 50/60 Hz. Included in the SPROClab Development System, SDH-1000-00.	SDH-1300-00
AC Power Cord	Connect power supply to AC outlet. Included in the SPROClab Development System, SDH-1000-00.	part number depends on country
	North America	SDH-1310-00
	United Kingdom	SDH-1311-00
	Germany	SDH-1312-00
Auxiliary Power Cable	Daisy chain DC power connection from SPROCbox to SPROCboard. Included in the SPROClab Development System, SDH-1000-00.	SDH-1140-00

STAR Semiconductor Product Listing (Continued)

Product	Description	Part #
Access Port Cable	Supports SPROCbox to SPROC chip communications. Included in the SPROClab Development System, SDH-1000-00.	SDH-1130-00
PC Interface Cable	EIA RS-232C/CCITT 0.24 Included in the SPROClab Development System, SDH-1000-00.	SDH-1410-00
SPROClab Software		
SPROClab Software Kit	Suite of software tools to support the entire SPROC development process. Includes: 1 SPROCview Graphical Design Interface, SDS-1100-00 1 SPROCcells Function Library, SDS-1400-00 1 SPROCfil Filter Design Interface, SDS-1300-00 1 SPROCbuild Utility 1 SPROCdrive Interface 1 SPROClink Microprocessor Interface 1 SPROCinstall Installation Utility, SDS-1200-00 Included in the SPROClab Development System, SDH-1000-00.	SDS-1000-00
SPROCview Graphical Design Interface	Schematic entry package interface. Supports use of OrCAD schematic entry software. Included in the SPROClab Software Kit, SDS-1000-00.	SDS-1100-00

STAR Semiconductor Product Listing (Continued)

Product	Description	Part #
SPROCcells Function Library	Library of signal processing functions, with icons. Included in the SPROClab Software Kit, SDS-1000-00.	SDS-1400-00
SPROCfil Filter Design Interface	Custom filter design tool. Included in the SPROClab Software Kit, SDS-1000-00.	SDS-1300-00
SPROCbuild Utility	Automatic code generator / scheduler. Included in the SPROClab Software Kit, SDS-1000-00.	none
SPROCdrive Interface	Loading and debugging tool. Included in the SPROClab Software Kit, SDS-1000-00.	none
SPROClink Microprocessor Interface	Software to support embedded system development of microprocessor C applications including the SPROC chip. Included in the SPROClab Software Kit, SDS-1000-00.	none
SPROCinstall Installation Utility	Included in the SPROClab Software Kit, SDS-1000-00.	SDS-1200-00
Optional SPROClab Software		
SPROCsim Simulator	Software simulator of SPROC chip.	SDS-1500-00
VIEWlogic option for SPROCview Graphical Design Interface	Supports schematic entry using VIEWlogic software.	SDS-1120-00

STAR Semiconductor Product Listing (Continued)

Product	Description	Part #
SPROClab Technical Documentation		
SPROClab Development System Document Set	Comprehensive documentation covering all hardware and software tools in the SPROClab Development System. Includes: 1 SPROClab Development System User's Guide, UG1000 1 SPROClab Development System Unpacking and Installation Guide, IG1000 1 SPROCdrive Interface Reference Manual, SDIR1000 1 SPROCcells Function Library Reference Manual, CELR1000 1 SPROClink Microprocessor Interface Reference Manual, LINR1000 1 SPROC Description Language Reference Manual, SDLR1000 1 SPROCbox Interface Unit Reference Manual, IUR1000 1 SPROCboard Evaluation Board Reference Manual, EBR1000 1 SPROC Programmable Signal Processor Data Sheet (number depends on specific chip) Included in the SPROClab Development System, SDH-1000-00.	STI-1000-00
SPROClab Development System User's Guide	Guidelines and tutorials on using all the software tools in the SPROClab Development System. Included in the SPROClab Development System Document Set, STI-1000-00.	UG1000

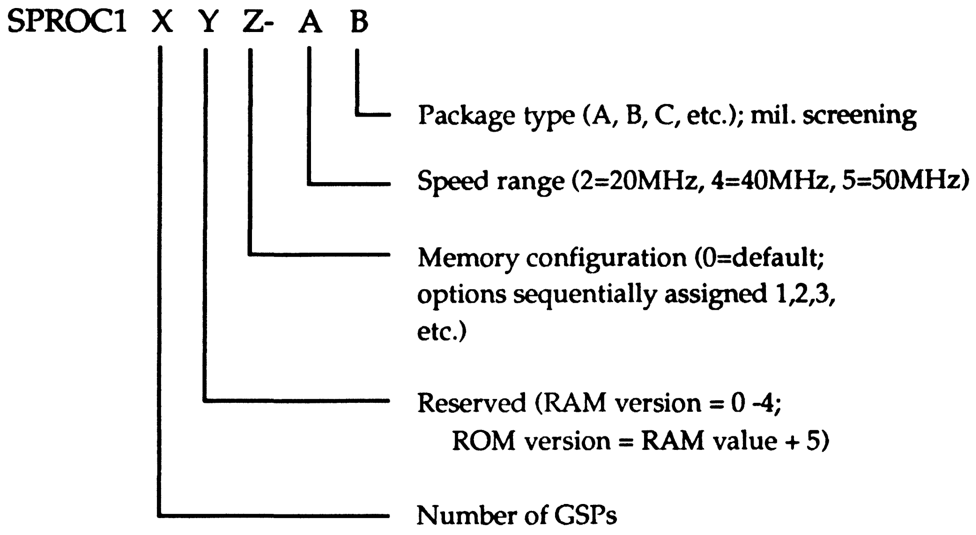
STAR Semiconductor Product Listing (Continued)

Product	Description	Part #
SPROClab Development System Unpacking and Installation Guide	Guidelines and instructions for installing SPROClab hardware and software. Included in the SPROClab Development System Document Set, STI-1000-00.	IG1000
SPROCdrive Interface Reference Manual	Command reference for the SPROCdrive software. Included in the SPROClab Development System Document Set, STI-1000-00.	SDIR1000
SPROCcells Function Library Reference Manual	Reference covering all cells in the SPROCcells library. Included in the SPROClab Development System Document Set, STI-1000-00.	CELR1000
SPROClink Microprocessor Interface Reference Manual	Reference covering the SPROClink software. Included in the SPROClab Development System Document Set, STI-1000-00.	LINR1000
SPROC Description Language Reference Manual	Overview and reference covering the SPROC Description Language (SDL). Included in the SPROClab Development System Document Set, STI-1000-00.	SDLR1000
SPROCbox Interface Unit Reference Manual	Reference covering the SPROCbox. Included in the SPROClab Development System Document Set, STI-1000-00.	IUR1000
SPROCboard Evaluation Board Reference Manual	Reference covering the SPROCboard. Included in the SPROClab Development System Document Set, STI-1000-00.	EBR1000

STAR Semiconductor Product Listing (Continued)

Product	**Description**	**Part #**
SPROC Programmable Signal Processor Data Sheet	Technical specification of the SPROC chip. Included in the SPROClab Development System Document Set, STI-1000-00.	number depends on the specific chip
SPROCsim Simulator Reference Manual	Reference covering the SPROCsim software.	SIMR1000
SPROCbuild Utility Reference Manual	Reference covering the SPROCbuild software.	SBR1000
SPROC Data Book	Comprehensive overview of STAR Semiconductor technology and products.	DB1000

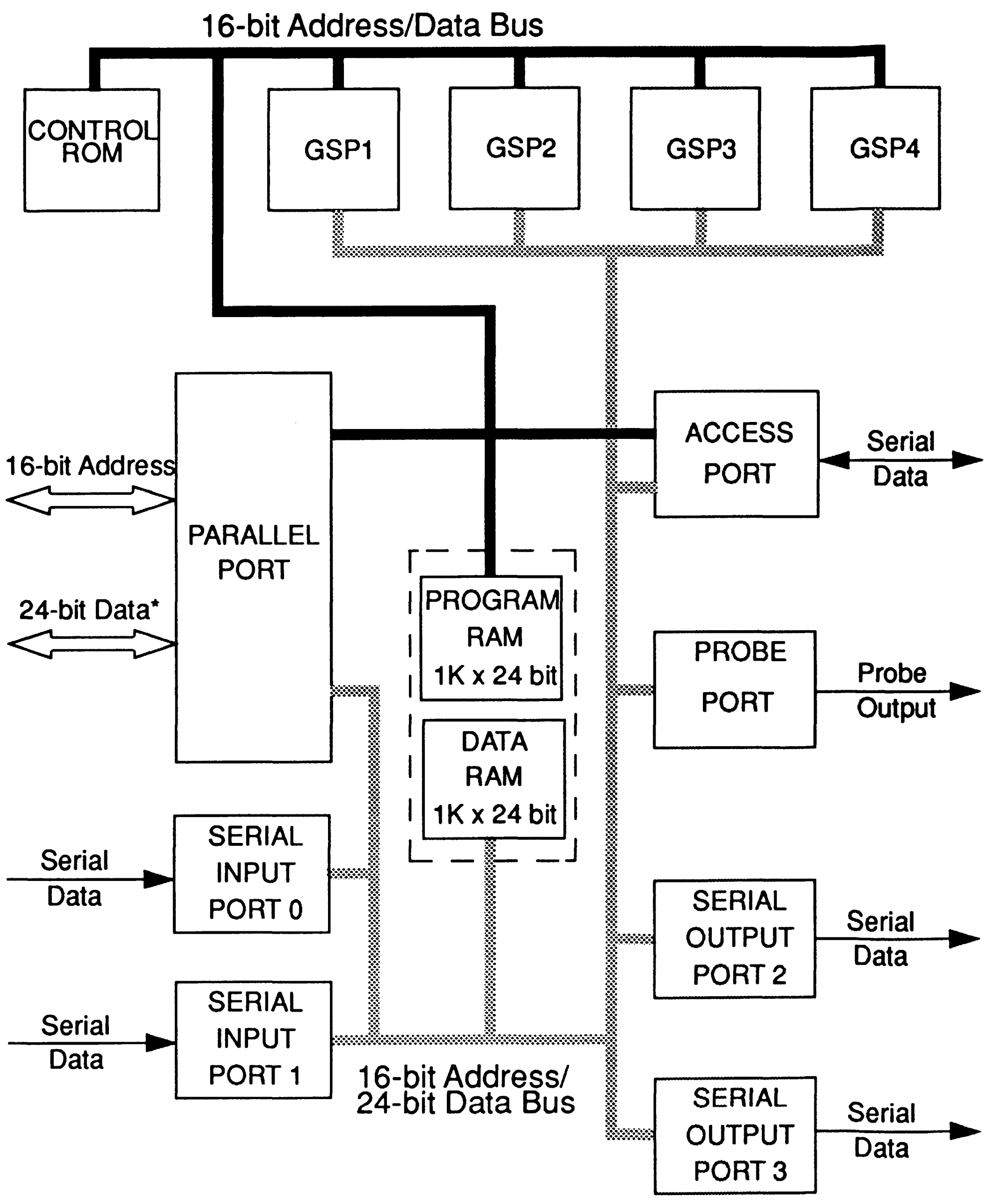
Chip Naming: SPROC-1000 Series


SPROC1 X Y Z- A B
Package type (A, B, C, etc.); mil. screening
Speed range (2=20MHz, 4=40MHz, 5=50MHz)
Memory configuration (0=default; options sequentially assigned 1,2,3, etc.)
Reserved (RAM version = 0 -4; ROM version = RAM value + 5)
Number of GSPs

Product Overview

STAR Semiconductor produces easy-to-use, high-performance digital signal processing integrated circuits for analog and digital applications. STAR's unique SPROC technology enables the direct transformation of block diagrams to production-ready signal-processing systems in minutes. SPROC combines an innovative digital signal processing architecture with powerful development tools for unprecedented ease of use without sacrificing performance. With STAR's products, both analog designers and experienced DSP users can enhance the features and performance of their systems while dramatically reducing development time and cost.

The SPROC signal processing family of products consists of programmable integrated circuits and a development system including a comprehensive line of hardware and software.


16-bit Address/Data Bus
CONTROL ROM
GSP1
GSP2
GSP3
GSP4
16-bit Address
24-bit Data*
PARALLEL PORT
ACCESS PORT
Serial Data
PROGRAM RAM 1K x 24 bit
DATA RAM 1K x 24 bit
PROBE PORT
Probe Output
Serial Data
SERIAL INPUT PORT 0
Serial Data
SERIAL INPUT PORT 1
SERIAL OUTPUT PORT 2
Serial Data
SERIAL OUTPUT PORT 3
Serial Data
16-bit Address/ 24-bit Data Bus

SPROC-1400 Family

STAR Semiconductor's SPROC-1400 family chips includes four independent General Signal Processors (GSPs) with 1k x 24-bit program memory, a shared Central Memory Unit (CMU) with 1K x 24-bit RAM, and both serial and parallel I/O ports. The SPROC-1400 family incorporates a powerful and flexible mechanism for managing the data and instruction flows between the serial ports, CMU, and each GSP through 16-bit address and 24-bit data buses.

A SPROC-1400 chip provides two classes of I/O interfaces that are compatible with the majority of existing microprocessors, memories, and peripheral devices. The serial interface consists of four independent serial ports - - two input and two output. Each serial port can operate with its own internal or external clock, input/output word width, and bit ordering. The parallel port is a single 24-bit, asynchronous, bidirectional interface. Users can select word widths (up to 24 bits) and byte orderings for data transferred over the parallel port. Through the parallel port, the SPROC can be dynamically reprogrammed by a microprocessor while operating in slave mode.

An access port provides the link between the SPROCbox interface unit and the SPROC-1400 chip. SPROCdrive software allows the user to dynamically modify system parameter values through the access port for calibration and debugging, thus furnishing the means to observe and control system function and performance under various conditions.

A built-in software-directed signal probe facilitates interactive development and debugging of SPROC-based systems. The probe circuit steers selected internal signals to a dedicated output port. Using the SPROCdrive software, designers can access internal signals for display or measurement.

Multiple interconnected SPROC chips - - with or without a microprocessor or controller - - can implement arbitrarily complex signal-processing functions. SPROC-1400 chips are available in several speed ranges, from 20 MHz to 50 MHz, for real-time bandwidths to 250 kHz.

SPROClab Development Environment

The SPROClab development system is a complete set of hardware and software tools that you use with your PC to create, test, and debug digital signal processing designs. It was created to support the development of code for the SPROC signal processing chip.

The development system provides an interactive design environment that lets you create processing subsystems in graphical form, as signal-flow diagrams, and implement those subsystems easily and efficiently on the SPROC chip. Using the system, you can develop efficient signal processing subsystems without having to manually write code.

Together with your PC and oscilloscope or other verification equipment, the development system supports the entire development process, including interactive debugging and design verification. Once you complete design development, you can easily include the signal processing subsystem in your actual application using a SPROC chip and the code generated by the development system.

SPROClab Components

SPROClab consists of software tools and hardware modules that support the entire SPROC development process. Software tools comprise:

- SPROCview graphical signal-flow editor
- SPROCcells function block library
- SPROCfil custom filter design tool
- SPROCbuild fully automatic scheduler
- SPROCdrive loading and debugging tool
- SPROClink microprocessor interface

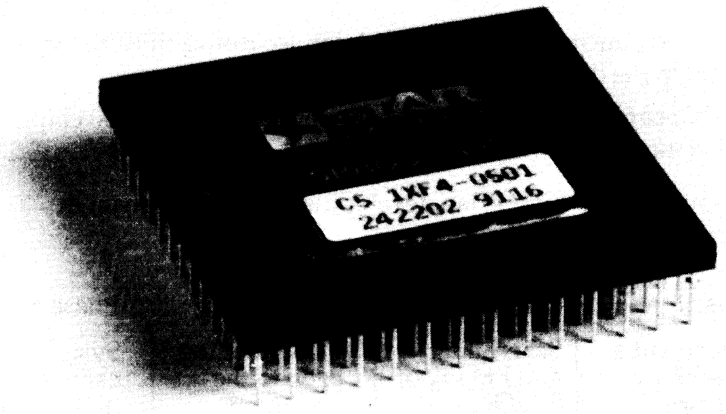
Hardware tools comprise:

- SPROCbox microprocessor-based interface from a SPROC target system to the development system
- SPROCboard evaluation board, containing a SPROC 1400 chip, analog interface converters, and logic.

A detailed description of the SPROClab development system is given in Section 3, *Product Technical Data.*

SPROClab tool	Phase 1: Creation of a signal-flow diagram	Phase 2: Scheduling and linking of SPROC executable code	Phase 3: Downloading code to the execution environment	Phase 4: System calibration, debugging, and refinement
SPROCview	▨			
SPROCcells	▨			
SPROCfil	▨			
SPROCbuild		▨		
SPROCdrive			▨	▨
SPROClink			▨	▨
SPROCbox & SPROCboard			■	■

▨ Software ■ Hardware


CS 1XF4-0601
242202 9116

Section 3
Product Technical Data

Note: *The information contained in this section is preliminary and may change without notice. Always consult the most recent version of the appropriate technical documentation for current information to be used when creating designs for the SPROC signal processor.*

SPROC-1400 Signal Processor

The SPROC-1400 processor is part of the SPROC-1000 series of easy-to-use digital signal processors. They contain all program memory, data memory, signal logic, and microprocessor interface logic necessary for efficient system design, implementation, and test.

General Description

The SPROC-1400 processors use the basic SPROC-1000 series chip design and include four on-board general signal processors (GSPs). Table 3-1 lists the available SPROC-1400 processors.

Table 3-1. SPROC-1400 Digital Signal Processors

Chip Part Number	Maximum Clock Frequency
SPROC-14xx-5x	50 MHz

Note: An x in the part number indicates "any entry". Refer to the explanation of part numbering conventions provided in the *Additional Specifications* section later in this section.

SPROC-1400 processors may be configured in standalone (master) mode, or in embedded (slave) mode. In slave mode, a SPROC chip may be connected to other SPROC chips, or to a microprocessor.

Features

- Self-contained single chip signal processing subsystems
- Optimized multiprocessor architecture
- Real-time signal bandwidths up to 250 KHz
- 24-bit internal precision with 56-bit accumulation
- Internally generated clock (up to 50 MHz)
- Dynamically reprogrammable on-chip RAM
- Four serial ports configurable for 8-, 12-, 16-, or 24-bit data
- 24-bit parallel port configurable for 8-, 16-, or 24-bit data bus widths
- Software-directed built-in probe
- Fully static CMOS technology
- Single 5-volt power supply
- Initialization from 16-Kbyte file in microprocessor or external ROM-based self initialization
- Parallel port supports connection to common Motorola or Intel microprocessors
- Stand-alone (master) or embedded (slave) operating modes
- Dedicated serial port for development system interface

The Central Memory Architecture

The SPROC-1000 series uses a central memory architecture that is optimized for concurrent processing of complex, interrelated signal flows. At the center of the architecture is a multi-ported shared data memory called the *central memory unit* (CMU). Four *general signal processors* (GSPs) on the chip perform computation and provide parallel processing. Input/output *data flow managers* (DFMs) coordinate simultaneous data streams. Serial channels interface signals, and parallel interfaces enable connection to external processors. Special I/O interfaces provide connections to the SPROClab development system and to the on-chip probe.

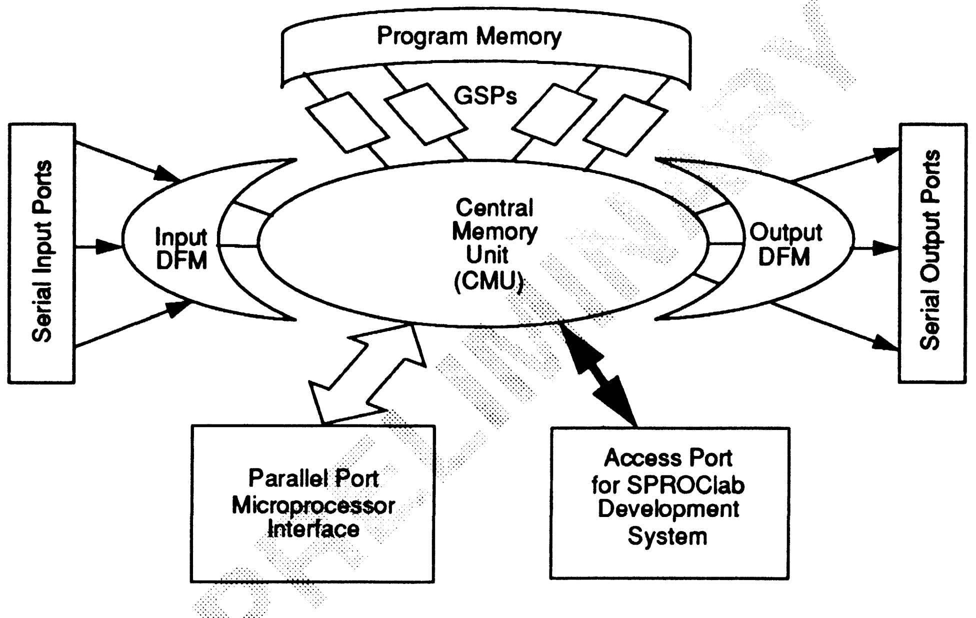

Program Memory
GSPs
Serial Input Ports
Input DFM
Central Memory Unit (CMU)
Output DFM
Serial Output Ports
Parallel Port Microprocessor Interface
Access Port for SPROClab Development System

Figure 3-1. SPROC-1000 Series Central Memory Architecture

This central memory architecture represents a departure from the single processor approach to concurrent processing. Instead of using time division multiplexing of the processing unit, through interrupts, the SPROC central memory architecture uses multiprocessors and time division multiplexing of memory. No interrupts are necessary to handle multiple data streams.

The Central Memory Unit (CMU)

The CMU is a multi-ported data space. It uses a frame composed of time slots, or memory access periods, allotted for each GSP and for I/O. The basic frame represents one SPROC chip machine cycle (five master clock cycles) and includes five time slots of one master clock cycle each.

Time slots 1 through 4 are used by the GSPs. During time slot 1, GSP 1 can read or write the CMU; during time slot 2, GSP 2 can read or write the CMU, and so on.

Time slot 5 is used by I/O operations. It is submultiplexed into eight divisions for parallel I/O and other I/O operations. One half of the time (during every other machine cycle), time slot 5 is used by the parallel port. Through this "subslot", the parallel port can support signal or parametric data flow without interrupting the processing of any of the GSPs. The other half of the time (on machine cycles when the parallel port does not use slot 5), slot 5 supports one of seven additional subslot divisions for serial ports, the access port, and the on-chip probe.

Figure 3-2 shows the time slots in the CMU frame.

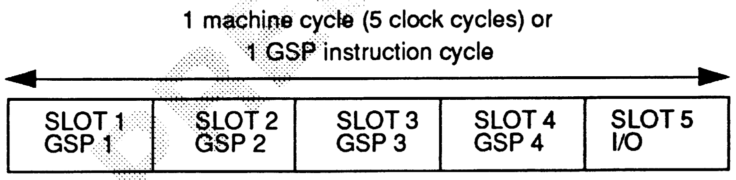


Figure 3-2. CMU Time Slot Divisions

One machine cycle is equal to one GSP instruction cycle. A GSP instruction cycle can be determined using the following formula:

$$\text{GSP instruction cycle} = 5 * \left(\frac{1}{\text{MASTER\_CLOCK}} \right)$$

The General Signal Processors (GSPs)

The GSPs are 24-bit fixed-point processors optimized for signal processing functions and for the SPROC central memory architecture. These processors can be used either individually or in groups depending on the application's requirements. The SPROCbuild utility in the SPROClab development system automatically generates all necessary scheduling and task allocation functions required for the GSPs. The GSPs execute signal processing code concurrently, and without interruption.

The Data Flow Managers (DFMs)

The DFMs coordinate the filing of input and output data into or out of the CMU with no impact on GSP performance. They communicate with the GSPs and other on-chip elements through a 24-bit data bus, and with off-chip data converters and digital data streams through programmable serial ports. Because the CMU is a multi-ported memory space, this activity can take place in parallel with the signal processing, enabling the SPROC chip to service high sample rates with high efficiency. The DFMs set up either double buffers or vector FIFOs in the CMU.

I/O Interfaces

Communication with a processor and with the development system is achieved via the parallel port and the access port.

The parallel port allows an external processor to read or write any part of the CMU completely asynchronously from any other on-chip activity. The SPROC chip provides all the necessary interfacing signals to the supported processor, as well as the necessary translation from the word width within the SPROC chip to that of the external processor.

The access port is a custom I/O port that provides connection to the SPROClab development system.

Other Elements

Other circuitry, not shown in Figure 3-1, handles the software-directed real-time probe and process scheduling activity.

Functional Description

Overview

The SPROC-1400 chip is a general purpose signal processing chip with internal parallel-processing resources. Its architecture contains four independent general signal processors (GSPs), a shared central memory unit (CMU) with 1K of 24-bit RAM data memory and 1K of 24-bit code memory, and both serial and parallel I/O ports. Control and status registers are memory mapped into the internal address space. The chip incorporates a powerful and flexible mechanism for managing the data and instruction flow between the serial ports, CMU, and each GSP.

The chip is implemented in a submicron, CMOS process that supports a master clock rate of up to 50 MHz. With its optimized architecture and 40 MIPS performance potential, the SPROC-1400 chip is an appropriate solution for wide-band, real-time signal processing applications. Multiple interconnected SPROC chips -- with or without a microprocessor or controller -- can implement arbitrarily complex signal processing functions.

Figure 3-3 shows the functional diagram of the SPROC-1400 chip.

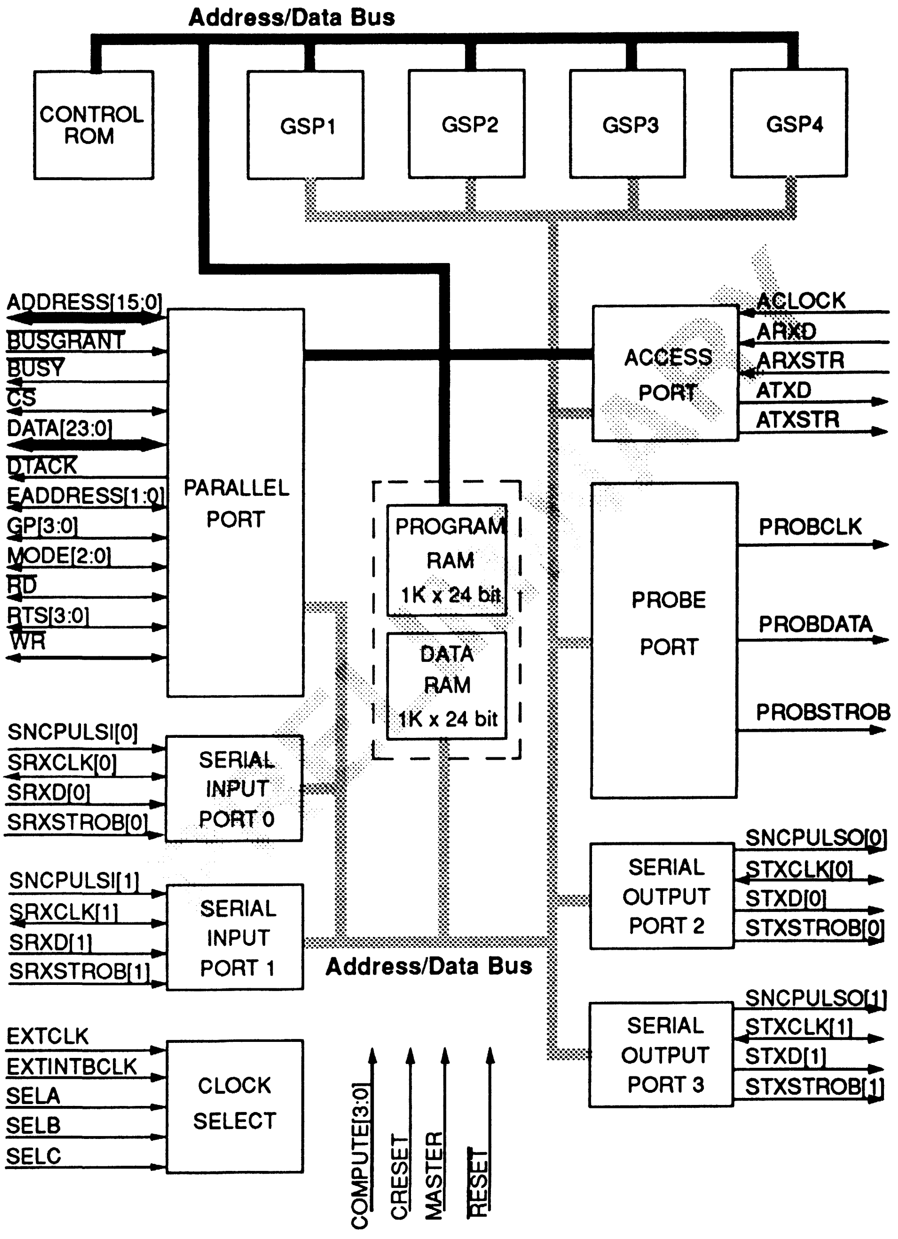


Figure 3-3. SPROC-1400 Chip Functional Diagram

A SPROC-1400 chip provides both serial and parallel I/O interfaces, designed to be compatible with the majority of existing microprocessors, memories, and peripheral devices.

The *serial interface* has a capacity of four serial ports -- two input and two output. Each serial port can operate independently with its own internal or external clock, input/output word width, and bit ordering.

The *parallel port* is a single 8-, 16-, or 24-bit, asynchronous, bidirectional interface. Data can be transferred over the parallel port in dynamically selectable word widths and byte orderings. Through the parallel port, the SPROC chip can be dynamically reprogrammed by a host microprocessor while operating in slave mode.

A special serial interface called the *access port* is used by the SPROCbox interface unit to communicate with the SPROC chip from the development system. The development system's SPROCdrive interface software allows the user to dynamically modify system parameters through this access port during debugging, thus furnishing the means to observe system performance under various conditions.

The SPROC-1400 chip's master clock can be supplied from its internal ring oscillator, or from an external source.

The SPROC-1400 chip includes an on-chip signal probe to aid in the development and debugging of SPROC-based designs. Under control of the SPROCdrive interface software, any internal signal may be accessed for display or measurement.

Clock Selection

The master clock can either be externally supplied or internally generated, as governed by the EXTINTBCLK input.

External Clock

Configure an external clock by setting the EXTINTBCLK input HIGH and connecting the external clock as an input to the EXTCLK pin. The frequency of the external clock may not exceed the maximum clock frequency supported by the SPROC-1400 chip, as indicated by the chip part number. Table 3-2 lists the maximum master clock frequencies for SPROC-1400 chips.

Table 3-2. SPROC-1400 Chip Frequencies

Chip Part Number	**Maximum Clock Frequency (MHz)**
SPROC-14xx-5x	50

Note: An x in the part number indicates "any entry". Refer to the explanation of part numbering conventions provided in the *Additional Specifications* section later in this section.

Internal Clock

Configure an internally generated master clock by setting the EXTINTBCLK input LOW.

The SELA, SELB, and SELC inputs can be used to select one of several frequencies for the internally generated master clock. Table 3-3 lists the frequency selections obtained for the various input settings.

Table 3-3. Internal Master Clock Frequency Selections

SELA INPUT	SELB INPUT	SELC INPUT	FREQUENCY (nominal) of SPROC-14xx-5x CHIP
HIGH	HIGH	HIGH	100 MHz*
HIGH	HIGH	LOW	81 MHz*
HIGH	LOW	HIGH	69 MHz*
HIGH	LOW	LOW	60 MHz*
LOW	HIGH	HIGH	50 MHz
LOW	HIGH	LOW	43 MHz
LOW	LOW	HIGH	37 MHz
LOW	LOW	LOW	28 MHz
* Operation at this speed not guaranteed.			

Note: *These data are preliminary only. Actual operation of the SPROC-14xx-5x chip remains to be characterized.*

The Memory Map

The 4K x 24-bit SPROC-1400 chip memory map is allocated as follows:

- Hex addresses 000 through 3FF -- program RAM
- Hex addresses 400through 4FF -- internal memory-mapped registers
- Hex addresses 800 through BFF -- parametric and data RAM

Table 3-4 lists the memory-mapped registers. Descriptions of these registers are given in the appropriate sections later in this data sheet.

Table 3-4. SPROC Memory Map

ADDRESS (HEX)	CONTENTS
000 - 3FF	Program RAM
400	reserved
405	Serial port reset (write)
406	Global break entry (write)
407	Global break exit -- start -- (write)
410	Serial input port 0 internal clock rate
411	Serial input port 1 internal clock rate
412	Serial output port 2 internal clock rate
413	Serial output port 3internal clock rate
418	reserved
440 - 447	Serial input port siport0 configuration
448 - 44F	Serial input port siport1 configuration
450 - 457	Serial output port soport0 configuration
458 - 45F	Serial output port soport1 configuration
480 - 487	Probe port
488 - 48F	Probe serial output port
490 - 495	reserved
4FB - 4FF	Parallel port registers
800 - BFF	Data RAM
800 - 813	Trigger flags (write to activate)
C00 - FFF	reserved

The Parallel Port

Overview

The parallel port is a 24-bit asynchronous, bidirectional port with a 16-bit (64K) address bus. The port allows for 8-, 16-, or 24-bit parallel data transfers between the SPROC chip and an external controller, memory-mapped peripheral, or external memory. The port has programmable WAIT states to allow for slow memory access. A data acknowledge signal is also generated for this interface.

Two operating modes -- *master* and *slave* -- allow the SPROC chip to operate either as a system controller (master mode), or as a memory-mapped peripheral to an external controller (slave mode). An input pin, MASTER, is dedicated to setting master or slave mode operation. In master mode, the SPROC chip automatically up-loads its configuration program from an external 8-bit PROM into internal RAM, at the initiation of boot. In slave mode, the chip relies on an external controller for its configuration.

A system using multiple SPROC chips must have a single bus controller. This may be an external controller or a master SPROC chip. All other SPROC chips in the system must be configured in slave mode.The bus controller must individually enable the chip select input, $\overline{CS}$, of each slave SPROC chip while the slave chip is being configured.

The 16-bit address field (ADDRESS[15:0]) supports up to 16 SPROC chips interconnected in the same system.

The external controller, memory-mapped peripheral, or memory may communicate with a SPROC chip in 8-, 16-, or 24-bit format. Format selection is accomplished with the MODE[2:0] pins. In 8- or 16-bit formats, the data may be most significant (msb) or least significant (lsb) byte or word first. In 16- and 24-bit modes, data is always msb-justified within the word being transferred, and the lsb byte is zero-filled for 32-bit data transfer (i.e., in the second 16-bit word). To accommodate 8- and 16-bit modes, two extended address bits are included. These bits (EADDRESS[1:0]) are located at the lsb-end of the address bus. In master mode, these are driven output lines. In slave mode, they are configured as inputs and are driven by the external controller.

The following subsections describe data transfers via the parallel port for different sources and destinations. In all types of parallel port data transfers, signal values at the slave SPROC chip's mode (MODE[2:0]) and address (ADDRESS[15:0]) inputs must be stable before the chip select ($\overline{CS}$) and read ($\overline{RD}$), or chip select and write ($\overline{WR}$) request goes LOW. At that time, the address is latched into the slave SPROC chip. Subsequently, after values on the data bus (DATA[23:0]) become valid, data is latched at the destination on the rising edge of the request.

To allow asynchronous communication with slow peripherals in master mode, the parallel port supports programmable WAIT states. A maximum of seven WAIT states are possible, where each state corresponds to one SPROC chip machine cycle, or five master clock pulses.

The parallel port also generates a handshaking signal, $\overline{DTACK}$ (data transfer acknowledge) in slave mode. This normally-HIGH signal goes LOW when the SPROC chip presents valid data in a read operation, or is ready to accept data in a write operation. $\overline{DTACK}$ is cleared when the external $\overline{RD}$ or $\overline{WR}$ strobe goes HIGH.

If enabled, a watchdog timer monitors all data transfers, and resets the parallel port if the transaction time is greater than 256 machine cycles.

Master SPROC Chip Read from Slave SPROC Chip or Peripheral

Prior to initiating the READ, the master SPROC chip must set up the communication mode. This includes 8-, 16-, or 24-bit data select, msb/lsb byte order, and number of WAIT states required for the peripheral. The master's internal parallel port mode register controls these options, and therefore must have been previously written to. In master mode, three bits of the parallel port mode register determine number and order of bytes transferred and are output at pins MODE[2:0]. These pins should be connected to the corresponding slave SPROC chip pins, which function as inputs in slave mode, to ensure the slave's communication mode matches the master's.

After a read cycle is initiated by the master SPROC chip, no further read or write requests to the parallel port are possible until the current read cycle has been completed. The parallel port will set up a stable address and then drive the $\overline{RD}$ strobe LOW. The strobe will remain LOW for the number of WAIT states configured in the master's parallel port mode register, and will then be driven HIGH. The data resident on the data bus will be latched into the master SPROC chip on the rising edge of the $\overline{RD}$ strobe.

If the transmission mode is 8- or 16-bit format, the read cycle will be repeated with the next extended address output, as determined by the state of EADDRESS[1:0], until 24 bits of data have been received. The master's parallel port input register is then updated, and the read cycle is complete. The GSP in the master that initiated the read operation must then read the contents of the parallel port input register. With the read cycle completed, the data bus I/O drivers will be reconfigured as output drivers to prevent the data bus from floating. The address bus will be driven with the last address.

Master SPROC Chip Write to Slave SPROC Chip or Peripheral

A master SPROC chip initates a read or write operation to a slave SPROC chip or a peripheral by reading or writing to an off-chip memory location. Prior to initiating the WRITE, the master SPROC chip must set up the communication mode. This includes 8-, 16-, or 24-bit data select, msb/lsb byte order, and number of WAIT states required for the peripheral. The master's internal parallel port mode register controls these options, and therefore must have been previously written to. In master mode, three bits of the parallel port mode register determine number and order of bytes transferred and are output at pins MODE[2:0]. These pins should be connected to the corresponding slave SPROC chip pins, which function as inputs in this mode, to make the slave's communication mode match the master's.

After a write cycle is initiated by the master SPROC chip, no further read or write requests to the parallel port are possible until the current write cycle is complete. The parallel port will output a stable address and then drive the $\overline{WR}$ strobe LOW. The strobe will remain LOW for the number of WAIT states configured in the master's parallel port mode register. Valid data will be setup on the data bus, and the $\overline{WR}$ strobe will be driven HIGH after the WAIT interval, latching the data into the slave SPROC chip or peripheral. If the interface is configured in 8- or 16-bit mode, the cycle will be repeated until all bytes have been output. After transmission of the last byte or word, the address bus and data bus will remain driven.

Read from Slave SPROC Chip by an External Controller

The external controller will set up address, extended address, and mode inputs, and drive the SPROC chip's chip select input LOW. (If the communication mode will never change, the SPROC chip's MODE[2:0] inputs could be tied to the appropriate logic levels.) The external controller will then drive $\overline{\text{RD}}$ LOW, which will latch the address, extended address (EADDRESS[1:0]), and mode inputs into the slave SPROC chip. The SPROC chip will asynchronously fetch data from the requested internal RAM location. Data will be latched into the external controller when it drives the $\overline{\text{RD}}$ line HIGH again. The controller must ensure that enough time has been given to the slave SPROC chip to fetch the data, given the asynchronous nature of the interface. Alternatively, the SPROC chip drives its normally-high $\overline{\text{DTACK}}$ (data transfer acknowledge) LOW after it has completed the READ, and the controller need only wait for this event before raising $\overline{\text{RD}}$. At that time, the SPROC chip would correspondingly raise $\overline{\text{DTACK}}$.

If the interface is configured for 8- or 16-bit communication, the external controller must set up multiple extended addresses and $\overline{\text{RD}}$ strobes (see *Data Transfer Modes*).

Write to Slave SPROC Chip by an External Controller

The external controller will set up address, extended address, and mode inputs, and drive the SPROC chip's chip select input LOW. (If the communication mode will never change, the SPROC chip's MODE[2:0] inputs could be tied to the appropriate logic levels.) The external controller will then drive $\overline{\text{WR}}$ LOW, which will latch the address, extended address, and mode inputs into the slave SPROC chip. When the controller returns $\overline{\text{WR}}$ to HIGH, the data present on the data bus will be latched into the SPROC chip.

If the interface is configured for 8- or 16-bit communication, the external controller must set up multiple extended addresses and $\overline{\text{WR}}$ strobes (see *Data Transfer Modes*).

After the final byte or word has been transferred, the data will be asynchronously written to the requested address in SPROC chip RAM.

Data Transfer Modes

MODE[0] and MODE[1] determine the number of bytes transferred per $\overline{RD}/\overline{WR}$ strobe. MODE[0] distinguishes between a partial word of 8- or 16-bits, and a full 24-bit word. MODE[1] distinguishes between the partial transfers of 8- and 16-bits. All data transfers are aligned with the least significant byte of the data bus. For 16-and 24-bit modes, the most significant byte is left-justified within the data word, with descending order of significance in lower order data bus bytes.

MODE[1]	MODE[0]	DATA
0	0	8-bit
1	0	16-bit
X	1	24-bit

MODE[2] determines the byte or word ordering for 8- and 16-bit modes:

MODE[2]	BYTE/WORD ORDER
0	msb first
1	lsb first

EADDRESS[1,0], the extended address, specifies which portion of the full 24-bit word is currently being output on the data bus for 8- and 16-bit modes:

8-BIT MODE, MODE[2]=0

EADDRESS[1]	EADDRESS[0]	BYTE
0	0	msb
0	1	mid
1	0	lsb
1	1	unused (write) 0 byte (read)

8 BIT MODE, MODE[2]=1

EADDRESS[1]	EADDRESS[0]	BYTE
0	0	unused (write) 0 byte (read)
0	1	lsb
1	0	mid
1	1	msb

In receive data mode, the lower byte of the lsb 16-bit word is unused by the SPROC chip. Similarly, in transmit mode, the lower byte of the lsb 16-bit word is filled with zeros. All data is msb-justified. The word ordering for 16-bit data is determined by EADDRESS[1]:

16 BIT MODE, MODE[2]=0

EADDRESS[1]	EADDRESS[0]	WORD
0	X	msb
1	X	lsb

16 BIT MODE, MODE[2]=1

EADDRESS[1]	EADDRESS[0]	WORD
0	X	lsb
1	X	msb

Data transfer in 8- and 16-bit modes is completed when the EADDRESS lines designate the final byte or word, namely, the lsb when MODE[2] is LOW, or the msb when MODE[2] is HIGH.

Boot Mode

A SPROC chip enters boot mode when it is configured as a master SPROC chip (its MASTER input is HIGH) and the reset input ($\overline{\text{RESET}}$) executes a LOW to HIGH transition. During boot, the parallel port is set for 8-bit mode with the maximum number of WAIT states (seven). The master SPROC chip runs an internal program, stored in its control ROM, to upload its configuration from an external 8-bit EPROM into internal RAM. The master SPROC chip will then configure any slave SPROC chips present in the system. The EPROM will be selected by a HIGH on the master SPROC chip's chip select ($\overline{\text{CS}}$) pin, which is an output in master mode. Slave SPROC chips or memory-mapped peripherals will be selected by a LOW at this signal. In master mode, the value of the $\overline{\text{CS}}$ output is controlled by a bit set in the transmit mode register, which is the second byte of the parallel port mode register.

Watchdog Timer

The parallel port incorporates a simple watchdog timer circuit to prevent any undesirable lockup states in the interface. In both master and slave modes, a read or a write flag is set (in the parallel port status register) on the initiation of a read or write operation. This flag is reset on a successful completion of the operation. If, for some reason, the host controller hangs-up in slave mode, or an invalid condition occurs in master mode, the watchdog timer will detect the situation and clear the interface flags, allowing the next operation to be accepted and executed. The watchdog timer is fixed at 256 machine cycles (1280 master clock cycles).

The watchdog timer is enabled by setting bit 16 of the parallel port mode register. SPROC reset will disable the watchdog timer. If the watchdog timer is triggered, a flag is set in the parallel port status register.

Multiple I/O Lockout

If the parallel port is performing a read or write operation in master mode, and a second write or read operation is initiated before the first I/O operation is completed, the second I/O request is locked out. A lockout flag is set in the parallel port status register.

Input/Output Flags and Lines

The RTS and GPIO signals can be used for communication protocols between master and slave SPROC chips. These signals could be used as data-ready signals, requests for data, or microprocessor interrupt requests.

RTS[3:0] (request to send) are four pins that function as inputs for a master SPROC chip and as outputs for a slave SPROC chip. The RTS signals of a slave SPROC can be individually set or cleared via the parallel port, as described below.

GP[3:0] are four general purpose pins that are individually configurable as either inputs or outputs. During reset, when $\overline{\text{RESET}}$ is LOW, all GPIO signals are set up as inputs. In addition to being subject to internal program control, the configuration of each GP pin, and the value of each GPIO signal configured as an output, are also individually controllable via the parallel port.

Parallel Port Registers

The parallel port utilizes five memory-mapped registers for status and control functions. Tables 3-5 and 3-6 list the parallel port registers.

Table 3-5. Parallel Port Registers

REGISTER ADDRESS	REGISTER NAME	READ/WRITE
4FB	Lockout and watchdog flag clear	write
4FC	Parallel port status register	read
4FD	Parallel port input register	read
4FE	Parallel port GPIO/RTS control register	write
4FF	Parallel port mode register	write

Table 3-6. Parallel Port Register Bit Definitions

BIT	REGISTER 4FC	REGISTER 4FE	REGISTER 4FF
0	GP[0] INPUT	SET RTS[0]	RX MODE[0]
1	GP[1] INPUT	SET RTS[1]	RX MODE[1]
2	GP[2] INPUT	SET RTS[2]	RX MODE[2]
3	GP[3] INPUT	SET RTS[3]	RX WAIT STATES[0]
4	MODE[0]	CLEAR RTS[0]	RX WAIT STATES[1]
5	MODE[1]	CLEAR RTS[1]	RX WAIT STATES[2]
6	MODE[2]	CLEAR RTS[2]	RX STROBE DELAY
7	PARALLEL PORT BUSY FLAG	CLEAR RTS[3]	PARALLEL PORT SOFT RESET
8	LOCK OUT FLAG	SET GPIO[0]	$\overline{\text{CS}}$ (master mode only)
9	WATCHDOG FLAG	SET GPIO[1]	TX MODE[0]
10	READ FLAG	SET GPIO[2]	TX MODE[1]
11	WRITE FLAG	SET GPIO[3]	TX MODE[2]
12	RTS[0] INPUT	CLEAR GPIO[0]	TX WAIT STATES[0]
13	RTS[1] INPUT	CLEAR GPIO[1]	TX WAIT STATES[1]
14	RTS[2] INPUT	CLEAR GPIO[2]	TX WAIT STATES[2]
15	RTS[3] INPUT	CLEAR GPIO[3]	TX STROBE DELAY
16	N/A	OUTPUT GPIO[0]	N/A
17	N/A	OUTPUT GPIO[1]	N/A
18	N/A	OUTPUT GPIO[2]	N/A
19	N/A	OUTPUT GPIO[3]	N/A

Table 3-6. Parallel Port Register Bit Definitions (Continued)

BIT	REGISTER 4FC	REGISTER 4FE	REGISTER 4FF
20	NA/	INPUT GPIO[0]	N/A
21	N/A	INPUT GPIO[1]	N/A
22	N/A	INPUT GPIO[2]	N/A
23	N/A	INPUT GPIO[3]	N/A

The Parallel Port Status Register

The parallel port status register, a 16-bit register, contains signal values of selected SPROC chip pins and I/O status flags. This register is updated every machine cycle (5 master clock cycles). Bits 0 through 3 contain the current signal values at the GP pins, which could individually be configured either as inputs or outputs. Similarly, bits 12 through 15 contain the current values at the RTS pins, which are inputs for a master SPROC chip and outputs for a slave. Bits 4 through 6 contain the current values of the MODE configuration.

Parallel port status register bit 10 contains the read flag, which is set while the parallel port is performing a read operation. Similarly, bit 11 contains the write flag, which is set during a write operation. (For 8- and 16-bit modes, these flags remain set until the entire 24-bit data word has been transferred.)

Bit 7 is set while the parallel port is busy servicing an I/O transaction. Bit 8 is set if the parallel port is busy in master mode and another read or write request is received. The second request will be locked out and the lockout flag set. Bit 9 is set if the watchdog timer is enabled and it detects a timeout condition. Bits 8 and 9 can only be cleared by a SPROC reset or any write to the lockout and watchdog flag clear register.

The Watchdog/Lockout Flag Clear Register

Any write to this register clears watchdog and/or lockout flags set in the parallel port status register.

The Parallel Port Input Register (Master Mode Only)

The parallel port input register, a 24-bit register, holds the data word received during a read operation for subsequent storage at the destination address. This register also buffers and assembles the incoming data for 8- and16-bit modes. This register must be read by a GSP or the access port.

The Parallel Port GPIO/RTS Control Register

The parallel port GPIO/RTS Control register, a 24-bit register, is used to independently configure each GP pin as either an input or an output. It is also used to individually set and clear GP pins that are outputs, and slave SPROC chip RTS pins.

Each RTS or GPIO signal has a dedicated pair of SET and CLEAR bits in the parallel port GPIO/RTS control register. SET and CLEAR bits for RTS signals are in the low byte; SET and CLEAR bits for GPIO signals are in the mid byte. LOW values written to both SET and CLEAR bits results in no change to the associated signal. A HIGH value at the SET bit sets the associated signal HIGH. A HIGH value at the CLEAR bit sets the associated signal LOW. If a HIGH value is written to both SET and CLEAR bits, the CLEAR dominates.

Each GPIO signal additionally has a dedicated pair of OUTPUT and INPUT bits in the high byte of the parallel port GPIO/RTS control register to configure the signal as either an output or an input. LOW values written to both OUTPUT and INPUT bits results in no change to the associated signal. A HIGH value at the OUTPUT bit configures the associated GPIO signal as an output. A HIGH value at the INPUT bit configures the associated GPIO signal as an input. If a HIGH value is written to both OUTPUT and INPUT bits, the INPUT dominates.

The Parallel Port Mode Register

The master SPROC chip's parallel port mode register, a 16-bit register, controls the parallel port mode and timing.

When the master SPROC chip is reading from a slave SPROC chip or peripheral, bits 0 through 2 of the parallel port mode register (the RX MODE bits) are output at the master SPROC chip's MODE pins. Register bits 3 through 5 contain the number of WAIT states programmed for the read operation (i.e., they determine the duration of the read strobe LOW level generated by the master SPROC chip). The HIGH level between read strobes is 2 master clock cycles; this duration can be stretched to 5 master clock cycles for slower peripherals by setting bit 6 of the mode register (the RX strobe delay bit).

Similarly, when the master SPROC chip is writing to a slave SPROC chip or peripheral, bits 9 through 11 of the parallel port mode register (the TX MODE bits) are output at the master SPROC chip's MODE pins. Register bits 12 through 14 contain the number of WAIT states programmed for the write operation. The HIGH level between write strobes can be stretched for slower peripherals by setting bit 15 of the mode register (the TX strobe delay bit).

Bit 8 of the mode register is output at the master SPROC chips 's $\overline{\text{CS}}$ pin. A soft reset of the parallel port, which resets the interface flags and RTS lines (but not the GPIO or MODE signals), can be initiated by setting bit 7 of this register.

<u>Parallel Port Signals</u>

Table 3-7 lists the parallel port signals.

Table 3-7. Parallel Port Signal Definitions

SIGNAL	TYPE*	DESCRIPTION
ADDRESS[15:0]	O (M) I (S)	ADDRESS BUS
$\overline{\text{BUSGRANT}}$	I	BUS GRANT causes the SPROC chip to three-state the address and data buses, and MODE pins, when LOW.
$\overline{\text{BUSY}}$	O	PARALLEL PORT BUSY is set LOW when an I/O operation is occurring, set HIGH when completed. Also reset HIGH by watchdog timer if a timeout occurs.
$\overline{\text{CRESET}}$		Tied LOW.
$\overline{\text{CS}}$	O (M) I (S)	CHIP SELECT signal. A slave SPROC chip is selected by setting its $\overline{\text{CS}}$ input LOW. A master SPROC chip generates this signal as an output, expecting to select a slave SPROC chip by setting $\overline{\text{CS}}$ LOW, and an external ROM (containing every slave SPROC chip's configuration) by setting it HIGH.

Table 3-7. Parallel Port Signal Definitions (Continued)

SIGNAL	TYPE*	DESCRIPTION
DATA[23:0]	I/O	PARALLEL PORT DATA BUS -- 24-bit input/output/three-stateable bidirectional bus.
$\overline{\text{DTACK}}$	O	DATA TRANSFER ACKNOWLEDGE. In slave mode, set LOW by SPROC chip after $\overline{\text{RD}}$ or $\overline{\text{WR}}$ has gone LOW and the SPROC chip has completed the data transfer, set HIGH after $\overline{\text{RD}}$ or $\overline{\text{WR}}$ line goes HIGH. This output is always HIGH for a master SPROC chip.
EADDRESS[1:0]	O (M) I (S)	EXTENDED ADDRESS specifies which portion of the full 24-bit word is currently being transferred in 8- and 16-bit modes.
GP[3:0]	I/O	GENERAL PURPOSE I/O lines, individually configurable as either input or output. Can be used to interface SPROC chips with each other or with an external controller as data-ready, microprocessor interrupt requests, etc. Controlled and configured by a write to parallel port GPIO/RTS control register.
MASTER	I	MASTER causes SPROC chip to operate in master mode when HIGH, and in slave mode when LOW.
MODE[2:0]	O (M) I (S)	MODE[0] differentiates between full 24-bit mode (HIGH) and partial (8- or 16-bit) modes (LOW). MODE[1] differentiates between 8-bit mode (HIGH) and 16-bit mode (LOW) for partial data transfers. MODE[2] specifies whether the first 8- or 16-bit transmission contains the lsb (HIGH) or the msb (LOW).
$\overline{\text{RED}}$		Tied LOW.
$\overline{\text{RD}}$	O (M) I (S)	READ strobe generated by master SPROC chip or external controller. A LOW value on $\overline{\text{RD}}$ initiates a READ operation. $\overline{\text{RD}}$ must remain LOW long enough to successfully complete the READ; programmed WAIT states or $\overline{\text{DTACK}}$ handshaking may be utilized for this purpose. Data latches at the destination when $\overline{\text{RD}}$ returns HIGH.

Table 3-7. Parallel Port Signal Definitions (Continued)

SIGNAL	TYPE*	DESCRIPTION
$\overline{\text{RESET}}$	I	$\overline{\text{RESET}}$ must be held LOW for a minimum of 25 master clock cycles after power and clock have stabilized. This input is a Schmitt trigger type which is suitable for use with an RC time constant to provide power-on reset. While $\overline{\text{RESET}}$ is LOW, a master mode SPROC chip will force address, extended address, and SPROC select address LOW, while driving $\overline{\text{CS}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ HIGH. Slave SPROC chips connected to the bus will then be deselected and have driven inputs. MODE[2:0] will be configured for 8-bit boot mode with msb byte first and zero WAIT states. The data bus will be driven.
RTS[3:0]	I (M) O (S)	RTS REQUEST TO SEND flags. These pins are outputs for slave SPROC chips and inputs for master SPROC chips. Can be used to interface slave with master or external controller as data-ready, microprocessor interrupt requests, etc. Controlled and configured by write to parallel port GPIO/RTS control register.
$\overline{\text{WR}}$	O (M) I (S)	WRITE strobe generated by master SPROC chip or external controller. A LOW value on $\overline{\text{WR}}$ initiates a WRITE operation. $\overline{\text{WR}}$ must remain LOW long enough to successfully complete the WRITE; programmed WAIT states or $\overline{\text{DTACK}}$ handshaking may be utilized for this purpose. Data latches at the destination when $\overline{\text{WR}}$ returns HIGH.

* (M) = master mode, (S) = slave mode, I = input, O = output

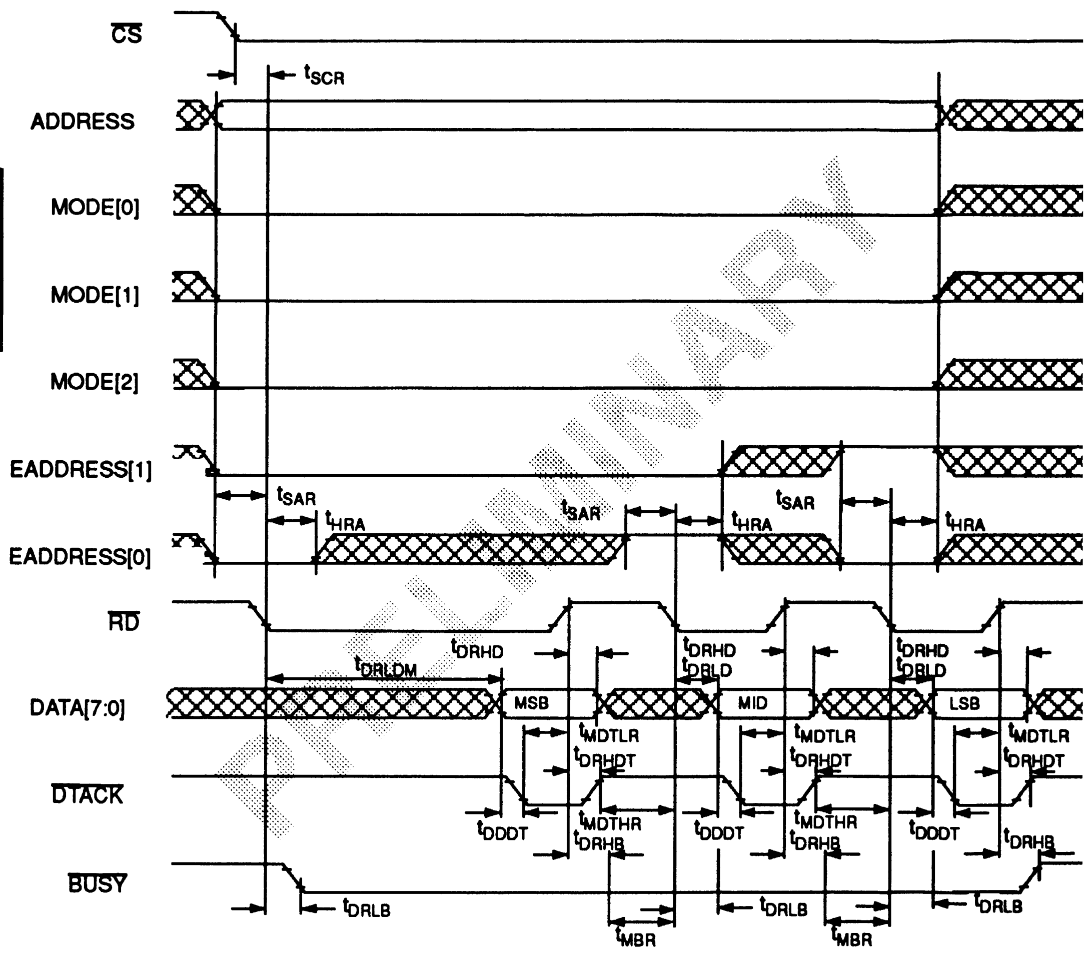
Parallel Port Timing

Table 3-8. Timing for External Controller Read from Slave SPROC Chip

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
Setup and Hold Times					
t_{DRHD}	DATA hold after $\overline{RD}$ ↑	20		ns	
t_{DRLD}	Delay time: DATA valid after $\overline{RD}$ ↓ for second byte/word and third byte of 8-bit and 16-bit modes (MODE[0] = LOW)	10	20	ns	
t_{DRLDM}	Delay time: DATA valid after $\overline{RD}$ ↓ for 24-bit mode and first byte/word of 8-bit and 16-bit modes (RAM access)	5	10	t_{CYK}	
t_{HRA}	Hold time: ADDRESS after $\overline{RD}$ ↓	0		ns	
t_{HRC}	Hold time: $\overline{CS}$ ↑ after $\overline{RD}$ ↑	0		ns	
t_{SAR}	Setup time: ADDRESS to $\overline{RD}$ ↓	10		ns	
t_{SCR}	Setup time: $\overline{CS}$ ↓ to $\overline{RD}$ ↓	0		ns	
Read Strobe					
t_{WRH}	Minimum pulse width $\overline{RD}$ HIGH	20		ns	
t_{WRL}	Minimum pulse width $\overline{RD}$ LOW for last two bytes and second word of 8-bit and 16-bit modes (MODE[0] = LOW)	20		ns	

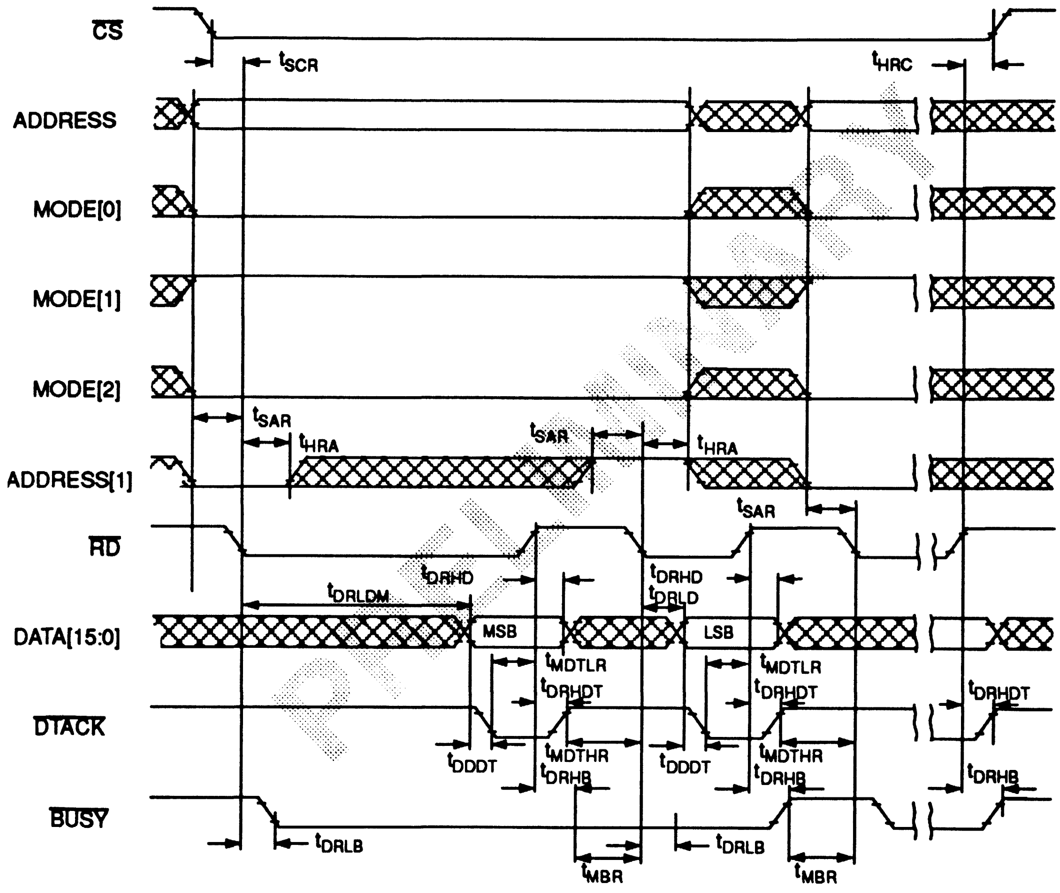
Table 3-8. Timing for External Controller Read from Slave SPROC Chip (Continued)

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{WRLM}	Minimum pulse width $\overline{RD}$ LOW for 24-bit mode and first byte/word of 8- and 16-bit modes (RAM access)	10		t_{CYK}	
t_{DDDT}	Delay time: Data valid to $\overline{DTACK}$ ↓	0	10	ns	
t_{DRHB}	Delay time: $\overline{RD}$ ↑ to $\overline{BUSY}$ ↑	20	30	ns	
t_{DRHDT}	Delay time: $\overline{RD}$ ↑ to $\overline{DTACK}$ ↑	10	20	ns	
t_{DRLB}	Delay time: $\overline{RD}$ ↓ to $\overline{BUSY}$ ↓	10	20	ns	
t_{MBR}	Minimum time from $\overline{BUSY}$ ↑ to $\overline{RD}$ ↓	20		ns	
t_{MDTHR}	Minimum time from $\overline{DTACK}$ ↑ to $\overline{RD}$ ↓	20		ns	
t_{MDTLR}	Minimum time from $\overline{DTACK}$ ↓ to $\overline{WR}$ ↑	0		ns	



NOTE: For lsb-first operation, complement MODE[2] and EADDRESS inputs.

Figure 3-4. External Controller Read from Slave SPROC Chip: 8-bit Mode



NOTE: For lsb-first operation, complement MODE[2] and EADDRESS[1] inputs.

Figure 3-5. External Controller Read from a Slave SPROC Chip: 16-bit Mode

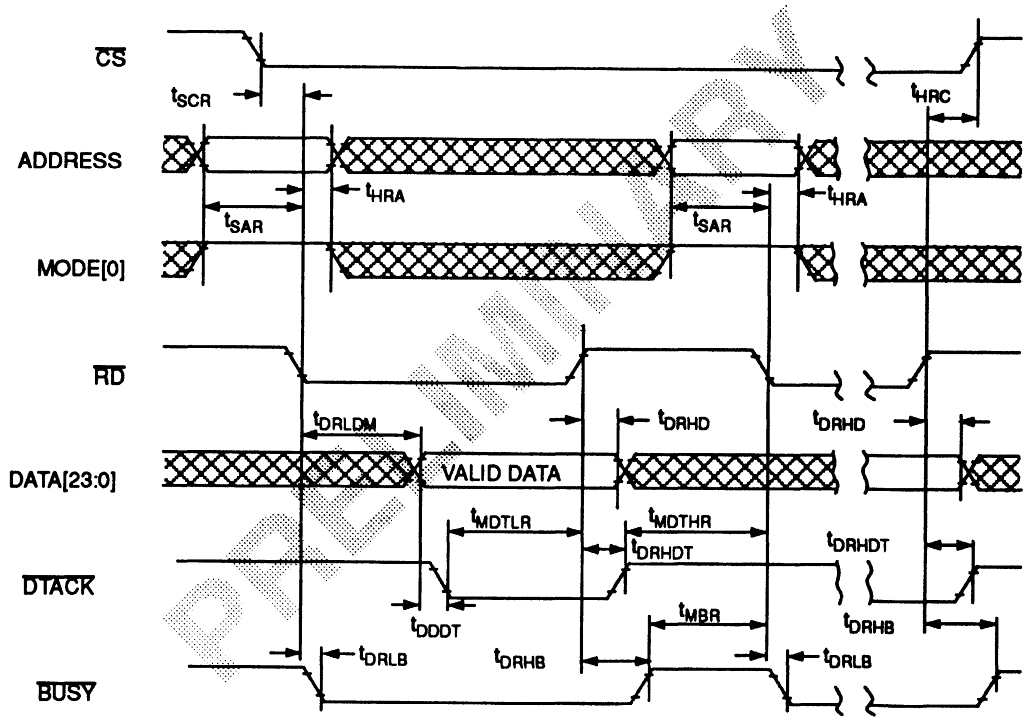


Figure 3-6. External Controller Read from Slave SPROC Chip: 24-bit Mode

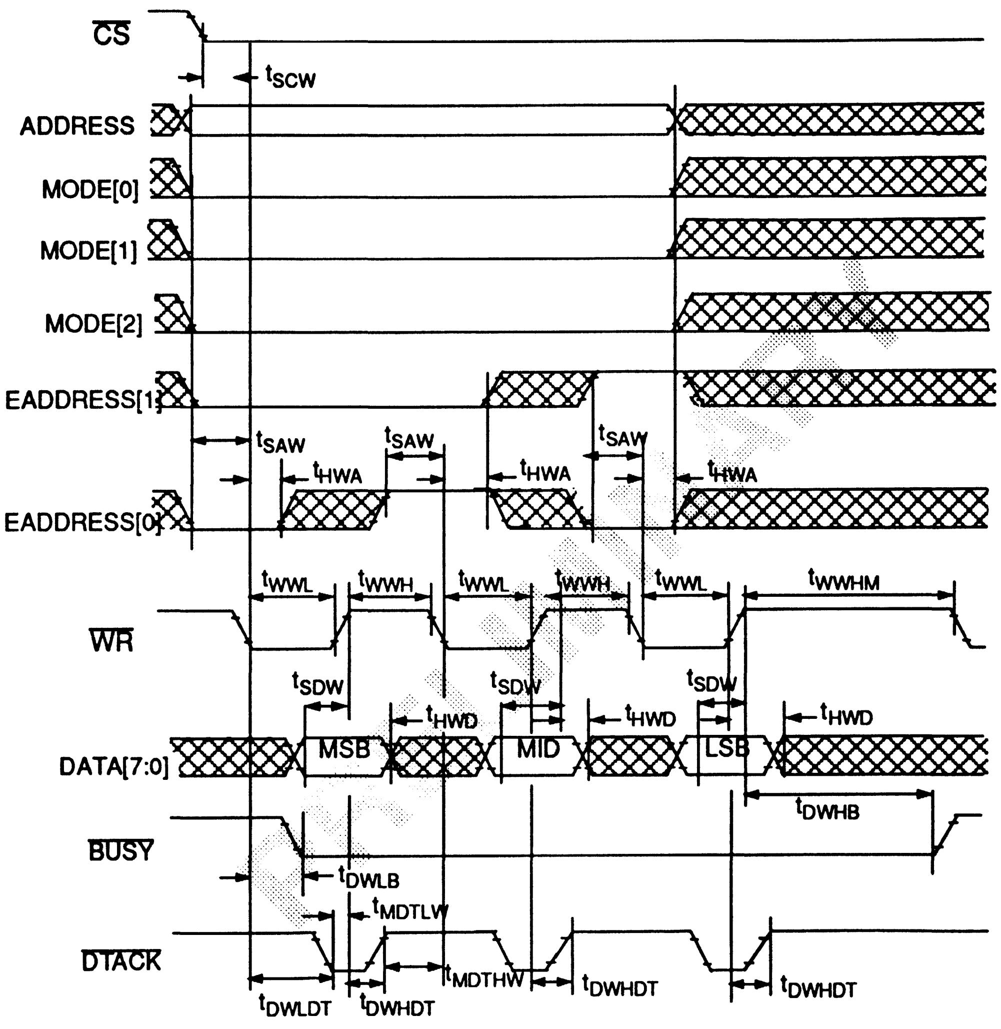
Table 3-9. Timing for External Controller Write to Slave SPROC Chip

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
Setup and Hold Times					
t_{HWA}	Hold time: ADDRESS after $\overline{WR}$ ↓	0		ns	
t_{HWC}	Hold time: $\overline{CS}$ ↑ after $\overline{WR}$ ↑	0		ns	
t_{HWD}	Hold time: DATA after $\overline{WR}$ ↑	0		ns	
t_{SAW}	Setup time: ADDRESS to $\overline{WR}$ ↓	40		ns	
t_{SCW}	Setup time: $\overline{CS}$ ↓ to $\overline{WR}$ ↓	0		ns	
t_{SDW}	Setup time: DATA to $\overline{WR}$ ↑	20		ns	
Write Strobe					
t_{WWH}	Minimum pulse width $\overline{WR}$ HIGH for 24-bit mode and last byte/word of 8-bit and 16-bit modes (RAM access)	10		t_{CYK}	
t_{WWHM}	Minimum pulse width $\overline{WR}$ HIGH for first two bytes and first word of 8-bit and 16-bit modes (MODE[0] = LOW)	20		ns	
t_{WWL}	Minimum pulse width $\overline{WR}$ LOW	20		ns	
t_{DWHB}	Delay time: $\overline{WR}$ ↑ to $\overline{BUSY}$ ↑	5	10	t_{CYK}	(See note below)
		10	20	ns	

Table 3-9. Timing for External Controller Write to Slave SPROC Chip (Continued)

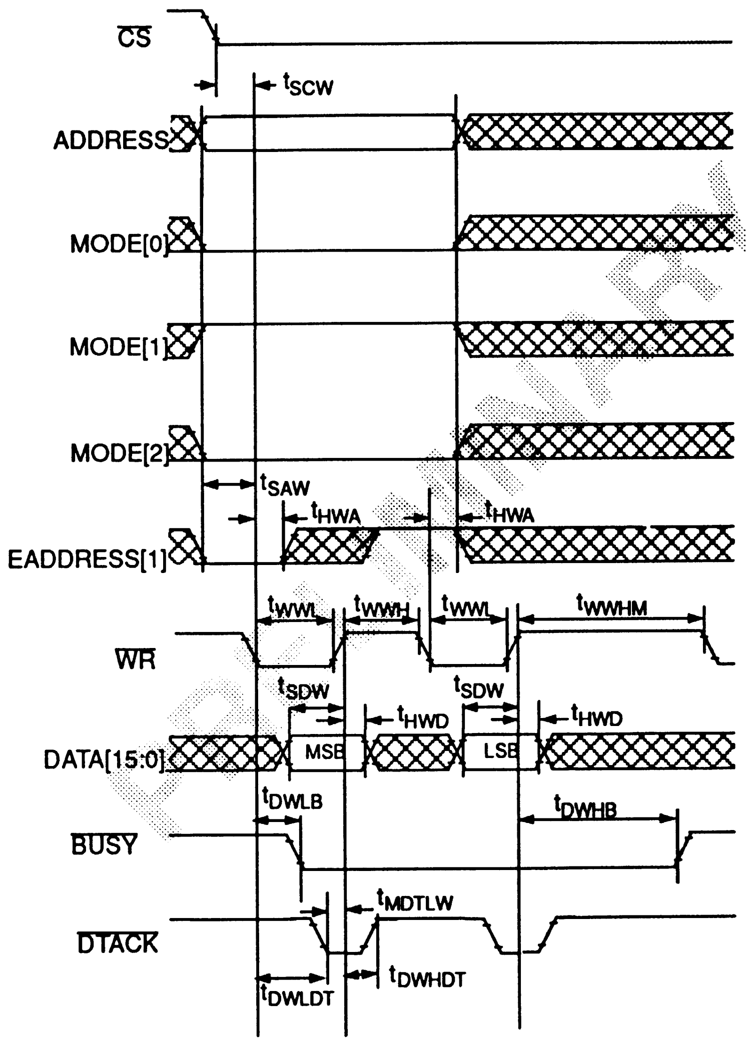
SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{DWHDT}	Delay time: $\overline{WR}$ ↑ to $\overline{DTACK}$ ↑	10	20	ns	
t_{DWLB}	Delay time: $\overline{WR}$ ↓ to $\overline{BUSY}$ ↓	10	20	ns	
t_{DWLDT}	Delay time: $\overline{WR}$ ↓ to $\overline{DTACK}$ ↓	5	10	t_{CYK}	(See note below)
		10	20	ns	
t_{MBW}	Minimum time from $\overline{BUSY}$ ↑ to $\overline{WR}$ ↓	20		ns	
t_{MDTHW}	Minimum time from $\overline{DTACK}$ ↑ to $\overline{WR}$ ↓	10		t_{CYK}	(See note below)
		20		ns	
t_{MDTLW}	Minimum time from $\overline{DTACK}$ ↓ to $\overline{WR}$ ↑	0		ns	

NOTE: Top duration applies when an actual write to RAM occurs. Bottom duration applies when MODE[0] is LOW and the first or second byte in 8-bit mode, or the first word in 16-bit mode, is transferred prior to the actual write to RAM.



NOTE: For lsb-first operation, complement MODE[2] and EADDRESS[1] inputs.

Figure 3-7. External Controller Write to Slave SPROC Chip: 8-bit Mode



NOTE: For lsb-first operation, complement MODE[2] and EADDRESS[1] inputs.

Figure 3-8. External Controller Write to Slave SPROC Chip: 16-bit Mode

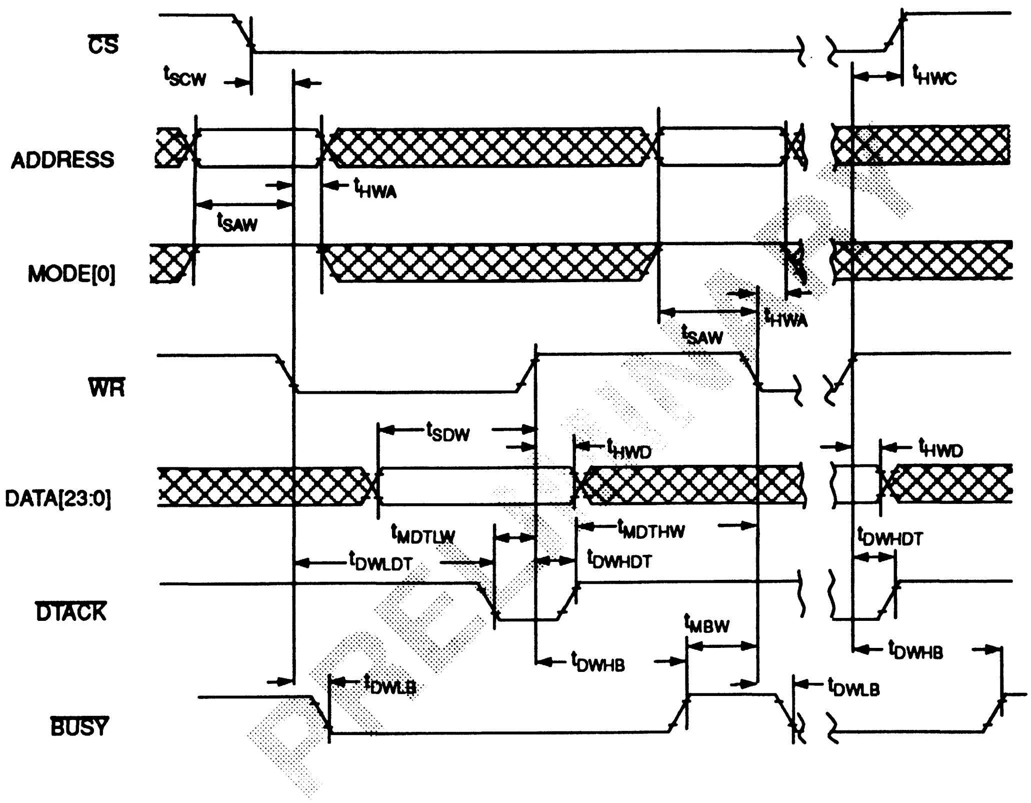


Figure 3-9. External Controller Write to Slave SPROC Chip: 24-bit Mode

The Serial Interface

Overview

The chip serial interface provides four independently programmable serial ports; two input ports and two output ports. Each port consists of four lines: ***data, clock, strobe,*** and *sync*. The strobe line functions as a "data valid" signal, and the sync line, when raised HIGH, associates the serial data with the beginning of an internal data buffer for inputs, or with the end of a FIFO for outputs.

The following port options are programmable:

1. Data width -- 8-, 12-, 16-, or 24-bits of data per word.
2. Order of data -- either most significant bit (msb) or least significant bit (lsb) first.
3. Short, long, or stereo strobe -- a short strobe is a pulse at the active strobe level that is one serial clock cycle wide, occurring prior to the first valid data bit. A long strobe remains at the active strobe level during the entire interval bounded by the first and last valid data bits for the given data width. A stereo strobe differentiates left and right channel data at a common serial port.
4. Gated or continuous clock -- in gated clock mode, a burst of clock pulses occurs only when data is valid, and clock is quiescent otherwise.
5. Clock source -- can be either internally generated by the SPROC chip or externally supplied. The internal clock is obtained from the master clock via the following formula

$$SC_f = \frac{MC_f}{4(256 - (r))}$$

   where SC_f is the serial clock frequency, MC_f is the master clock frequency, and r is the value of the appropriate serial port clock rate select register.

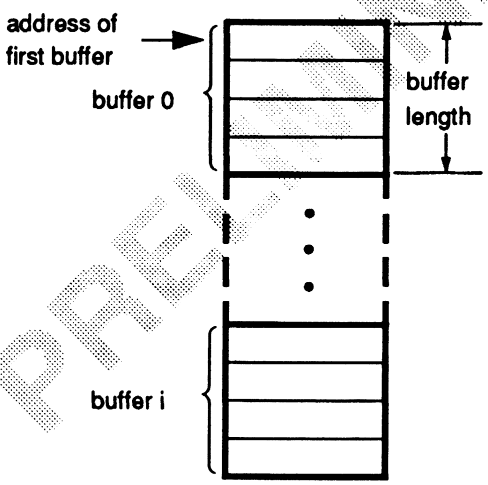
6. Clock polarity -- data transitions are aligned with the positive edge of the serial clock for non-inverted clock, or the negative edge for inverted clock.

7. Active strobe level -- specifiable as either HIGH for non-inverted strobe, or LOW for inverted strobe.

8. Inverted msb -- for offset binary format.

Short and long strobes are associated with single channel data. Stereo strobes are associated with time-multiplexed dual channel data, where consecutive data words are differentiated by complementary strobe levels.

Each serial port is independently serviced by a dedicated data flow manager that organizes and stores data in and retrieves data from a circular FIFO storage area within the CMU. For input ports, the FIFO consists of equal length buffers whose size and index (number) are programmable.



Input ports with short or long strobes support a maximum of 255 buffers and a maximum buffer length of 511 words. Input ports with stereo strobes require that there be exactly two buffers, each buffer containing two words.

An output port FIFO contains a single buffer whose maximum size is programmable. For ports with stereo strobes, left and right channel data are output alternately.

For input ports, a HIGH on the sync input will force the current serial data word, when assembled, to be stored at the beginning of a FIFO buffer. If, at that time, the data word would have been stored at the beginning of a buffer anyway, the sync signal is ignored. Otherwise, the sync HIGH forces this data to be stored at the beginning of the first buffer. The duration of the HIGH sync pulse for input ports with short or long strobes must be at least 1.5 master clock cycles. No alignment with data or clock is necessary. Input ports with stereo strobes require that the sync and strobe inputs be tied together.

For output ports, the sync signal is an output pin whose HIGH value signals FIFO completion to other devices. The minimum guaranteed duration of the HIGH pulse is 1.5 master clock cycles.

On a write to the soft reset register, the data flow manager initializes all registers that define each port's configuration by reloading them from RAM, and clears all counters. The FIFO will start at its initial location.

Serial Port Configuration Registers

Each serial port's configuration and status, including its FIFO organization, is governed by seven consecutive memory-mapped registers, designated addr0 through addr6. Their widths and functions are shown in Tables 3-10 and 3-11. The configuration registers for serial input ports siport0 and siport1 each occupy seven addresses. The registers for serial output ports soport0 and soport1 also occupy seven addresses each.

Table 3-10. Serial Port Configuration Registers

REGISTER	INPUT		OUTPUT	
	BIT WIDTH	FUNCTION	BIT WIDTH	FUNCTION
addr0	9	FIFO buffer length	N/A	N/A
addr1	8	FIFO index length	12	FIFO length
addr2	12	FIFO start address	12	Same as input
addr3	11	Port setup register, see definition below		
addr4	N/A	N/A	8	Decimation register
addr5	N/A	N/A	1	Run register
addr6	N/A	N/A	24	Wait trigger mask

Table 3-11. Port Setup Register (addr3) Bit Definitions

BIT	FUNCTION		
0,1	Sets data width		
	bit1	bit0	data width
	0	0	24-bits
	0	1	16-bits
	1	0	12-bits
	1	1	8-bits
	HIGH		LOW
2	msb first		lsb first
3	short strobe		long strobe
4	gated clock		continuous clock

Table 3-11. Port Setup Register (addr3) Bit Definitions (Continued)

BIT	FUNCTION			
5	internal clock		external clock	
6	inverted clock*		non-inverted clock*	
7	inverted strobe		non-inverted strobe	
8	inverted msb		non-inverted msb	
9,10	Sets I/O format			
	bit 10	bit 9	Input	Output
	0	0	Use strobe defined by bit 3	
	0	1	Use format 2**	
	1	0	Use format 3**	reserved
	1	1	reserved	Use format 3**

* inverted clock -- data changes on negative edge
non-inverted clock -- data changes on positive edge
**stereo-strobe formats

The independent port attributes supported by register addr3 allow definition of a wide variety of serial interfaces. A specific configuration is matched by selecting the corresponding options. For example, compatibility with stereo format 2 is obtained by configuring 16-bit data, msb first, continuous clock, non-inverted clock, non-inverted strobe, non-inverted msb, and format 2 stereo strobe (transition at same clock edge as first valid data bit). Similarly, compatibility with stereo format 3 is obtained by configuring 16-bit data, msb first, continuous clock, inverted clock, inverted strobe, non-inverted msb, and format 3 stereo strobe (transition one clock cycle prior to first valid data bit).

The SPROClab development system automatically generates the register values corresponding to user-supplied serial port options. The registers are then configured during boot by the external controller. Subsequent reconfiguration can also be performed.

Serial Input Ports

The SPROC chip pins for the two input ports are:

- data -- SRXD[0, 1]
- clock -- SRXCLK[0, 1]
- strobe -- SRXSTROB[0, 1]
- sync -- SNCPULSI[0, 1]

The serial data, SRXD, enters a bidirectional shift register, with shift clock SRXCLK, for proper alignment of either msb- or lsb-first formats.When the trailing edge of a long strobe occurs at SRXSTROB, or when the number of data bits following the leading edge of a stereo strobe or the trailing edge of a short strobe equals the data width, shifting stops and the data is latched into a holding register. Data for 8-, 12-, and 16-bit formats are left-justified, and right-padded with zeros.

For all strobe types, consecutive data words (whether single-channel or stereo) are written to consecutive words of the same FIFO buffer, until the buffer is filled.

When the end of a buffer is reached, a write to a dedicated write-trigger address informs the GSPs that a full buffer has been input and is available for processing. The serial interface will then begin to fill the next FIFO buffer. After the last FIFO buffer has been filled, the cycle starts again from the first buffer.

Table 3-12 lists the dedicated write-trigger addresses for the serial input ports.

Table 3-12. Serial Input Port Write Trigger Addresses

HEX ADDRESS	SERIAL PORT
800	serial input port siport0
801	serial input port siport1

If the buffer length, as contained in configuration register addr0 is zero, the data flow manager is considered disabled and the port data will be written to a preset memory address.

Serial Output Ports

The SPROC chip pins for the two output ports are:

- data – STXD[0, 1]
- clock -- STXCLK[0, 1]
- strobe -- STXSTROB[0, 1]
- sync -- SNCPULSO[0, 1]
- trigger -- COMPUTE[3:0]

The parallel data from the output FIFO buffer is latched into a holding register during the host slot of the SPROC chip machine cycle. It is then loaded into a bidirectional shift register for either msb- or lsb-first serial output at STXD, with shift clock STXCLK.

In order to initiate serial output, the port must be put into "running mode" by setting bit 23 of the run register (addr5). This is normally done by a GSP when there is sufficient data in the output FIFO.

In order to synchronize output, the 24-bit wait trigger mask (addr6) is loaded into a mask register. Twenty bits of the mask register, bits 0 through 19, can be reset to 0 by the occurrence of internal preprogrammed events; for $0 \leq i \leq 13$ hex, bit i is cleared by writing any value to memory location (800+i) hex. The four remaining register positions, bits 20 through 23, are cleared to 0 by positive transitions at inputs COMPUTE[0] through COMPUTE[3], respectively, to allow external control of output synchronization. When all bits are cleared, a pulse is generated to allow output of data, depending on the contents of the decimating register.

The 8-bit decimation register (addr4) provides a mechanism for selecting output rates, and is useful for decimating systems. This register activates divide-by-N reduction, where N is the numeric contents of the decimation register; the mask register is loaded and cleared N times before data is output.

When the last word of the output FIFO buffer is about to be output, a HIGH pulse 4 master clock cycles wide is output at SNCPULSO. After the last data word has been completely output, the data flow manager reinitializes the serial output port (counters are cleared and registers are reloaded).

Serial Interface Signals

Table 3-13 lists the serial interface signals.

Table 3-13. Serial Interface Signals

SIGNAL	TYPE	DESCRIPTION
COMPUTE[3:0]	I	COMPUTE inputs allow synchronization of internal events for the two serial output ports. Positive transitions at these inputs clear bits 20 through 23 of the output port mask registers. Each output port's mask register must be entirely cleared before the port will output data.
SRXD	I	SERIAL INPUT DATA can be 8-, 12-, 16-, or 24-bit words, msb- or lsb-first, synchronized to positive or negative clock edges, inverted msb (for offset binary format) or twos-complement format.
SRXCLK	I/O	SERIAL INPUT CLOCK can be externally supplied or generated by the SPROC chip. Clock can be gated or continuous, inverted (data changes on negative edge) or non-inverted (data changes on positive edge).
SRXSTROB	I	SERIAL INPUT STROBE flags valid data. Short and long strobes, associated with single-channel data, can have an inverted (data valid when LOW) or non-inverted (data valid when HIGH) active level. A short strobe is a pulse at the active level one serial clock cycle wide prior to the first valid data bit. A long strobe remains at the active level from the first valid data bit through the last. Stereo strobes are associated with dual-channel data, and alternate in level to distinguish data from each channel.

Table 3-13. Serial Interface Signals (Continued)

SIGNAL	TYPE	DESCRIPTION
SNCPULSI	I	SERIAL INPUT SYNCHRONIZING pulse synchronizes input FIFO buffer to incoming data. When HIGH, forces incoming data to be written to the first word of the first buffer. If a new buffer was about to be started when the sync pulse occurs, the sync pulse is ignored. For short and long strobes, the pulse must be at least 4 master clock cycles wide. For stereo strobes, this input pin should be tied to SRXSTROB.
SNCPULSO	O	SERIAL OUTPUT SYNCHRONIZING pulse signals that the end of the output FIFO buffer has been reached. For all strobe modes, a HIGH pulse at least 4 master clock cycles wide is output at the start of the last serial output word.
STXD	O	SERIAL OUTPUT DATA can be 8-, 12-, 16-, or 24-bit words, msb- or lsb-first, synchronized to positive or negative clock edges, inverted msb (for offset binary format) or twos-complement format. To accommodate slower peripherals, outputting each data word can be synchronized. Decimation allows further reduction of output rate by requiring that the events occur the configured number of times before enabling output.
STXCLK	I/O	SERIAL OUTPUT CLOCK can be externally supplied or generated by the SPROC chip. Clock can be gated or continuous, inverted (data changes on negative edge) or non-inverted (data changes on positive edge).

Table 3-13. Serial Interface Signals (Continued)

SIGNAL	TYPE	DESCRIPTION
STXSTROB	O	SERIAL OUTPUT STROBE flags valid data. Short and long strobes, associated with single-channel data, can have an inverted (data valid when LOW) or non-inverted (data valid when HIGH) active level. A short strobe is a pulse at the active level one serial clock cycle wide prior to the first valid data bit. A long strobe remains at the active level from the first valid data bit through the last. Stereo strobes are associated with dual-channel data, and alternate in level to distinguish data from each channel. A stereo strobe executes a transition either at (format 2), or one serial clock cycle prior to (format 3), the first valid data bit, and remains quiescent until data for the other channel is to be output.

Serial Interface Timing

Table 3-14. Serial Output Port Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{DTCD}	Delay time: active STXCLK edge to valid STXD data	0	30	ns	
t_{DTCSLF}	Delay time: active STXCLK edge to fall of long strobe at STXSTROB	0	20	ns	
t_{DTCSLR}	Delay time: active STXCLK edge to rise of long strobe at STXSTROB	0	20	ns	
t_{DTCSSF}	Delay time: active STXCLK edge to fall of short strobe at STXSTROB	0	20	ns	

Table 3-14. Serial Output Port Timing (Continued)

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{DTCSSR}	Delay time: active STXCLK edge to rise of short stobe at STXSTROB	0	20	ns	
t_{WTC}	Serial clock STXCLK period	4(256-(r))/MC_f			(See note below)
		2		t_{CYK}	
t_{WTCH}	Serial clock STXCLK high time	0.4	0.6	t_{WTC}	
t_{WTSNC}	Output sync pulse high time at SNCPULSO	4		t_{CYK}	

NOTE: Top duration applies when internal serial clock is selected. Bottom duration applies when external serial clock is selected.

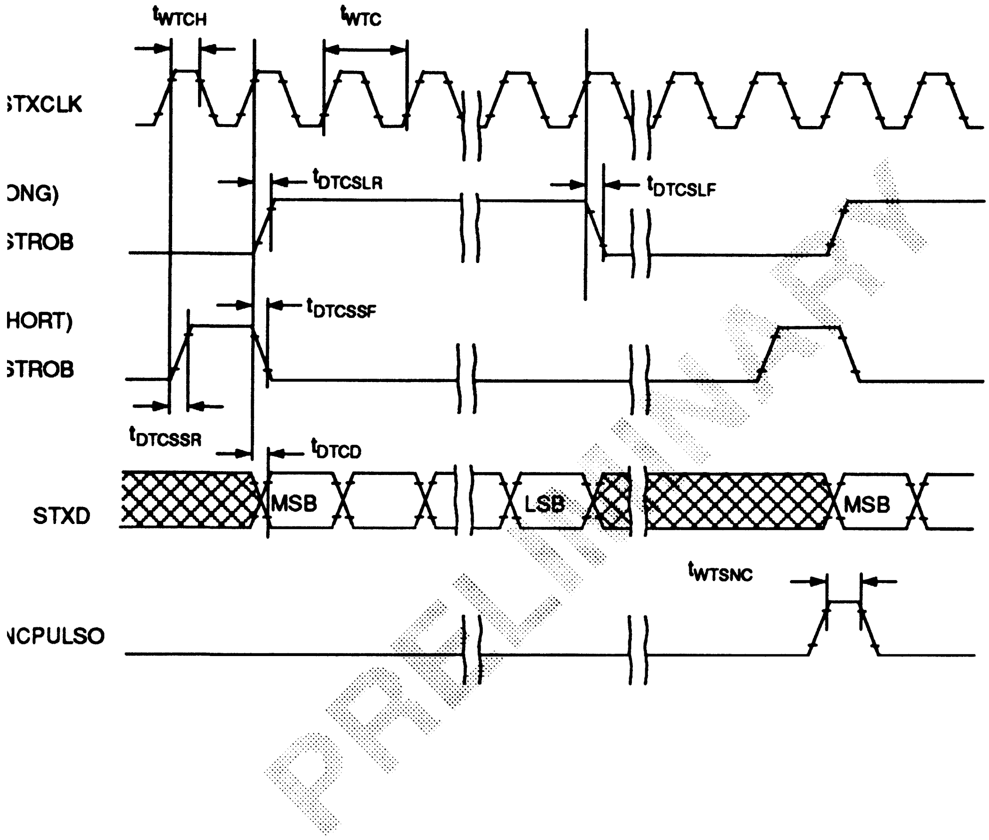


Figure 3-10. Serial Output Port Timing

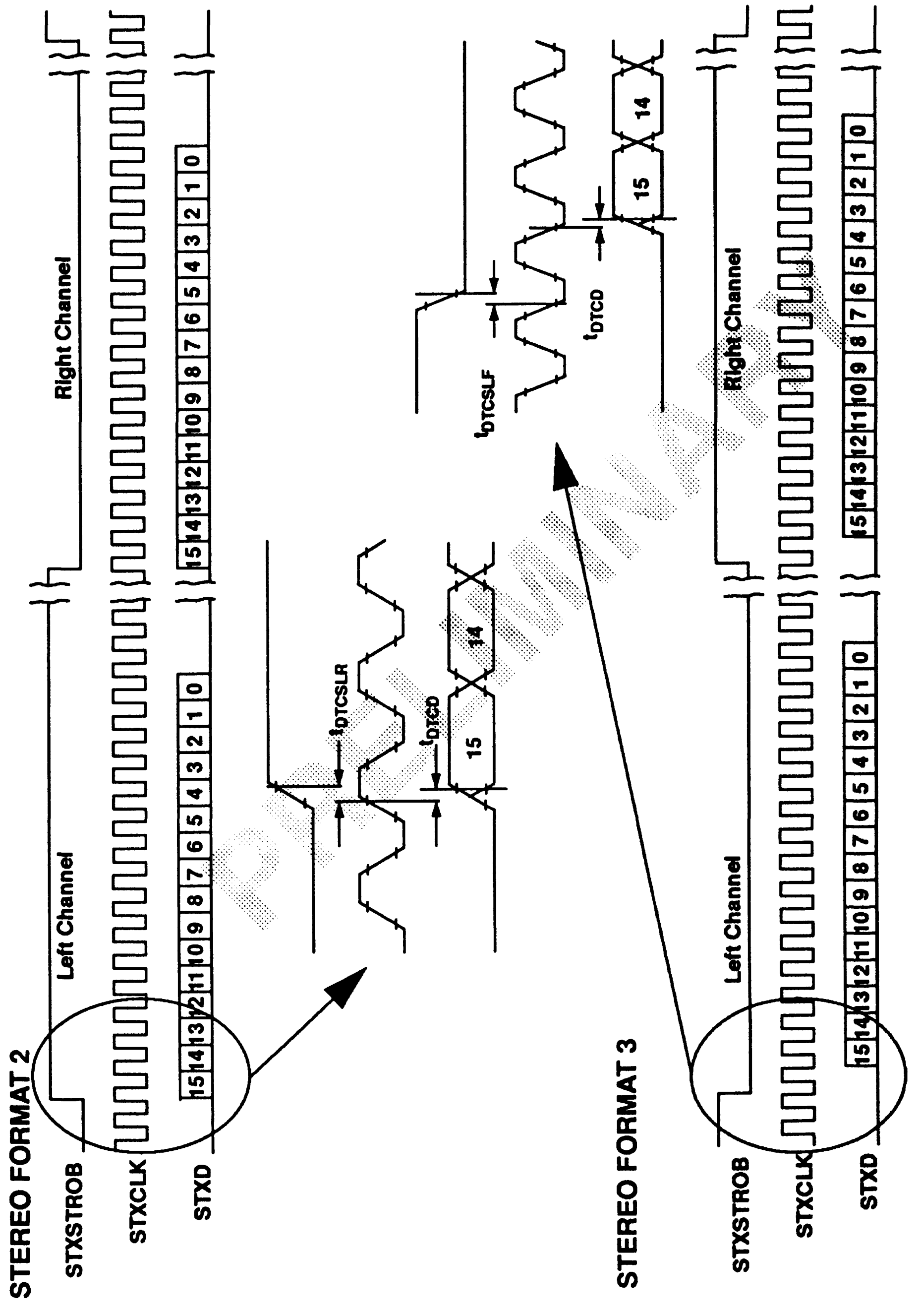
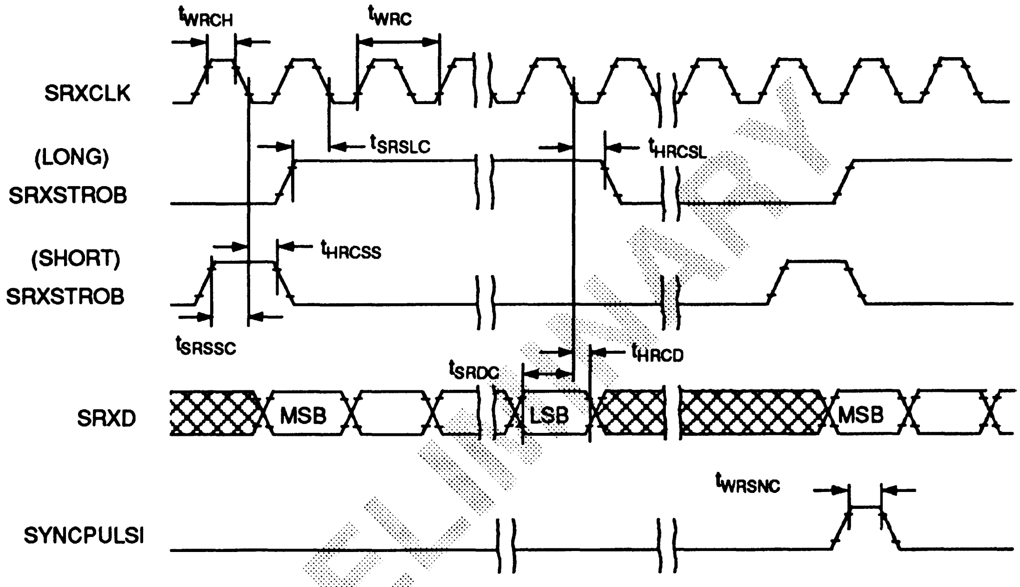


Figure 3-11. Serial Output Timing: Stereo Formats

Table 3-15. Serial Input Port Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{HRCD}	Hold time: valid SRXD data after inactive SRXCLK edge	10	30	ns	
t_{HRCSL}	Hold time: fall of long strobe at SRXSTROB after inactive SRXCLK edge	10	20	ns	
t_{HRCSS}	Hold time: fall of short strobe at SRXSTROB after inactive SRXCLK edge	10	20	ns	
t_{SRDC}	Setup time: valid SRXD data before inactive SRXCLK edge	20		ns	
t_{SRSLC}	Setup time: rise of long strobe at SRXSTROB before inactive SRXCLK edge	20		ns	
t_{SRSSC}	Setup time: rise of short strobe at SRXSTROB before inactive SRXCLK edge	20		ns	
t_{WRC}	Serial clock SRXCLK period	4(256-(r))/MC$_f$			(See note below)
		2		t_{CYK}	
t_{WRCH}	Serial clock SRXCLK high time	0.4	0.6	t_{RWC}	
t_{WRSNC}	Input sync pulse high time at SNCPULSI	1.5		t_{CYK}	

NOTE: Top duration applies when internal serial clock is selected. Bottom duration applies when external serial clock is selected.



NOTES:
1. Clocks and strobes are shown non-inverted.
2. Although timing is shown for both long and short strobes, these options are mutually exclusive.

Figure 3-12. Serial Input Port Timing

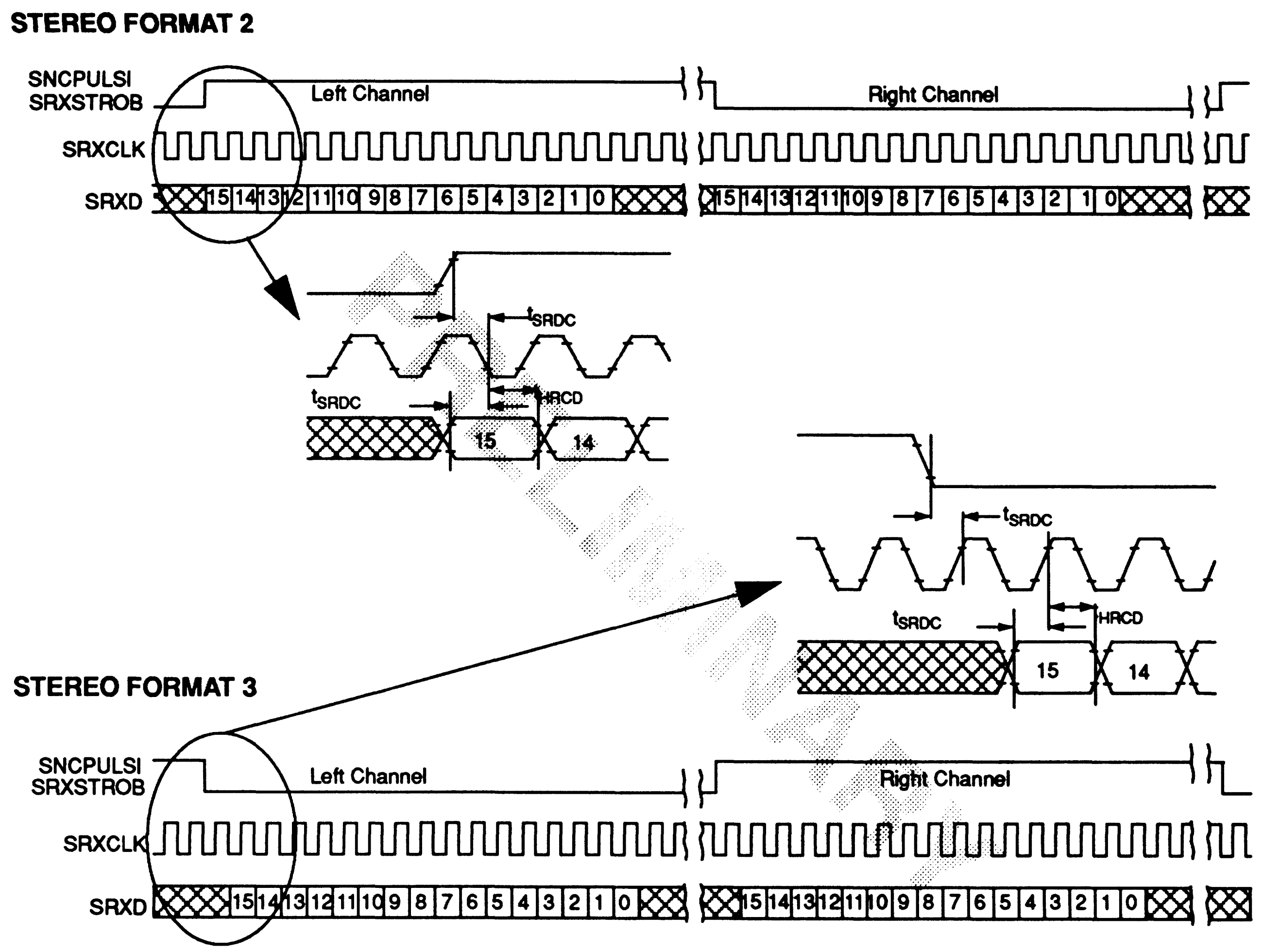


Figure 3-13. Serial Input Timing: Stereo Formats

The Access Port

Overview

The access port provides a serial interface for dynamically modifying and observing the contents of internal SPROC chip memory, without any hardware changes or interruption to processing. This port is useful for debugging a design, for interactively modifying the design, and for testing SPROC-based implementations.

The SPROClab development supports interactive design debugging and verification via the SPROCbox interface unit and the SPROC chip's access port. This system allows the user to probe selected signals, modify parameters and code, and reconfigure the SPROC chip, if necessary, until the desired functionality is obtained. The user exercises this control from the workstation console, without requiring detailed knowledge of the compiled SPROC chip program or the access port timing.

After the design phase has been completed, the access port can also be used to optimize or adjust the implemented SPROC-based design while it is operating within its targeted environment.

Access Port Interface

The SPROC chip access port pins are:

- clock -- ACLOCK
- data in -- ARXD
- strobe in -- ARXSTR
- data out -- ATXD
- strobe out -- ATXSTR

ACLOCK is an externally-supplied clock that provides the access port timing for both input and output pins. Input data transitions at ARXD must be aligned with the rising edge of ACLOCK. Output data is always aligned with this clock edge. The maximum ACLOCK frequency is 40% of the master clock frequency.

The first bit of the serial data input, ARXD, is a control bit that specifies the operation; HIGH for a read and LOW for a write. The next 15 bits of the serial input stream specify an extended memory location; the first three bits are reserved for future expansion and must all be LOW, while the remaining bits specify an internal SPROC chip address, with most significant bit first.

For a read operation, 16-bit serial input causes the contents of the specified memory location to be asynchronously read and then output at ATXD, most significant bit first. The 24-bit output strobe, ATXSTR, remains HIGH while valid data is being shifted out at ATXD.

For a write operation, the last address bit should be immediately followed by the 24-bit data word to be written to the specified memory location, most significant bit first. When all 40 serial bits have been received, the data word is asynchronously written to this address.

The input strobe, ARXSTR, will remain HIGH during the 16 valid bits of a read request and the 40 valid bits of a write request.

Access Port Signal Table

Table 3-16 lists the access port signals.

Table 3-16. Access Port Signals

SIGNAL	TYPE	DESCRIPTION
ACLOCK	I	Externally-supplied clock providing the access port timing for both input and output pins. Maximum frequency is 40% of the master clock frequency.
ARXD	I	SERIAL INPUT (RECEIVE) DATA specifying a READ or WRITE operation, aligned with the rising edge of ACLOCK. For a READ, the serial input data consists of 16 bits. The first bit is HIGH, bits 2 through 4 are LOW, and bits 5 through 16 specify an internal SPROC chip address, with most significant bit first. For a WRITE, the serial input data consists of 40 bits. Bits1 through 4 are LOW, bits 5 through 16 specify an internal SPROC chip address, with most significant bit first, and bits 16 through 40 specify the word to be written to this address, most significant bit first.
ARXSTR	I	SERIAL INPUT STROB. The strobe must be aligned with the rising edge of ACLOCK. and remain HIGH while valid data is at ARXD (16 ACLOCK cycles for a READ and 40 cycles for a WRITE).
ATXD	O	SERIAL OUTPUT (TRANSMIT) DATA in response to a READ operation. The 24-bit word is shifted out most significant bit first, aligned with the rising edge of ACLOCK.
ATXSTR	O	SERIAL OUTPUT STROBE. ATXSTR remains HIGH while valid data is at ATXD. It is aligned with the rising edge of ACLOCK.

Access Port Timing

Table 3-17. Access Port Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{DATCD}	Delay time: ACLOCK ↑ to valid ATXD data	0	30	ns	
t_{DATCSF}	Delay time: ACLOCK ↑ to fall of ATXSTR strobe	0	20	ns	
t_{DATCSR}	Delay time: ACLOCK ↑ to rise of ATXSTR strobe	0	20	ns	
t_{HARCD}	Hold time: valid ARXD data after ACLOCK ↓	20		ns	
t_{HARCS}	Hold time: fall of ARXSTR after ACLOCK ↓	20		ns	
t_{SARDC}	Setup time: valid ARXD data before ACLOCK ↓	20		ns	
t_{SARSC}	Setup time: ARXSTR rise before ACLOCK ↓	20		ns	
t_{WAC}	Access port clock ACLOCK period	2.5		t_{CYK}	
t_{WACH}	ACLOCK high time	0.4	0.6	t_{WAC}	

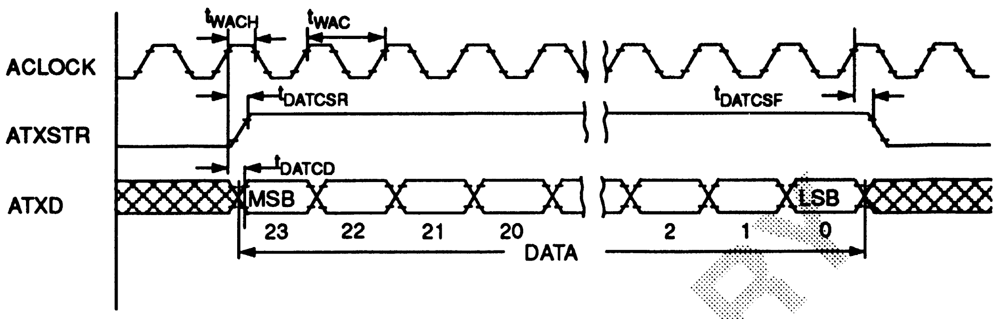


Figure 3-14. Access Port Output Timing

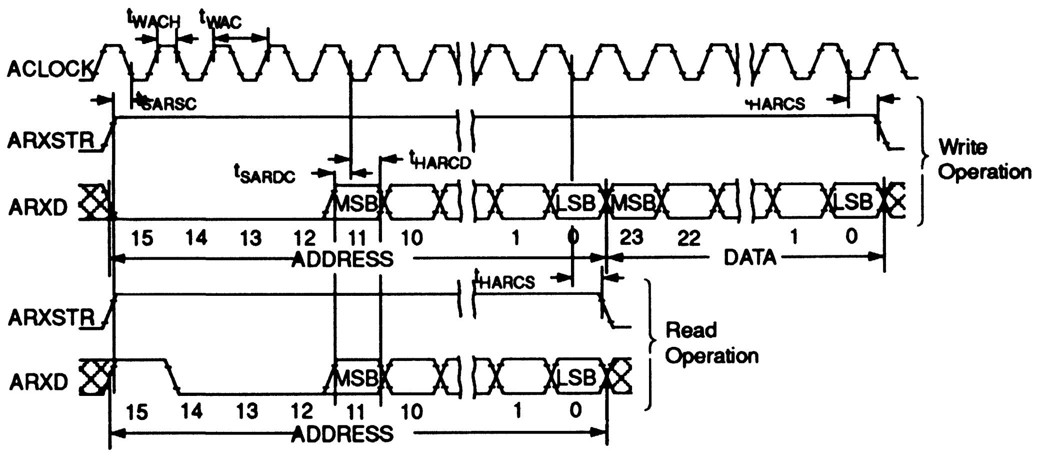


Figure 3-15. Access Port Input Timing

The Probe Port

Overview

The probe port provides a digital interface for dynamically observing the values of selected nodes while the SPROC chip is in operation, without requiring any interruptions or hardware changes. This port is useful for interactive design debugging and verification. The probed node's value is output in digital representation.

The SPROClab development system supports interactive probing via the SPROCbox interface unit and the SPROC chip's probe port. The user exercises this control from the workstation console, without requiring detailed knowledge of the compiled SPROC chip program or the probe port configuration and timing.

Functionally, the probe port contains three major subsections:

- input match circuitry
- a data flow manager
- gain adjustment circuitry

The address of the node to be probed is loaded into an internal address register of the input match circuitry. On each internal write strobe, the address register's contents are compared with the internal SPROC chip RAM address bus (except when the probe port is disabled). Whenever a match occurs, the corresponding data on the data bus is latched and entered into the port's output buffer.

The port's data flow manager is essentially identical to that of a serial port; most configuration options for a serial port are also available for the probe port.

The gain adjustment circuitry selects the bit range of the output node values. An internal gain register is loaded with a number between 0 and 15, inclusive, which specifies the number of times that the node's value must be shifted left (multiplied by 2) prior to being output.

Operation and Configuration

While the development system provides a convenient and easy-to-use interface for debugging designs, it may sometimes be desirable to monitor node values when the SPROC chip is operating in its targeted environment. This section provides the information necessary to support this debugging mode.

The probe port pins are:

- clock -- PROBCLOCK
- data out (digital) -- PROBDATA
- strobe -- PROBSTROBE

PROBCLOCK, the clock signal, is always generated by the SPROC chip. Its frequency is 25% of the master clock frequency.

The probe port is enabled simply by configuring a FIFO with non-zero buffer length.

The address of the node to be probed is input, via the parallel port, to the internal address match register. Whenever a new node is selected for probing in this manner, the data flow manager is completely reinitialized (registers are reloaded, counters are cleared, and FIFO pointer reset to its start address).

On an internal write, when the address on the internal SPROC chip RAM address bus agrees with the contents of the address match register, the data flow manager writes the corresponding RAM data bus value to the next available FIFO buffer location. When the FIFO is half filled, the port's run register is automatically set, and the buffered node values are output at a rate determined by the configured wait trigger mask (and its corresponding event occurrences, including positive transitions at COMPUTE[3:0]) and decimation count. Each node value sample is left-shifted by the amount stored in the gain register prior to being output at PROBDATA.

The probe port configuration registers are essentially identical to those of serial ports (addr0 through addr6). One exception is the probe port run register (addr5), which cannot be externally written to -- it is automatically set when the FIFO is half-filled.

Since the data flow manager fills the FIFO buffer as well as outputs its stored values, both input and output sections must be configured. The registers for the input section are in the contiguous memory-mapped region beginning at hex address 480. Similarly, the registers for the output section begin at hex address 488.

Since the input and output FIFOs occupy the same physical memory locations, the input and output sections must be configured consistently:

- product of buffer length (addr0) and index (addr1) of input section must equal FIFO length of output section (addr1), and
- the FIFO start address (addr2) in both sections must be identical.

Probe Port Configuration Registers

Tables 3-18 and 3-19 describe the seven registers, addr0 through addr6, that define the configuration of the probe port.

Table 3-18. Probe Port Configuration Registers

REGISTER	INPUT		OUTPUT	
	BIT WIDTH	FUNCTION	BIT WIDTH	FUNCTION
addr0	9	FIFO buffer length	N/A	N/A
addr1	4	FIFO index length	12	FIFO length
addr2	12	FIFO start address	12	Same as input
addr3	12	Match register	9	Port setup register
addr4	4	Gain register	8	Decimation register
addr5	N/A	N/A	1	Run register
addr6	N/A	N/A	24	Wait trigger mask

Table 3-19. Port Setup Register (addr3) Bit Definitions

BIT	FUNCTION		
0,1	Sets data width		
	bit1	bit0	data width
	0	0	24-bits
	0	1	16-bits
	1	0	12-bits
	1	1	8-bits
	HIGH		LOW
2	msb first		lsb first
3	short strobe		long strobe
4	gated clock		continuous clock
5	reserved		
6	inverted clock*		non-inverted clock*
7	inverted strobe		non-inverted strobe
8	inverted msb		non-inverted msb

* inverted clock -- data changes on negative edge
non-inverted clock -- data changes on positive edge

Probe Port Signal Table

Table 3-20 lists the probe port signals.

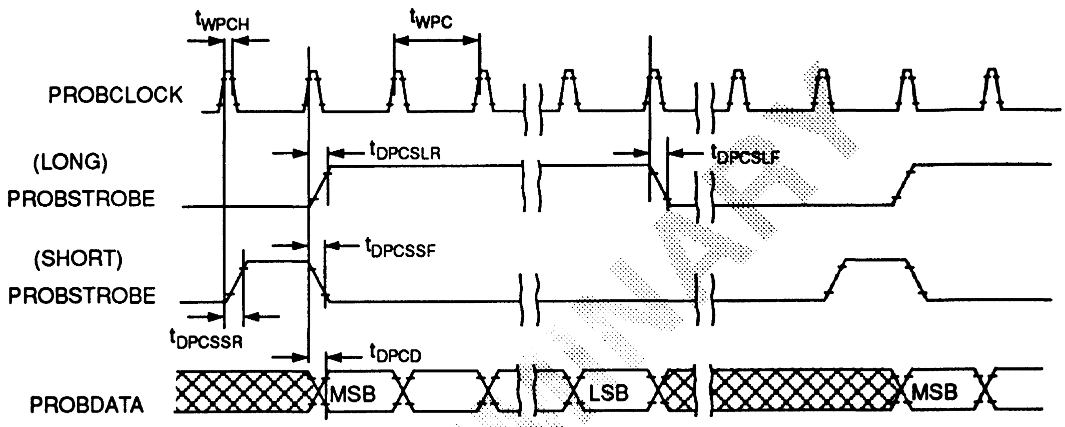
Table 3-20. Probe Port Signals

SIGNAL	TYPE	DESCRIPTION
PROBCLOCK	O	SPROC CHIP-SUPPLIED CLOCK provides the timing for the serial output port (master clock frequency divided by 4). Clock can be gated or continuous, inverted (data changes on negative edge) or non-inverted (data changes on positive edge).
PROBDATA	O	SERIAL OUTPUT DATA can be 8-, 12-, 16-, or 24-bit words, msb- or lsb-first, synchronized to positive or negative clock edges, inverted msb (for offset binary format) or twos-complement format.
PROBSTROBE	O	SERIAL OUTPUT STROBE flags valid data. Short and long strobes can have an inverted (data valid when LOW) or non-inverted (data valid when HIGH) active level. A short strobe is a pulse at the active level one serial clock cycle wide prior to the first valid data bit. A long strobe remains at the active level from the first valid data bit through the last.

Probe Port Timing

Table 3-21. Probe Port Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{DPCD}	Delay time: active PROBCLOCK edge to valid PROBDATA data	0	30	ns	
t_{DPCSLF}	Delay time: active PROBCLOCK edge to long strobe fall at PROBSTROBE	0	20	ns	
t_{DPCLSR}	Delay time: active PROBCLOCK edge to long strobe rise at PROBSTROBE	0	20	ns	
t_{DPCSSF}	Delay time: active PROBCLOCK edge to short strobe fall at PROBSTROBE	0	20	ns	
t_{DPCSSR}	Delay time: active PROBCLOCK edge to short strobe rise at PROBSTROBE	0	20	ns	
t_{WPC}	Serial clock PROBCLOCK period	4		t_{CYK}	
t_{WPCH}	Serial clock PROBCLOCK high time	0.5		t_{CYK}	



NOTES:
1. Clocks and strobes are shown non-inverted.
2. Although timing is shown for both long and short strobes, these options are mutually exclusive.

Figure 3-16. Probe Port Timing

Electrical Specifications

Basic Specifications

Table 3-22. Absolute Maximum Ratings

SPECIFICATION	ABBREVIATION	RATING
Power supply voltage range	V_{DD}	-0.5V to 7.0V
Input voltage range	V_I	-0.5V to 7.0V
Output current	I_O	10mA
Operating temperature range	T_{OPT}	0°C to 70°C
Storage temperature range	T_{STG}	-65°C to 150°C

Table 3-23. DC Electrical Specifications*

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{IL}	input low voltage	0	0.8	V	
V_{IH}	input high voltage	2.0	V_{DD}	V	
V_{OL}	output low voltage		0.4	V	I_{OL} = 2mA
V_{OH}	output high voltage	V_{DD} - 0.4		V	I_{OH} = -400µA
I_{IL}	input leakage current		10	µA	$0 \le V_I \le V_{DD}$
I_{OL}	output leakage current		10	µA	$0 \le V_I \le V_{DD}$
I_{DD}	supply current		300	mA	50 MHz

* T_A = 0°C to 70°C, V_{DD} = 5V +/- 5%

Table 3-24. Capacitance*

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
C_I	input capacitance		10	pF	(See note below)
C_O	output capacitance		10	pF	(See note below)
C_{IO}	input/output capacitance		15	pF	(See note below)
NOTE: f_c = 1MHz, unmeasured pins at 0V.					

* $T_A = 25^\circ C$

General Timing Specifications

Table 3-25. Master Clock Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{CYK}	clock cycle time	50		ns	SPROC 14xx-5x
t_{KF}	clock fall time	0	5	ns	
t_{KR}	clock rise time	0	5	ns	
t_{WKH}	clock pulse width high	$(t_{cyk(min)} / 2)$ - $(t_{WKH} + t_{WKL} / 2)$		ns	
t_{WKL}	clock pulse width low	$(t_{cyk(min)} / 2)$ - $(t_{WKH} + t_{WKL} / 2)$	0	ns	

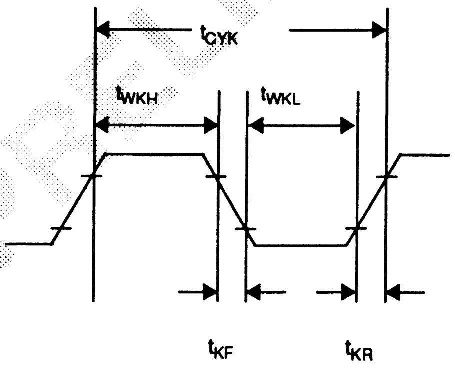


Figure 3-17. Master Clock Timing

Table 3-26. Input Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{IF}	input fall time		100	ns	
t_{IR}	input rise time		100	ns	

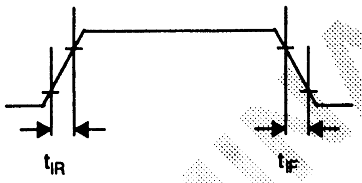


Figure 3-18. Input Timing

Table 3-27. Reset Timing

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
t_{RST}	reset pulse width	25		t_{CYK}	
t_{REC}	reset recovery time	200		ns	

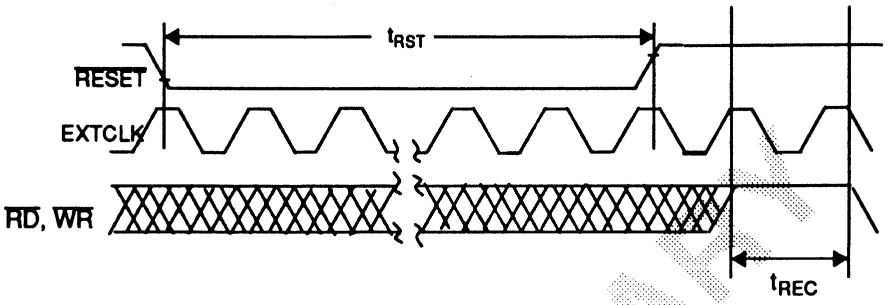
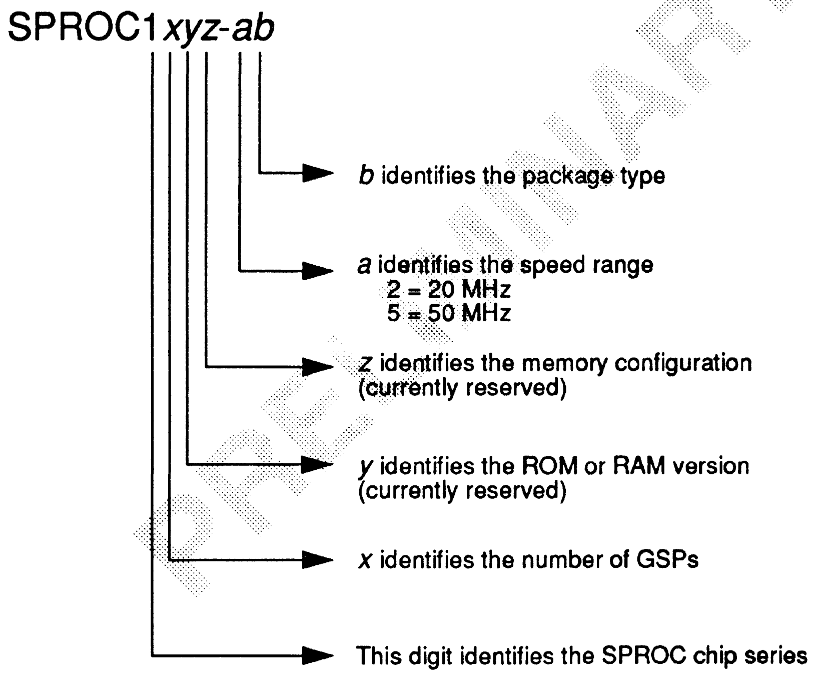


Figure 3-19. Reset Timing

Additional Specifications

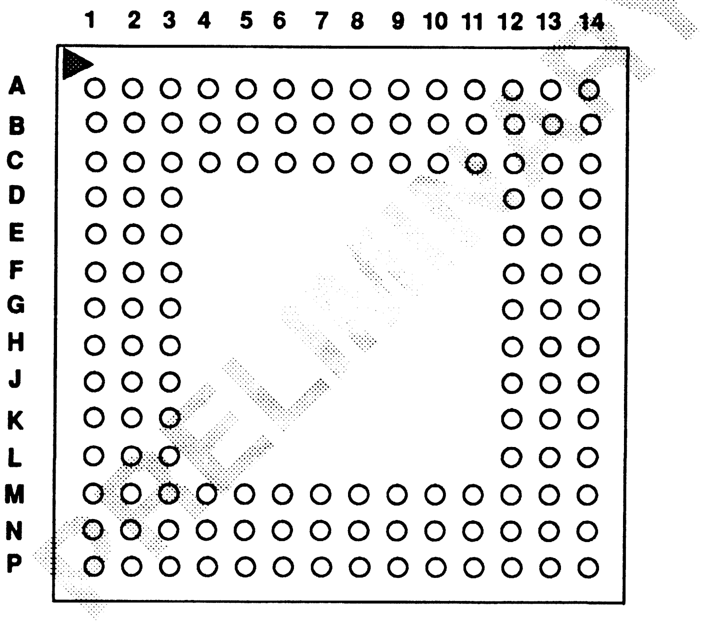
Chip Naming Conventions

The following conventions are used to determine part numbers for SPROC-1400 digital signal processors:


SPROC1*xyz-ab*
b identifies the package type
a identifies the speed range
2 = 20 MHz
5 = 50 MHz
z identifies the memory configuration (currently reserved)
y identifies the ROM or RAM version (currently reserved)
x identifies the number of GSPs
This digit identifies the SPROC chip series

Pin Configuration

Figure 3-20 and Table 3-28 detail the pinout of the SPROC-1400 digital signal processor.



Top View

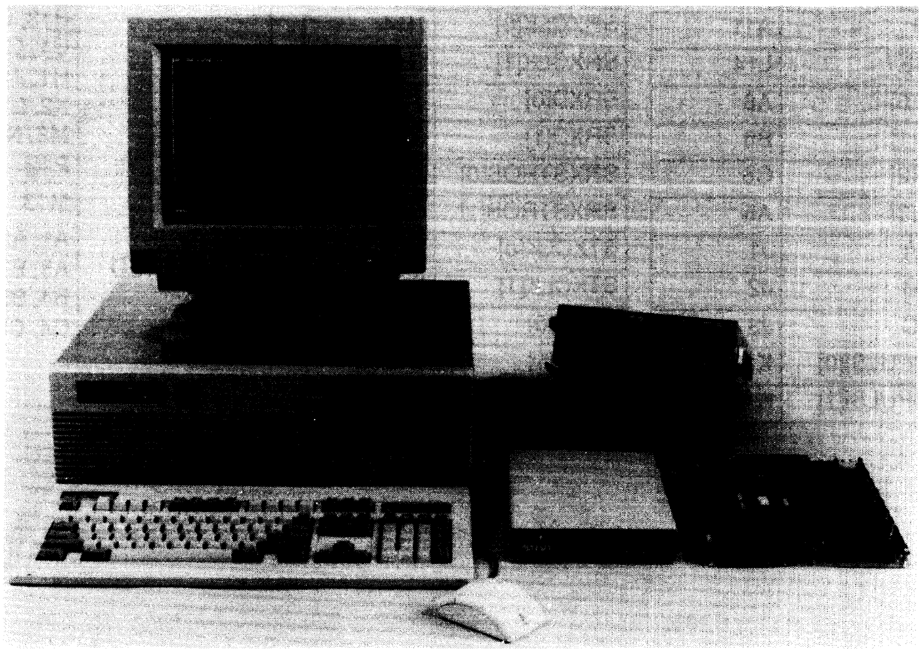
Figure 3-20. Pin Configuration for 132 Ceramic Package

Table 3-28. Pin Designations

SIGNAL	PIN
ACLOCK	B10
ADDRESS[0]	N10
ADDRESS[1]	P11
ADDRESS[2]	P10
ADDRESS[3]	M9
ADDRESS[4]	N9
ADDRESS[5]	P9
ADDRESS[6]	P8
ADDRESS[7]	N8
ADDRESS[8]	M8
ADDRESS[9]	M7
ADDRESS[10]	N7
ADDRESS[11]	P7
ADDRESS[12]	P5
ADDRESS[13]	N5
ADDRESS[14]	M5
ADDRESS[15]	P4
ARXD	A10
ARXSTR	C9
ATXD	B9
ATXSTR	A9
$\overline{\text{BUSGRANT}}$	L12
$\overline{\text{BUSY}}$	P14
COMPUTE[0]	A13
COMPUTE[1]	B12
COMPUTE[2]	C11
COMPUTE[3]	A12
CRESET	C12
$\overline{\text{CS}}$	N14
DATA[0]	B1
DATA[1]	C2
DATA[2]	D3
DATA[3]	C1
DATA[4]	D2
DATA[5]	D1
DATA[6]	E3
DATA[7]	E2
DATA[8]	E1
DATA[9]	F3
DATA[10]	F2
DATA[11]	F1
DATA[12]	M1
DATA[13]	L2
DATA[14]	K3
DATA[15]	N1
DATA[16]	M2
DATA[17]	L3
DATA[18]	P1
DATA[19]	M3
DATA[20]	P2
DATA[21]	N3
DATA[22]	M4
DATA[23]	P3
$\overline{\text{DTACK}}$	M12
EADDRESS[0]	M11
EADDRESS[1]	N12
EXTCLK	H3
EXTINTBCLK	G3
GND	B2, B7, B11, C3, D13, F13, H2, H12, K1, K12, L13, N2, N6, N13, P6
GP[0]	A8
GP[1]	B8
GP[2]	C8
GP[3]	C7
MASTER	M14
MODE[0]	P13
MODE[1]	M10
MODE[2]	N11
PROBCLOCK	D14
PROBDATA	E13

Table 3-28. Pin Designations (Continued)

SIGNAL	PIN
PROBSTROBE	C14
$\overline{\text{RD}}$	C10
RED	A11
RESET	L14
RTS[0]	A6
RTS[1]	B6
RTS[2]	C6
RTS[3]	A5
SELA	J1
SELB	J2
SELC	J3
SNCPULSI[0]	K13
SNCPULSI[1]	K14
SNCPULSO[0]	E14
SNCPULSO[1]	F12
SRXCLK[0]	H14
SRXCLK[1]	G14
SRXD[0]	J13
SRXD[1]	G12
SRXSTROB[0]	J14
SRXSTROB[1]	G13
STXCLK[0]	H13
STXCLK[1]	F14
STXD[0]	B14
STXD[1]	D12
STXSTROB[0]	C13
STXSTROB[1]	A14
VDD	A4, A7, B13, E12, G1, G2, H1, J12, K2, L1, M6, N4, P12
WR	M13
NOT CONNECTED	A1, A2, A3, B3, B4, B5, C4, C5



SPROClab Overview

SPROC chips are supported by the PC-based SPROClab development system that allows a designer to directly program the devices graphically by creating system-level block diagrams constructed from a library of familiar analog and digital signal-processing functions. In this fashion, even the most complicated ssoftware can be implemented within a few hours. Using SPROCdrive interface software, the development system supports interactive probing, debugging, and modification of SPROC devices running in target systems.

The SPROClab development system includes all the tools that an engineer needs to program a SPROC chip. It consists of a user supplied workstation (currently an IBM PC) that runs STAR's proprietary ssoftware and is coupled, via an RS232 port, to the SPROC development interface (SPROCbox). The SPROCbox contains the interface circuitry required to connect the emulator to the development workstation and target system. The development system itself is based on proprietary partitioning and optimizing algorithms that:

- have been developed specifically for signal processing;
- automatically control the partitioning and balancing of processing loads among multiple processors using unique signal-analysis and allocation algorithms;
- enable the generation of highly efficient SPROC machine code from a system block diagram; and
- interact with aspects of the SPROC architecture to support real-time parametric control in systems requiring self-adaptive signal processing.

The SPROC signal processing tools convert system block diagrams to short (2 - 4K bytes) files that contain the SPROC machine code for the system. The conversion process typically takes minutes, enabling a powerful iterative, interactive design process.

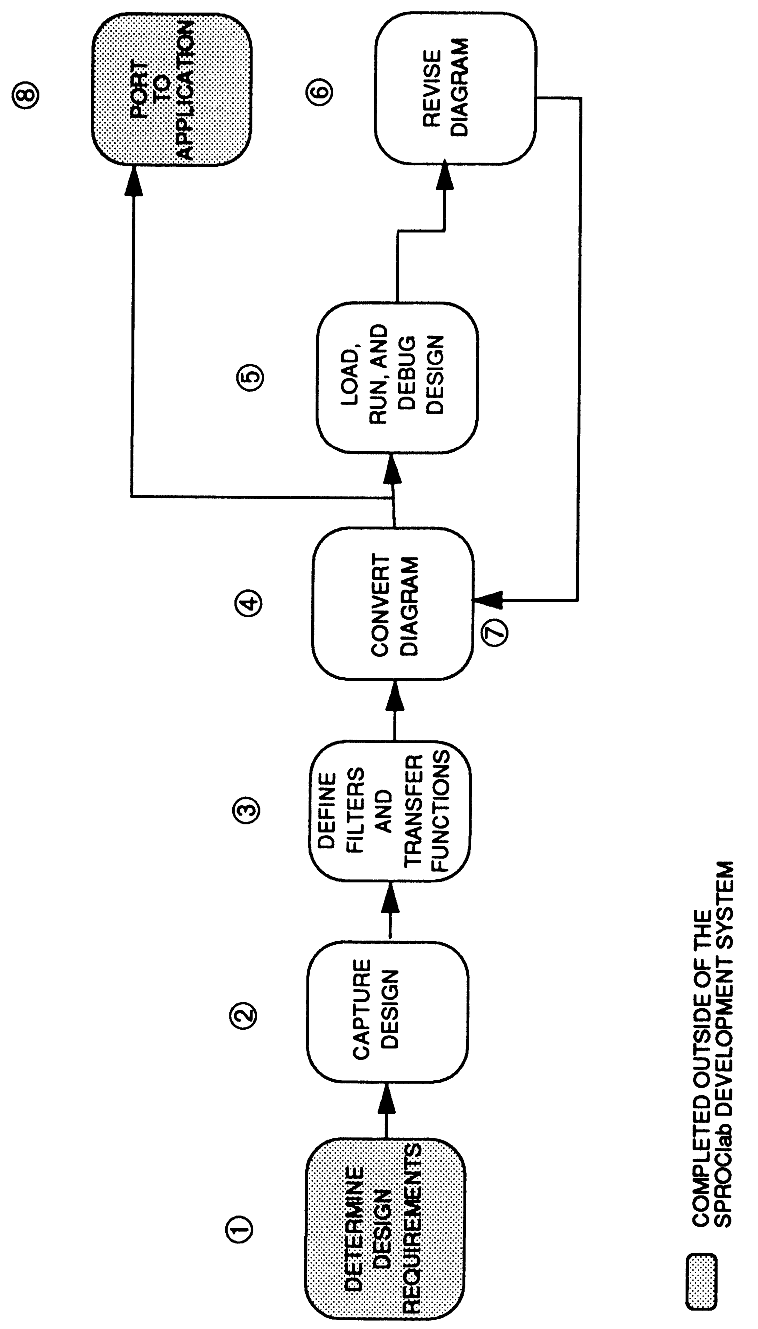
SPROClink software provides an interface between the SPROC system and a microprocessor development system.

The Development Process

The following steps describe the general process required to develop signal processing designs using the SPROClab development system:

1. Define the signal processing application and determine design requirements.
2. Capture the design as a signal flow diagram.
3. Define any necessary filters and transfer functions.
4. Convert the diagram and definitions into code and generate a load for the chip.
5. Load, run, and debug the design on the chip in real time.
6. Incorporate changes from debug into the design diagram.
7. Convert the revised diagram and definitions into code and generate an updated load.
8. Port the design into your actual application.

The following figure shows a diagram of the development process. The subsections that follow provide a general discussion of the steps in the process.



① DETERMINE DESIGN REQUIREMENTS
② CAPTURE DESIGN
③ DEFINE FILTERS AND TRANSFER FUNCTIONS
④ CONVERT DIAGRAM
⑤ LOAD, RUN, AND DEBUG DESIGN
⑥ REVISE DIAGRAM
⑦
⑧ PORT TO APPLICATION
▨ COMPLETED OUTSIDE OF THE SPROClab DEVELOPMENT SYSTEM

The Development Process

Capturing the Design

The development system's SPROCview graphical design interface enables a simple graphical approach to design capture. Capturing your design consists of entering the design as a signal flow diagram. To enter the diagram, you arrange and connect icons that represent processing functions into a schematic diagram that defines the signal flow of your system. As you select and place the icons, you must also enter certain variables, or parameters, that define how the functions represented by the icons will operate. For example, if your design includes an amplifier function, you must specify its gain value.

Some functions, like filters and transfer functions, are too complex to be defined using simple parameters. For these functions, you must create a separate data file that includes the detailed definition of the function. When you use a filter or a transfer function in a diagram, you must enter a parameter to identify the data file that contains the definition of the function.

The schematic diagram and its associated definition data files are the representation of the design upon which all other steps of the process build. You should consider them the base record of the design, and always make sure they are current.

Defining Filters and Transfer Functions

In designs that include filters or transfer functions, you must create the data files that specify the definition of the functions. The SPROCfil filter design interface provides an interactive environment for designing filters. You must define transfer functions using a text editor.

Converting Diagrams

After you capture the design and define any necessary filters or transfer functions, you must build the diagram and definition data files into a load that can run on the chip.

Each time you modify the diagram or the definition data files, you must re-build to convert the files again and produce an up-to-date load file.

Loading, Running, Debugging, and Verifying Designs

To debug and verify your design, transfer the load file onto the chip and run the design. The SPROCdrive interface (SDI) allows you to write the load to the chip and begin design execution. Using SDI, you can evaluate design performance by accessing the value of data in chip memory. If your development system is connected to an oscilloscope, you can view the waveforms represented by this data. If your development system is connected to a target analog subsystem, you can see how the design performs in the actual application.

To optimize your design, you can modify the values of data and observe the corresponding changes in design performance. If your development system is connected to a signal generator, you can simulate various input signals and evaluate how your design reacts.

Changes you make to design parameters using SDI are temporary. You must modify the schematic diagram and/or definition data files, then build the files and generate a new load file to make design modifications permanent.

Porting the Application

Once you have debugged and optimized your design, modified the diagram, and generated the final load file, you can port your signal processing design for use in your end application.

If the application is to run from a self-booting chip, you can use the load file to burn an EPROM, and place the chip and its EPROM on to your specific printed circuit board.

If the application is to run from a microprocessor, the SPROClink microprocessor interface (SMI) helps you to develop a microprocessor application that can use the signal processing design. You must generate a special version of the load file, use an embedded systems design tool to create the microprocessor application, and memory map the SPROC chip into the microprocessor configuration.

SPROClab Components

The development system comprises both hardware and software tools designed to help you complete the development process. These tools were designed in parallel with the SPROC chip to extract maximum efficiency and performance from the chip without compromising ease-of-use.

Hardware

Each hardware component is listed and described below:

- SPROCboard evaluation board

  The evaluation board is a printed circuit board with one SPROC chip, digital-to-analog and analog-to-digital converters, and various communications interfaces and additional components and circuity necessary to evaluate signal processing design performance during development. You can connect an oscilloscope, signal generator, or analog subsystem to the evaluation board to verify and evaluate your design.

  For details on the evaluation board, refer to the *SPROCboard Evaluation Board Reference Manual.*

- SPROCbox interface unit

  The interface unit provides an I/O connection between the SPROCboard evaluation board and your PC. It also connects the evaluation board to the power supply unit.

  For details on the interface unit, refer to the *SPROCbox Interface Unit Reference Manual.*

- Power supply unit

  The power supply unit converts AC power from a standard wall outlet to 5 VDC and ±12 VDC to supply interface unit and evaluation board.

- Interface cables

  An RS-232 cable connects the PC serial I/O port to the SPROCbox serial I/O port.

  A special access port cable connects the SPROCbox interface unit to the access port on the SPROCboard evaluation board.

For more information on the interface cables, refer to the *SPROCbox Interface Unit Reference Manual.*

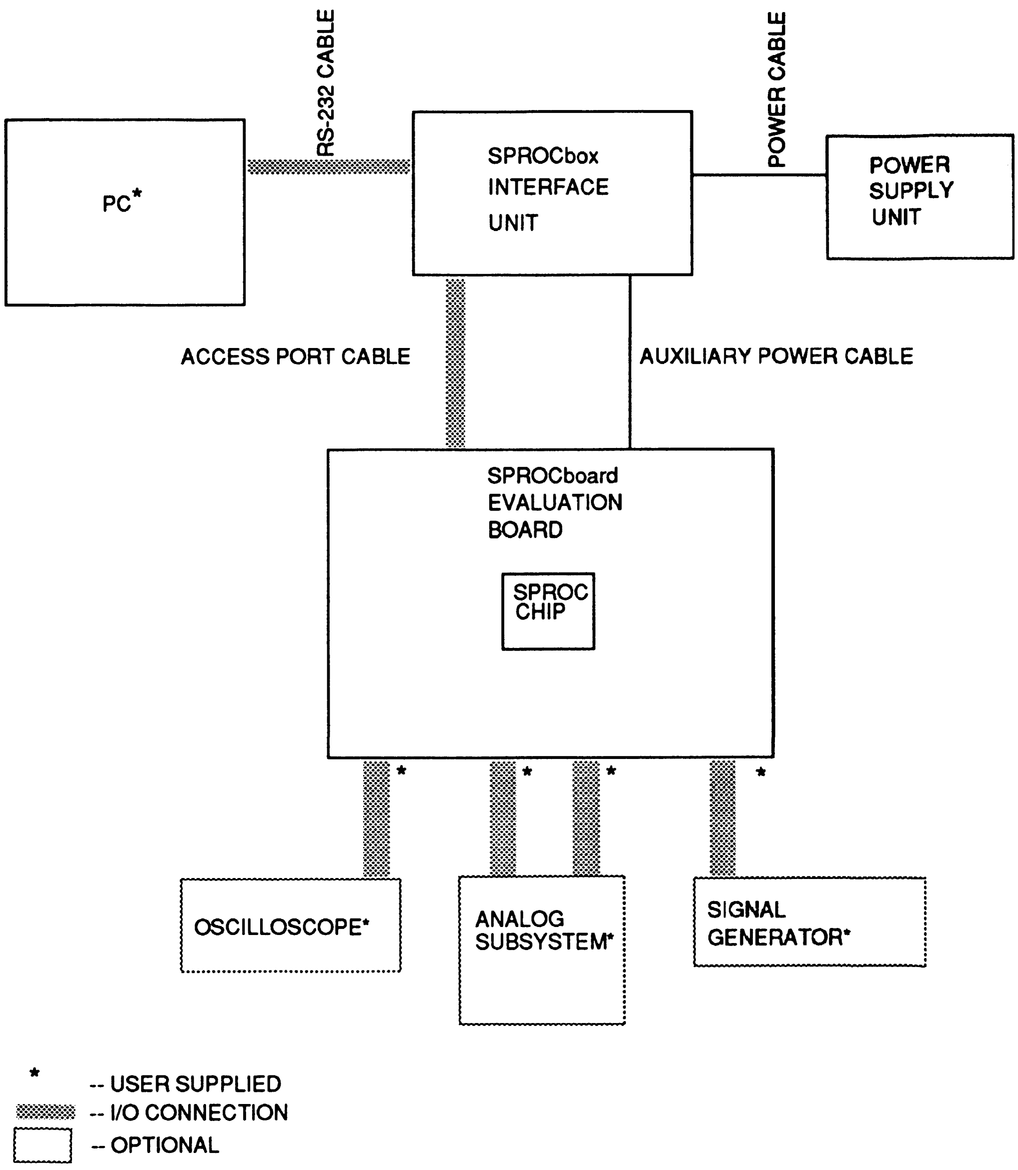
- Software security key

  The security key connects to the PC parallel port. It enables use of the software.

- Power cables

  A power cable connects the power supply unit to the AC outlet.

  An auxiliary power cable connects the power from the SPROCbox to the evaluation board.



SPROClab Development System Hardware

Software

Each software component is listed and described below:

- SPROClab development system shell

  The development system shell is an MS-DOS shell that provides access to all development system software components from a selection menu. The shell controls function calls among development system software components and provides a means for you to change certain system defaults.

  In some optional development system configurations, the environment shell is not necessary. All development system software components are available through menus or other methods in the optional environment. (For example, in systems using the VIEW*logic* environment, all development system software components are available through customized menus in the VIEW*logic* tool Workview.)

- SPROCview graphical design interface

  The graphical design interface provides for easy creation of signal-flow block diagrams by supporting the importation of designs created using several common schematic capture packages.

  The basic development system configuration supports version 4.04 of the OrCAD schematic capture tool DRAFT. The graphical design interface includes the library structure required to use the SPROCcells function library with the DRAFT tool. This tool is not supplied as a part of the graphical design interface. You must purchase it from OrCAD separately.

  In development systems with STAR's VIEWlogic option, the SPROCview graphical design interface supports importation of designs created using VIEW*logic* tools.

- SPROCcells function library

  The function library includes cells containing DSP and analog signal processing functions for use in diagram creation. A library cell is a design primitive that includes an icon required to place a function in a signal-flow diagram, the code required to execute the function, and specifications for the parameters required to define the cell.

  For details on the cell library, refer to the *SPROCcells Function Library Reference Manual.*

- SPROCfil filter design interface

  The filter design interface supports the definition and analysis of custom digital filters. The filter design interface creates the custom code and definition data for generic filter cells placed in designs during diagram entry.

- SPROCbuild utility

  SPROCbuild converts signal-flow block diagrams and their associated definition files into the load files necessary to run on the chip. SPROCbuild interprets the output from schematic entry and incorporates associated code blocks and parameter data for cells, filter design definitions, and transfer function definitions, then schedules and links the instructions to best utilize resources on the chip. The utility automatically generates efficient code based on your signal-flow block diagram.

- SPROCdrive interface

  The SPROCdrive interface (SDI) loads the design onto the chip and starts execution. SDI commands give you access, through the SPROCbox interface unit, to interactively test and debug the design while it runs on the chip. You can probe and modify signal values and design parameters to tune and optimize your processing subsystem.

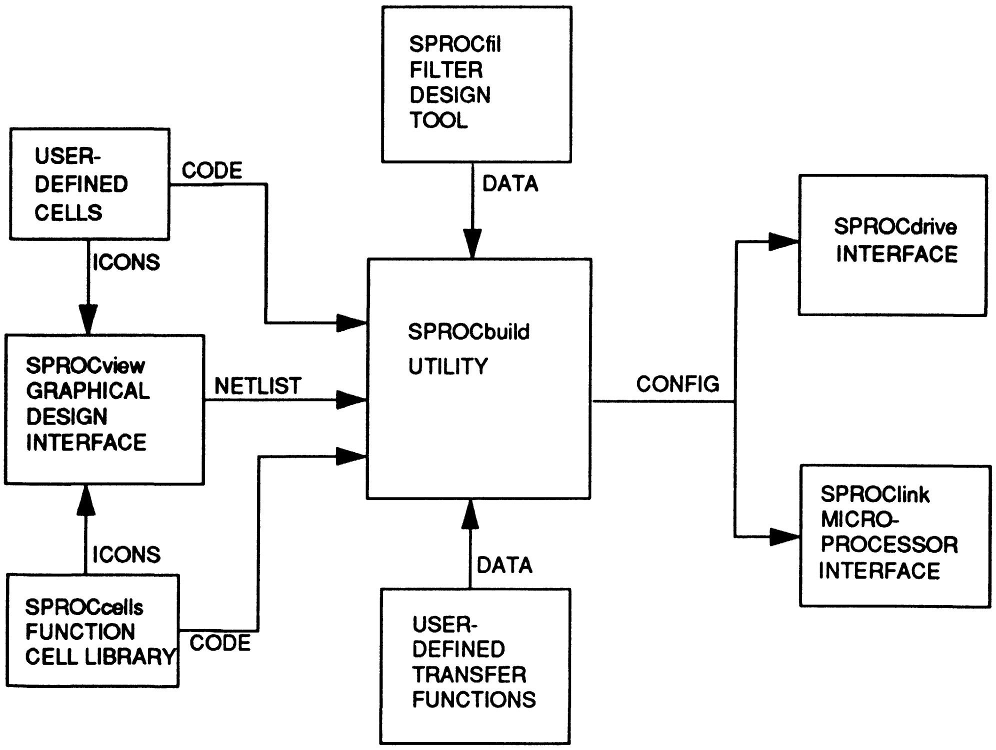
- SPROClink microprocessor interface

  The SPROClink microprocessor interface (SMI) provides software components necessary for you to develop microprocessor applications in ANSI C that include the SPROC chip as a memory-mapped device.

  For details on SMI, refer to the *SPROClink Microprocessor Interface Reference Manual.*

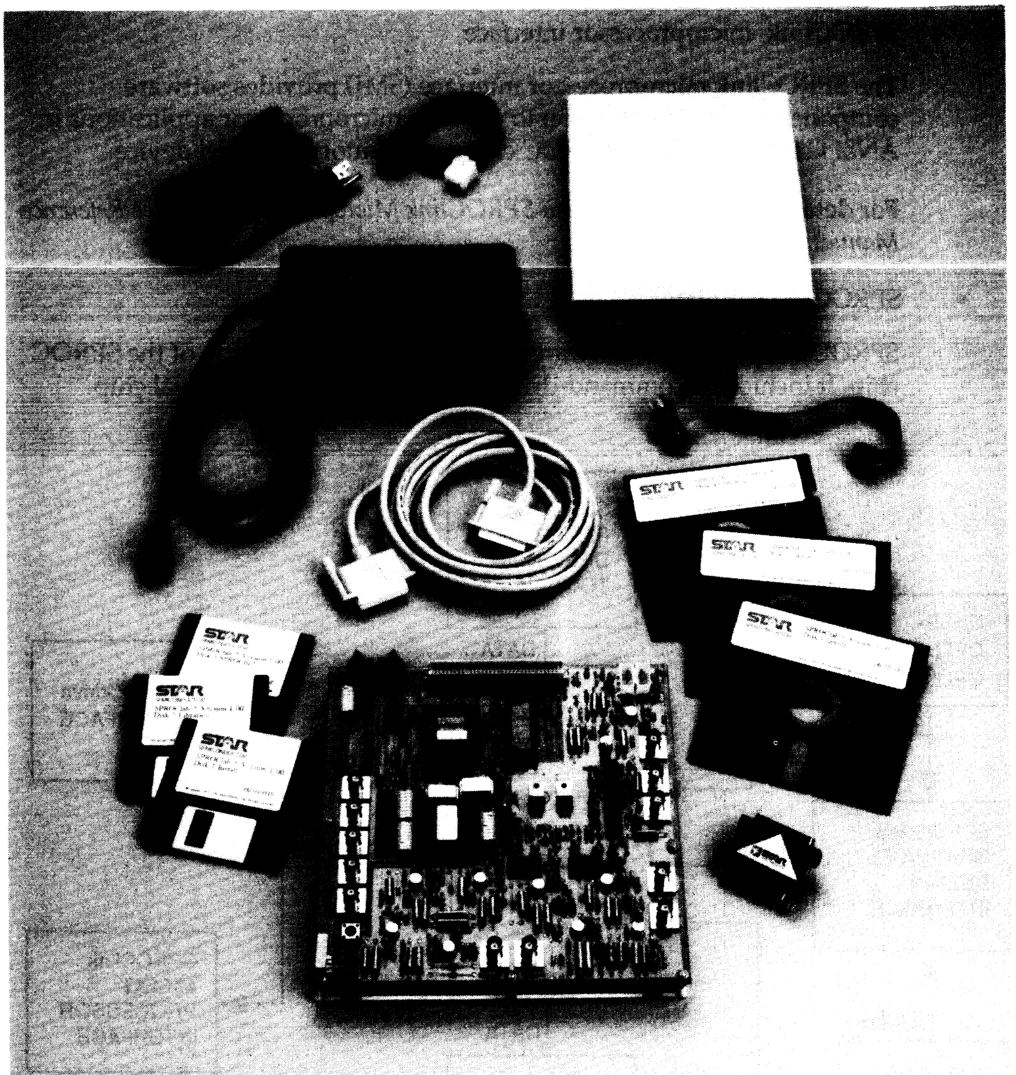
- SPROCsim simulator

  SPROCsim provides a software simulation of the operation of the SPROC chip. It includes a command-driven user interface to the virtual chip.


SPROCfil FILTER DESIGN TOOL
USER-DEFINED CELLS
CODE
DATA
SPROCdrive INTERFACE
ICONS
SPROCbuild UTILITY
SPROCview GRAPHICAL DESIGN INTERFACE
NETLIST
CONFIG
SPROClink MICRO-PROCESSOR INTERFACE
ICONS
DATA
SPROCcells FUNCTION CELL LIBRARY
CODE
USER-DEFINED TRANSFER FUNCTIONS

NOTE: THE DEVELOPMENT SYSTEM SHELL IS NOT SHOWN.

SPROClab Development System Software Components


STAR
STAR
STAR
STAR
STAR
STAR

SPROClab System Hardware

SPROCbox

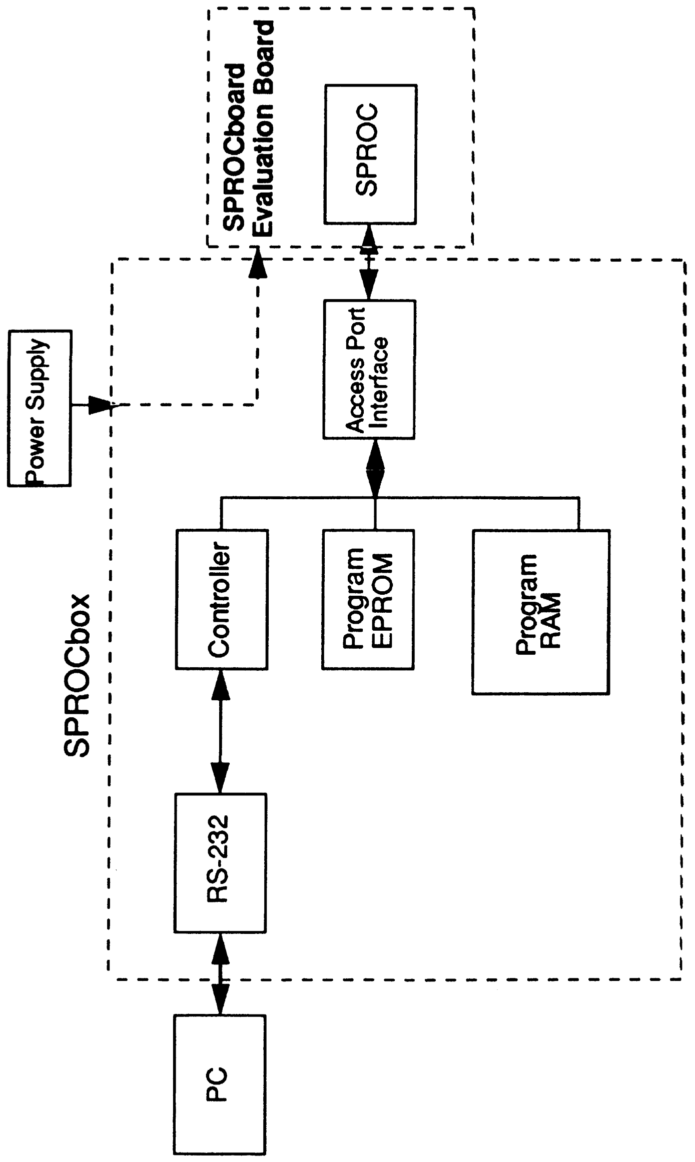
SPROCbox provides a bi-directional communications interface between the PC and SPROC chip. When used in conjunction with PC-resident SPROCdrive ssoftware it enables the designer to access the SPROC device for program downloading and for configuration and application debugging. All PC user requests to the SPROC integrated circuit are serviced by this interface.

Features

- Microprocessor-controlled asynchronous serial PC interface and synchronous serial SPROC interface (access port)
- EIA RS - 232C/CCITT 0.24 compatibility
- Power-up diagnostics/selftest
- Supports 1200, 2400, 9600 and 19200 based operation (default rate, 9600 baud) on RS232C port.
- Built-in custom serial interface signal cable for communicating to a SPROC chip.

Description

SPROCbox consists of a microprocessor, associated RAM and ROM storage, asynchronous serial interface to the PC, control logic, and a synchronous serial interface to the SPROC access port. The microprocessor processes commands received from the PC and generates commands to the SPROC access port. A DB-25 (female) connector is provided for the EIA RS - 232C compatible PC interface.



SPROCbox Interface

The access port interface is a custom serial interface specifically designed for communicating to a SPROC device. A cable with a universal 14-pin header is provided from the SPROCbox carrying TTL signals as shown.

HEADER	FUNCTION
1	RECEIVE DATA (INPUT)
2	RECEIVE STROBE (INPUT)
3	TRANSMIT STROBE (OUTPUT)
4	TRANSMIT DATA (OUTPUT)
5	SERIALCLOCK (OUTPUT)
6	RESETB (OUTPUT)
7	Not Connected
8 - 14	GROUND

RESETB (pin 6) has a series resistance of nominally 560 ohms which enables a user's circuit to overdrive this reset if desired.

The SPROC access port is intended for use by the SPROCbox but may also be used by the user's application. Timing specifications for the access port are given in the *SPROC Signal Processor Data Sheet.*

Power Connectors

Two identical 5-pin connectors are provided on the rear of the SPROCbox. The connector marked as POWER IN should be connected to the supplied tabletop power supply. The connector marked POWER OUT should be connected to the SPROCboard with the supplied daisy chain power cable. If the tabletop power supply is not used, refer to the following table, which specifies the configuration of the 5-pin type power connector.

PIN	VOLTAGE	SUPPLY
1	GROUND	
2	Not connected	
3	+5V	2.0A
4	-12V	0.5A
5	+12V	1.5A

Reset Switch

A rear panel momentary action reset switch is provided in case of a system malfunction. If the reset switch is activated while an application is being debugged, it will be necessary to download the application to the SPROC device again before resuming the debug session.

Power Requirements

The supplied tabletop power supply is sufficient to power both the SPROCbox and SPROCboard.

Power Supply (included): 115/230 Volts, 50/60 Hz

+5V, 1.5 amp; +12V, 1.5 amp; -12V, 0.3 amp

PC Interface

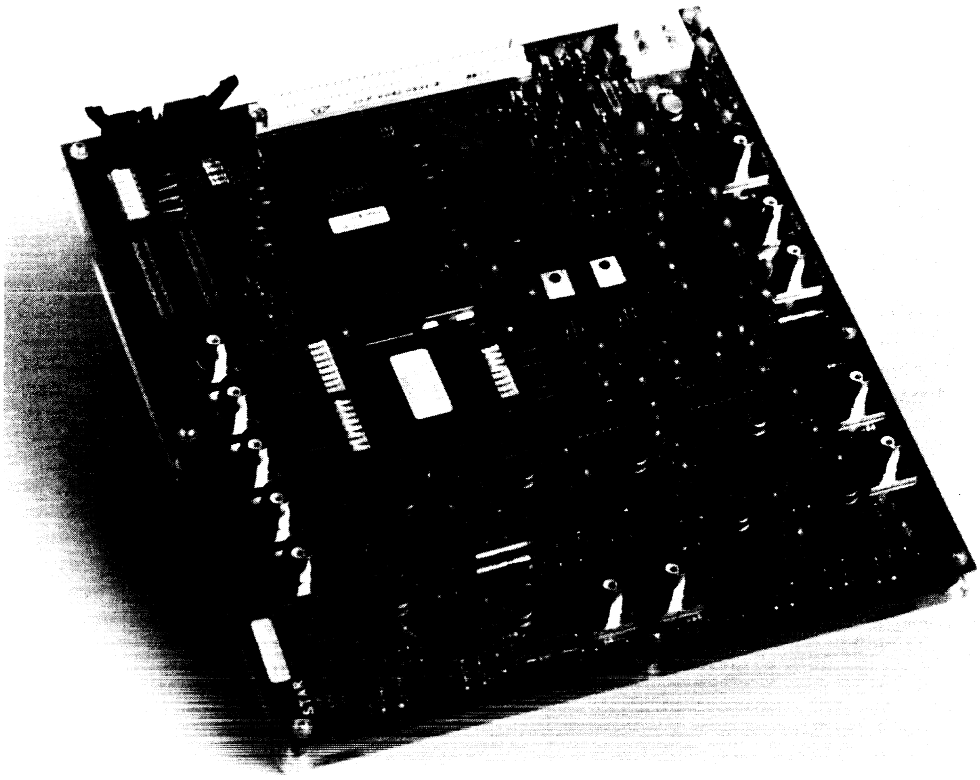
EIA RS-232C/CCITT 0.24

supports 1200, 2400, 9600 and 19200 baud operation; default is 9600

Physical dimensions:

L x W x H = 9.0″ x 7.3″ x 1.5″

Weight = 2 lbs., 7 oz.



SPROCboard Evaluation Board

Description

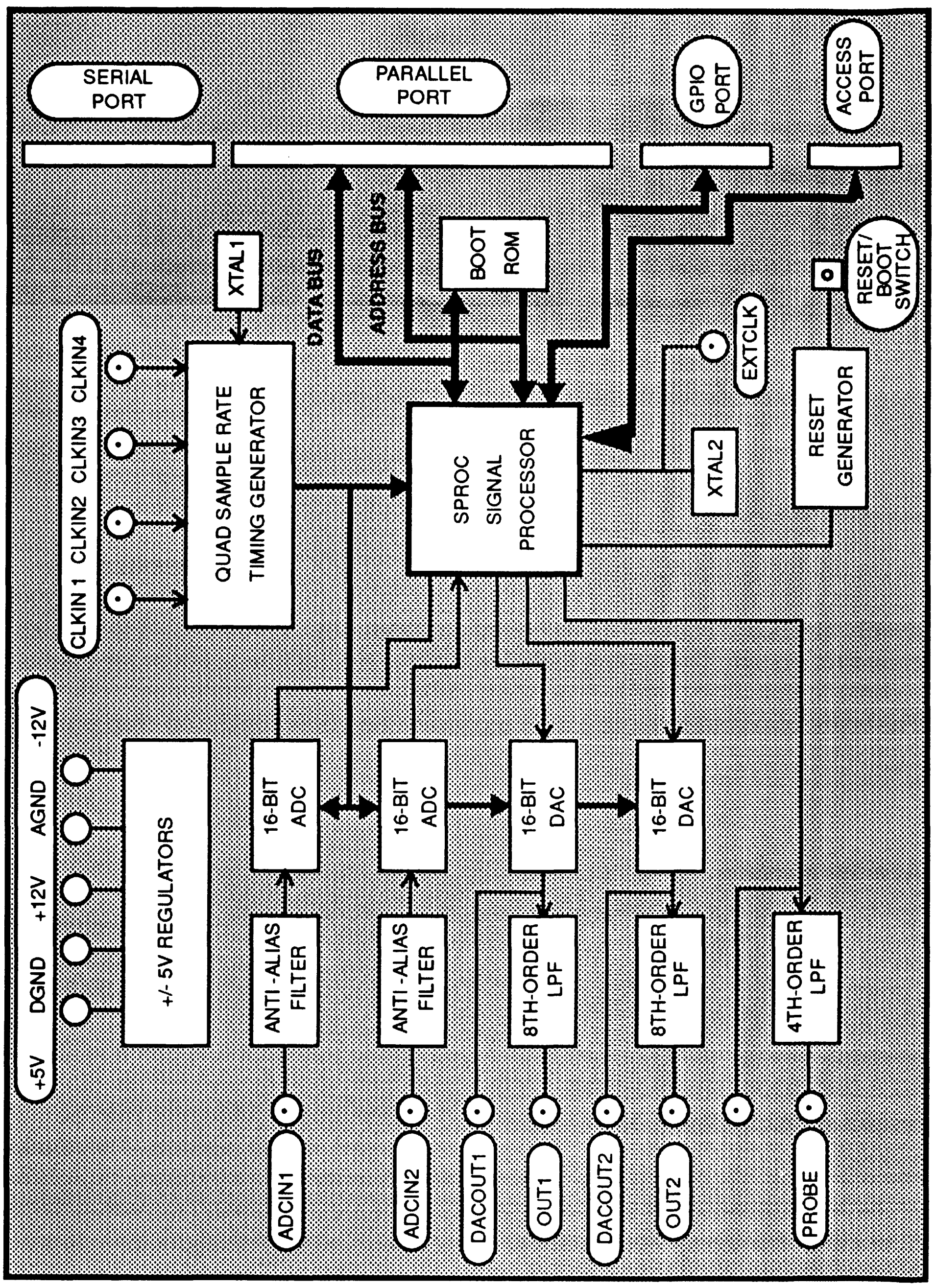
The SPROCboard is a versatile evaluation system. The SPROCboard requires a power source and a SPROCbox to provide a development interface to a PC.

The SPROCboard supports both standalone (master) and embedded microprocessor (slave) operating modes. In master mode the SPROC processor will automatically upload a program from the boot EPROM on the development board. In slave mode, the SPROC is configured via a microprocessor development system connected to the SPROCboard parallel port connector. Two SPROCboards may be interconnected as a master/slave SPROC configuration via the parallel port connector.

In the following figure, a block diagram of the SPROCboard shows its major sections and interface connections, which are subsequently described in greater detail. The board features a dual 16-bit analog-to-digital and digital-to-analog converter system. The system includes input anti-alias filters, 8th order output reconstruction filters, and independent sample rate generators for each converter. The board also contains a 4th order reconstruction filter for the on-chip probe digital-to-analog converter and a general purpose nibble wide, bidirectional interface port (GPIO port).

Analog Interface

The SPROCboard has two Crystal CS5317 16-bit A/D converters connected to serial input ports 0 and 1 of the SPROC 1400 signal processor chip, and two Burr Brown PCM56 16-bit D/A converters connected to serial output ports 0 and 1. The A/D converters are of the sigma delta design and require minimal anti alias filtering. A single pole RC filter is included on the board for this function. The D/A converters have 8th order 0.1dB ripple Chebyshev reconstruction filters, factory set with a 6kHz cutoff frequency. The filter cutoff frequency may be changed by the user by scaling the value of 12 equal valued resistors. The nominal resistor value is 3.3kOhm for a 6kHz cutoff frequency. The maximum sample rate of the A/D converters is 20 kilo conversions per second(kcps). The nominal 6kHz filter cutoff frequency was chosen to allow SPROC system designs with output harmonics extending to one half of Nyquist rate (5kHz). The unfiltered outputs of the D/A converters are provided to allow the user to bypass the on-board reconstruction filters.



SPROCboard Block Diagram

Input/Output BNC connectors

•	ADCin1:	ADC1 analog input connected to serial input siport0.
•	ADCIN2:	ADC2 analog input connected to serial input siport1.
•	DACOUT1:	DAC1 analog output connected to serial output soport0.
•		(unfiltered staircase waveform);
•	DACOUT2:	DAC2 analog output connected to serial output soport1.
•		(unfiltered staircase waveform);
•	OUT1:	filtered DAC2 analog output.
•		(reconstructed continuous time waveform);
•	OUT2:	filtered DAC2 analog output.
•		(reconstructed continuous time waveform);
•	PROBE:	filtered SPROCprobe analog output.
•		(reconstructed continuous time waveform);

(Note: An unfiltered probe output waveform is available at test point TP1. This is an unbuffered signal intended for a high impedance, low capacitance scope probe monitor point).

SPROCboard Sample Rate Selection

The conversion rate of the A/D converters may be independently controlled in 7 binary divisions from 19.53kcps to 152.6cps(19.53kcps/128) using 3 DIP switch settings per channel. Alternatively, the conversion rate may be controlled by an external clock frequency. Prescaling is required for the sigma delta A/D converters which use oversampling. If the external clock frequency is 10.24MHz, then the maximum sample rate would be 10.24 MHz/512 or 20kcps. The sample rate dividers operate with both internal and external clock source. For the above external clock source, the minimum conversion rate would be 20kcps/128 or 156.25cps.

D/A Converter Transport Clock Selection

The sample rate of the D/A converters is controlled by the SPROC output port configuration. The serial interface transport clock is provided by the analog board. This clock rate must be high enough to support data transport from the output serial port to the D/A converter at the desired sample rate. It is recommended that the transport clock is greater than 32x the sample rate of the serial port. In external clock mode, the D/A converters DAC1 and DAC2 use external clock inputs 'CLKIN3' and 'CLKLIN4' respectively. In internal mode, the clock source is one half that of the 20MHz board crystal oscillator frequency. The maximum rate is one half that of the clock source and the minimum rate is 1/16th that of the clock source. A per channel 2 bit DIP switch setting determines the divider rate of each channel.

SPROC Configuration Switches

The SPROC signal processor is configured via a DIP switch setting. This switch enables the user to configure the SPROCboard for master or slave operation. In slave mode, the configuration of the parallel port is determined by three mode switches. The SPROC signal processor may be driven by an external clock source or it may use its internal ring oscillator. The EXTINTBCLKP switch selects between the two clock modes. If external clock source is selected, then either the on-board crystal oscillator or an external clock source applied to BNC connector EXTCLK is used. A board jumper selects between the two external clock sources. The precise frequency of the internal oscillator will vary from chip to chip and is sensitive to supply voltage and temperature. The *SPROC Signal Processor Data Sheet* should be referenced for more detail.

Board Connectors

- PARALLEL PORT connector

  This connector provides direct access to the SPROC parallel port data and address buses. The port is unbuffered and consequently the user is referred to the SPROC electrical specifications for drive and load details.

- ACCESS PORT connector

  This connector provides a communication link between the SPROCbox and the SPROC chip. The link supports interactive configurations, probing, and debugging using the SPROClab development system.

Header Connectors

- General Purpose Interface Connector

  This header/connector is used to interface with the SPROC General Purpose Input/Output port (GPIO). This port is a quad bit-configurable interface. The port configuration is soft configured by the SPROClab development system.

  This connector is also used to interface with the SPROC compute trigger bus inputs and the SPROC RTS (Request To Send) flags.

SPROCboard Default Configuration

Factory-set default switch and jumper settings will configure the SPROCboard for normal operation at the maximum 19.53kcps provided by the on-board crystal oscillator clock source.

SPROClab Configuration

The SPROCboard is designed to be fully compatible with the SPROClab design tool suite. The serial ports of the SPROC 1400 signal processor chip, have been designed to be highly configurable to allow a wide selection of industry standard converters to be used without external interface hardware. The default mode of the serial port cells supplied with the SPROCview schematic entry system, are compatible with the A/D and D/A converters used on the SPROCboard.

To connect to either of the SPROCboard A/D converters, the user is only required to instantiate a serial input port cell and enter two parameters in the icon parameter edit field. The sample rate (rate = ?) and the trigger source (trigger = port?) must be specified. The sample rate must match the conversion rate set on the SPROCboard for the selected A/D converter. The port number entered selects which of the A/D converters is to be used as input; siport0 selects ADC1 and siport1 selects ADC2.

To connect to either of the SPROCboard D/A converters, the user is only required to instantiate a serial output port cell and enter a single parameter in the icon parameter edit field. The destination port number (dest = port?), specifies which of the two SPROCboard D/A converters is to be used; soport0 selects DAC1 and soport1 selects DAC2.

The user is referred to the *SPROClab Development System User's Guide* for more detailed information on design entry and for details on design examples which may be downloaded and run on the SPROCboard.

Electrical & Mechanical Specifications

Analog Specifications

CS5317 A/D converter:

resolution:	16 bits	(no missing codes)
differential nonlinearity:	+/- 0.4 bits	(typical)
dynamic range:	+/- 2.75V	(minimum)
impedance(1kHz):	80KOhms	(typical)
range:	84dB	(typical)
total harmonic distortion:	72dB	(minimum)

PCM56 D/A converter:

resolution:	16 bits	
monoticity:	15 bits	(typical
output voltage range:	+/- 3.0V	(typical)
dynamic range:	96dB	(typical)
total harmonic distortion:	0.008%	(maximum, Vo = +/-FS)
total harmonic distortion:	0.04%	(maximum, Vo = -20dB)

D/A output interpolation filter:

Filter prototype:	Chebyshev 8th order lowpass, 0.1dB in band ripple
Implementation:	4 Rauch biquadratic sections.
passband gain:	0dB (nominal)
SinX/X compensation:	NO
cutoff frequency:	6kHz (nominal)

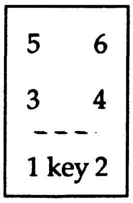
Clock Specifications

Maximum external clock frequency	20 MHz
Mark to space ratio	1 to 1 ± 10%

Input threshold is TTL and CMOS compatible

Power Connector

Pin 1 -	
Pin 2 -	
Pin3 - GND	(black)
Pin4 - Analog -12V	(purple)
Pin5 - Digital +5V	(red)
Pin6 - Analog +12V	(yellow)



(top view)

Power Rail	(Volts)	Power Current Requirement (Amps)
DIGITAL	+5V (+/- 5%)	<1.00A
ANALOG	+12V (+/-10%)	<0.50A
ANALOG	- 12V (+/-10%)	<0.50A

Connectors

External clock	BNC
Serial port clocks	4 x BNC
Probe out	BNC
DAC out (filtered)	2 x BNC
DAC out (unfiltered)	2 x BNC
ACD in	2 x BNC
Parallel port	64-pin inverted DIN
GPIO port	24-position header (2 x 12)
Access Port	14-pin universal ejection-style pin-header (2 x 7)

Physical dimensions:

L X W X H = 9.2" X 8.75" X 0.63" Weight = 1 lb., 7 oz.

SPROClab Packing List

Your SPROClab development system includes the following components:

Hardware

Power supply

SPROCbox

SPROCboard

SPROC security key

Cables

AC Power Cord

Auxiliary power cable

Access port cable

RS-232 serial cable

Diskettes

Install (3 1/2 and 5 1/4)

Libraries (3 1/2 and 5 1/4)

SPROCfil (3 1/2 and 5 1/4)

Documentation

SPROClab Development System Documentation Set includes:

SPROClab Development System User's Guide

SPROCboard Evaluation Board Reference Manual

SPROCdrive Interface Reference Manual

SPROC-1400 Programmable Signal Processor Data Sheet

SPROCbox Interface Unit Reference Manual

SPROClab Development System Unpacking and Installation Guide

SPROCcells Function Library Reference Manual

SPROClink Microprocessor Interface Reference Manual

SPROC Description Language Reference Manual

User-Supplied Equipment

You must supply the following equipment to complete the SPROClab development system environment.

IBM PC (386) or compatible

Mouse

BNC cables

Signal Generator

Oscilloscope (optional)

Recommended PC Configuration

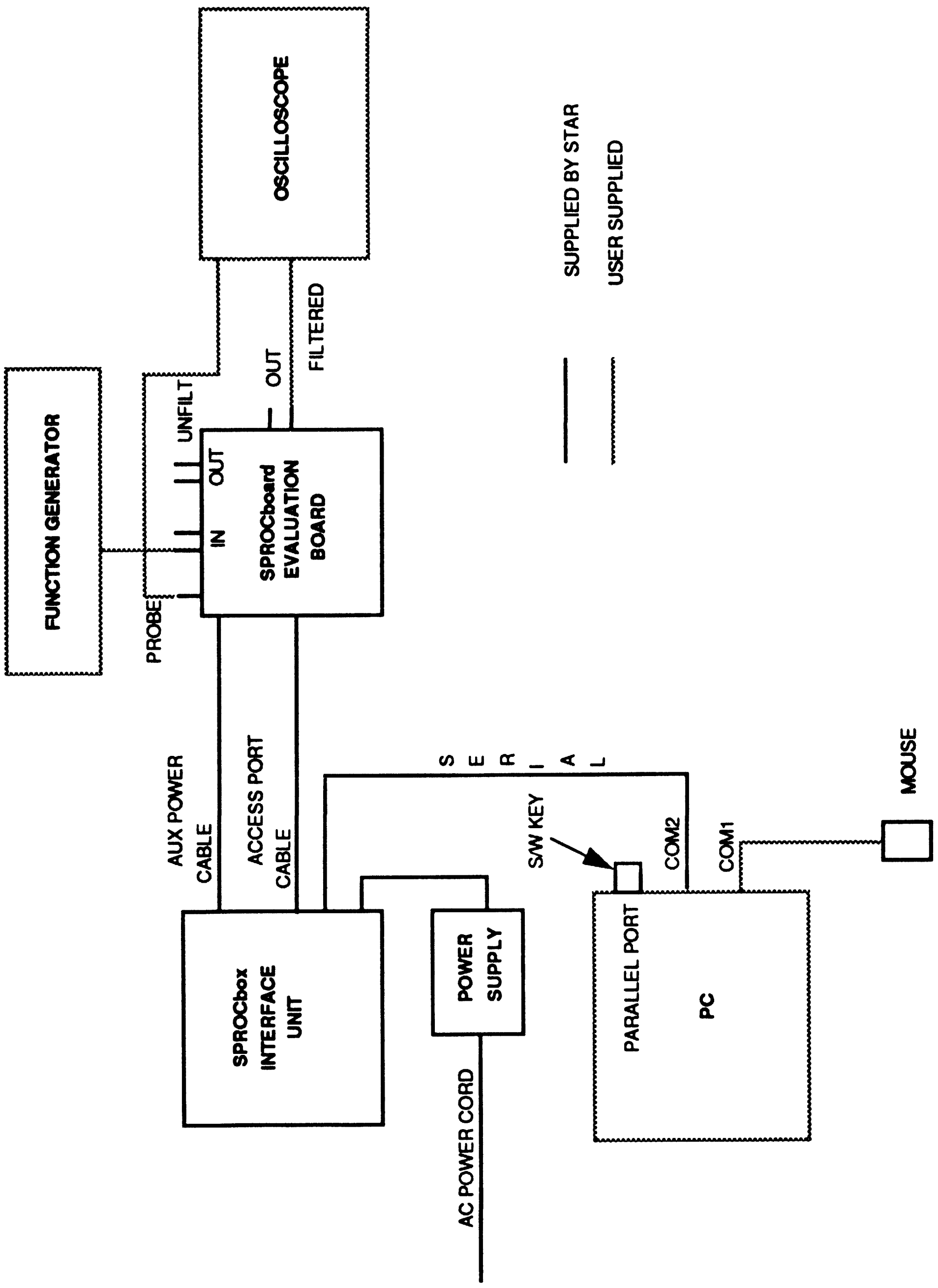
We recommend the following PC configuration for use with the SPROClab development system:

CPU:	80386 (80387 is optional)
Mem:	5 Mbyte extended (including the base 640K)
I/O:	2 serial ports (Mouse and SPROC)
	1 parallel (security key)

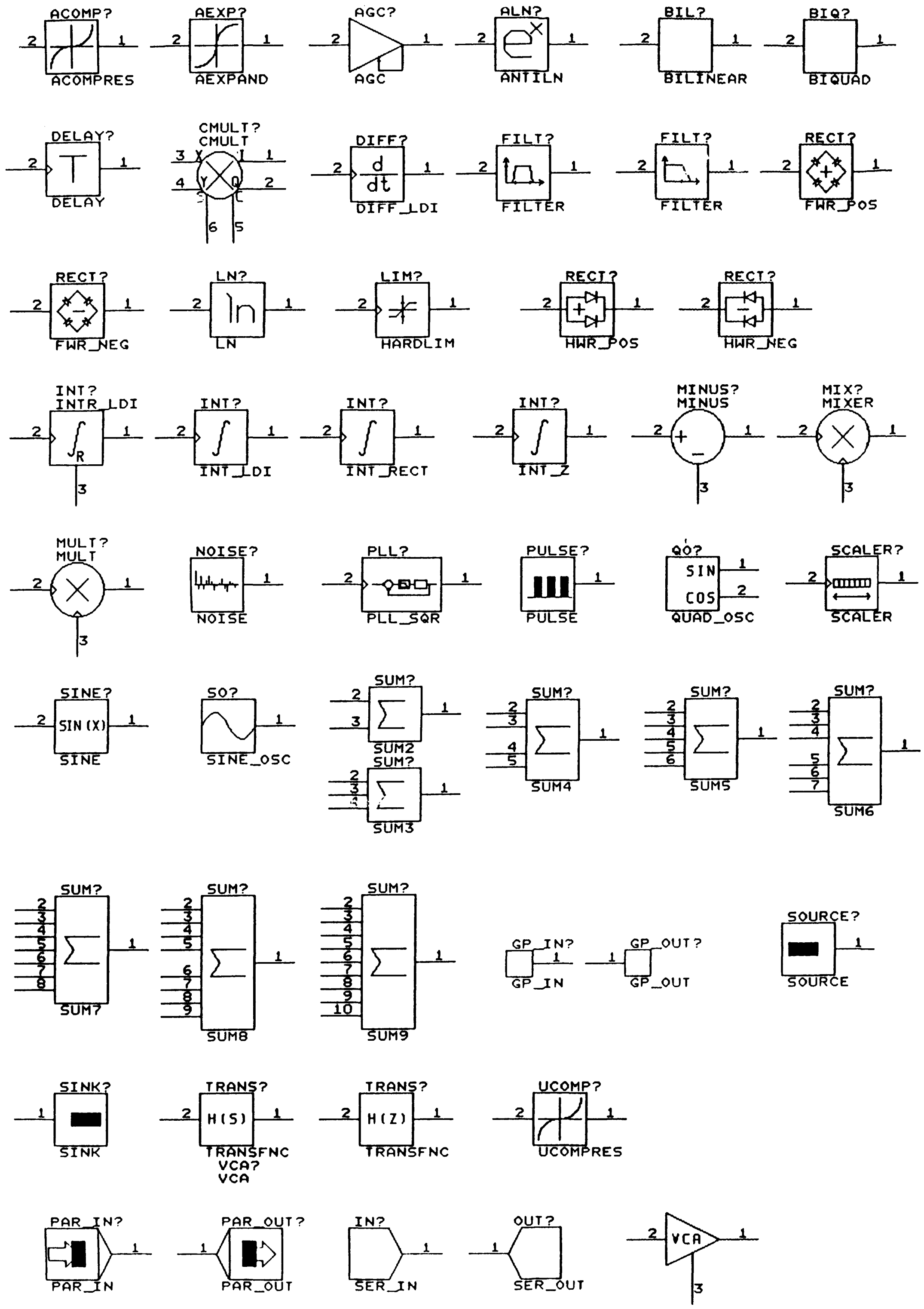
HDD:	40 Mbytes, with sufficient free space for development system ssoftware (2 M, 6 M with VIEW*logic*) and OrCAD software
FDD:	5 1/4 in., 1.2 M or
	3 1/2 in. 720 K
Display:	EGA/VGA (640 x 480)
Keyboard:	101 key
Mouse:	3 button serial
Operating system:	MS-DOS 3.2 or later

Hardware Installation

- Lay out the components.
- Connect the RS-232 cable to the PC COM2 connector and to the SPROCbox connector labeled SERIAL PORT.
- Connect the SPROCfil security key to the parallel port on the PC.
- Connect the access port cable to the connector on SPROCbox labeled ACCESS PORT.
- Connect the other end of the access port cable to the SPROCboard.
- Connect the power supply to the SPROCbox connector labeled PWR IN.
- Connect the auxiliary power cable to the connector on the SPROCbox labeled PWR OUT.
- Connect the other end of the auxiliary power cable to the SPROCboard.
- Verify that the switch on the power supply is in the OFF position and connect the AC power cord to the power supply.
- Plug the power cord into the wall outlet and turn the unit on. The indicator on the front of the SPROCbox should light.



Stand-alone Configuration for Use with Test Equipment


ACOMP? ACOMPRES
AEXP? AEXPAND
AGC? AGC
ALN? ANTILN
BIL? BILINEAR
BIQ? BIQUAD
DELAY? DELAY
CMULT? CMULT
DIFF? DIFF_LDI
FILT? FILTER
FILT? FILTER
RECT? FWR_POS
RECT? FWR_NEG
LN? LN
LIM? HARDLIM
RECT? HWR_POS
RECT? HWR_NEG
INT? INTR_LDI
INT? INT_LDI
INT? INT_RECT
INT? INT_Z
MINUS? MINUS
MIX? MIXER
MULT? MULT
NOISE? NOISE
PLL? PLL_SQR
PULSE? PULSE
QO? QUAD_OSC
SIN
COS
SCALER? SCALER
SINE? SINE
SIN(X)
SO? SINE_OSC
SUM? SUM2
SUM? SUM3
SUM? SUM4
SUM? SUM5
SUM? SUM6
SUM? SUM7
SUM? SUM8
SUM? SUM9
GP_IN? GP_IN
GP_OUT? GP_OUT
SOURCE? SOURCE
SINK? SINK
TRANS? TRANSFNC
H(S)
TRANS? TRANSFNC
H(Z)
UCOMP? UCOMPRES
VCA? VCA
PAR_IN? PAR_IN
PAR_OUT? PAR_OUT
IN? SER_IN
OUT? SER_OUT
VCA

SPROClab System Software

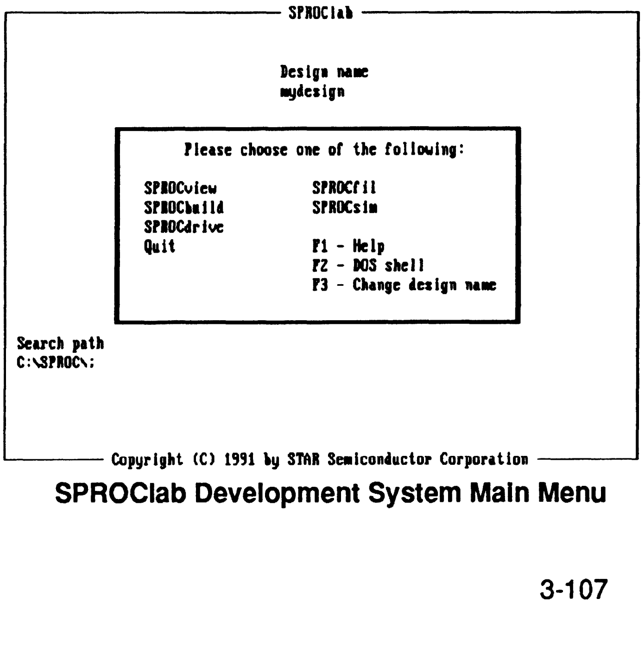
Development System Shell

In the default configuration, to get into the development system, you invoke the development system shell from a DOS prompt. The shell controls function calls among software components, and it displays a main menu that provides access to development system ssoftware tools and system configuration settings.

In the optional VIEW*logic* environment, you invoke the development system from a customized VIEW*logic* menu. The SPROClab main menu does not appear, although the functionality of the shell remains available through the customized VIEW*logic* menus.

When you invoke the development system shell, the shell checks the current directory (called your work directory) for certain files that include configuration information for development system software components. If the files are not found in the work directory (i.e., if the directory is a new work directory), the shell copies the files from the development system initialization area (determined at installation) to the work directory. When you change system configuration options using main menu features and save those changes, the shell modifies your work directory copies of the files, not the original files in the system initialization area.

When the development system copies all necessary files and completes initialization, the main menu appears.



```
SPROClab

Design name
mydesign

Please choose one of the following:

SPROCview          SPROCfil
SPROCbuild         SPROCsim
SPROCdrive
Quit               F1 - Help
                   F2 - DOS shell
                   F3 - Change design name

Search path
C:\SPROC\;

Copyright (C) 1991 by STAR Semiconductor Corporation
```

SPROClab Development System Main Menu

Entering a Diagram

You capture your signal processing subsystem design by creating a signal flow block diagram that represents it. You create the diagram by using a schematic capture package to arrange and connect signal processing functions, or *cells*, in an order that represents the signal flow of the subsystem.

SPROCcells Function Library

The SPROCcells library of pre-programmed function blocks provides the basic building elements to create the signal-flow diagram of the system under development. You can create an unlimited number of complex, custom functions by simply combining basic elements and modifying the parameter values of the supplied primitives.

A cell is a design primitive corresponding to a specific block of SPROC description language (SDL) code. The SPROCcells function library includes many commonly used cells, and you can create additional cells in SDL code to meet your own special needs.

Each cell has a graphical symbol, or *icon*, that represents the cell and illustrates the number of inputs and outputs the cell uses. You insert a function into the signal processing flow by placing the icon for that cell into the signal flow diagram and connecting, or *wiring*, the icon to other icons in the diagram.

In addition, each cell has a set of parameters, that identify the cell and let you define its detailed operational specifications. Most cells have parameters that specify simple operational values, but some cells are more complex. For example, filter and transfer function cells require data files to completely define their operations. In such cases, the cell's parameter does not define a simple operational value, it specifies the name of a data file containing the definition.

When you insert the icon for a cell into a signal flow diagram, connect the inputs and outputs, and specify parameters for that occurrence of the cell, you create an instance of the cell. A cell instance includes the function, identification, connection, and parameter definition for a single occurrence of a cell within a diagram.

The SPROCcells library contains an expanding list of functional bulding blocks, including amplifiers, multipliers, phase-locked loops, filters, and detectors. The partial list of available cells is provided on the following pages.

ACOMPRES

The A Law compression requires a left-justified linear input value. This value is converted to a left-justified, 8 bit A law encoded output value. The 8 bit output code has the format:

PSSSQQQQ

where

P = sign bit

SSS = 3 bit segment code

QQQQ = 4 bit quantization code.

The linear value is a fixed point, 13-bit, left-justified, two's complement representation for integer values (-4095 <= in <= +4095) with the leftmost bit interpreted as the msb. The rightmost 11 bits are ignored.

AEXPAND

The A Law expansion requires a left-justified, 8 bit A law input format. This code is converted to an integer in the +/- 4032 range as a left justified 13 bit value. The 8 bit input code has the format:

PSSSQQQQ

where

P = sign bit

SSS = 3 bit segment code

QQQQ = 4 bit quantization code.

The output value is a fixed point, 13-bit, left-justified, two's complement representation for integer values (-4032 <= in <= +4032) with the leftmost bit interpreted as the msb. The rightmost 11 bits are zero.

AGC

The automatic gain control (AGC) handles input signals from 0 to -48dB. The level parameter determines the average agc output value, and the tc parameter sets the time constant of the agc gain response.

AMP

The amplifier, multiplies the value of the input by the value of the gain parameter.

ANTILN

The natural anti-logarithm of the input is calculated using the series approximation:

$\text{antiln}(x) = 1 + x + x^2/2! + x^3/3! + x^4/4! + x^5/5! + x^6/6! + x^7/7!$

For $x > 0.692$, the output is clipped to 1.9977.

BILINEAR

This bilinear first order recursive filter has the following equation:

$$y(t) = a1.y(t-1) + b1.x(t-1) + b0.x(t)$$

where $x(t) = in$, $y(t) = out$ and $(t-1)$ indicates previous sample.

BIQUAD

This biquad second order recursive filter has the following equation:

$$y(t) = a2.y(t-2) + a1.y(t-1) + b2.x(t-2) + b1.x(t-1) + b0.x(t)$$

where $x(t) = in$, $y(t) = out$, $(t-1)$ & $(t-2)$ are the two previous samples.

CMULT

This complex multiplier cell performs multiplication of the form:

$$i + jq = (x + j.y)*(\cos + j.\sin) = (x.\cos - y.\sin) + j.(x.\sin + y.\cos)$$

DECIM

This cell decimates its input. The number of input samples per single output is set by the factor parameter.

DIFFAMP

This differential amplifier, multiplies the differential value of the input by the value of the gain parameter:

out = (ina - inb) * gain

DIFFCOMP

This differential comparator with hysteresis produces an output of +/-1.0 according to the magnitude and sign of the differential input (ina-inb). If the magnitude of the differential input is less thanthat of the hysteresis, the previous output level is retained. If the magnitude of the differential input equals or exceeds that of the hysteresis, the output shall have magnitude 1.0 with the same sign asthat of the differential input. Initial output state is preset to +1.0

DIFF_LDI

The DIFF_LDI cell, outputs the difference between the input and the previous value of input.

out = in - in(previous)

DSINK

The DSINK accumulates two series of input samples (each size determined by the length parameter) into two blocks of data RAM. The blocks are stored beginning at symbolic location 'instance_name.outvector1' and 'instance_name.outvector2'.

DSINKRD

The DSINKRD accumulates two series of input samples (each size determined by the length parameter) into two blocks of data RAM. The blocks are stored beginning at symbolic location 'instance_name.outvector1' and 'instance_name.outvector2'.

A reset input is available: if >0.5 the cell is held in reset otherwise the cell can capture a series of input samples. The done output is zero if the cell is reset or capturing input samples, else the done output is one. The done output needs to be terminated, either by another block or by a dummy module. Reset is only effective when the sink block is full.

EXT_IN

The EXT_IN cell provides an external (off chip) input into the SPROC device. Typically the external input cell is used in conjunction with an external microprocessor.

EXT_OUT

The EXT_OUT cell provides an external (off chip) output. Typically the external output cell is used in conjunction with an external microprocessor.

FILTER

The FILTER cell is used for the implementation of filters designed with SPROCfil. For each instance of this cell there must be an associated filter design file produced by SPROCfil, an .fdf file. This file is identified with the spec parameter (warning: do not use the design name as the filter design file name).

An optional type parameter allows filter type verification during the compilation process.

Algorithm:

Each IIR filter cell in a SPROCfil design is implemented as a cascade of biquad cells, plus a bilinear cell for odd order filters. An FIR filter cell in a SPROCfil design is split into blocks, with a default of 30 coefficients, this is a scheduler parameter.

FIR

The FIR cell provides a single stage finite impulse response (fir) filter output computation with current input data, finite input data delay line and predefined filter coefficients.

Transfer function of FIR filter:

H(z) = b0 + b1.z^-1 + b2.z^-2 + ... + bx.z^-x

where x = length-1 and z^-1 represents a unit sample delay.

FWR_NEG

The negative fullwave rectifier provides the function of an ideal fullwave rectifier with a negative polarity output.

out = - | in |

FWR_POS

The positive fullwave rectifier provides the function of an ideal fullwave rectifier with a positive polarity output.

$out = |in|$

GP_IN

One of the four bidirectional gpio pins is configured as an input. The pin status is read from the SPROC and written to the output. If the port pin is a logic zero then the output is set to zero, else the output is set to non-zero.

GP_OUT

One of the four bidirectional gpio pins is configured as an output. If the cell input is zero then the gpio pin is set to a logic zero, else the gpio pin is set to a logic one.

HARDLIM

The hard limiter clamps the output to the inclusive range from lower to upper clamp thresholds. Equal upper and lower clamp thresholds will force the signal node to the assigned value. The upper threshold must be greater than or equal to the lower threshold.

Algorithm:
```
    if { lower <= in <= upper}, then out = in
    else if { in > upper }, then out = upper
    else if { in < lower }, then out = lower
```

HWR_NEG

The negative halfwave rectifier provides the function of an ideal halfwave rectifier with a negative polarity output.

Algorithm:
```
    if { out < 0 } then out = in,
    else out = 0.0.
```

HWR_POS

The positive halfwave rectifier provides the function of an ideal halfwave rectifier with a positive polarity output.

Algorithm:
```
    if { out > 0 } then out = in,
    else out = 0.0.
```

INTERP

This cell interpolates its output. The number of output samples generated per input is set by the factor parameter. The value of the output sample equals the value of the input sample from which it was generated.

INTR_LDI

The integrator is a sample value accumulator with a reset input.

Algorithm:
```
    if reset < 0.5
      out  = in(previous) + out(previous)
    or else
      out  = 0.0
```

INT_LDI

This integrator is a sample value accumulator.

Algorithm:
```
    out  = in(previous) + out(previous)
```

INT_RECT

This is the rectangular integration approximation of 1/s --> (Tz^-1)/(1-z^-1). [In effect this is just INT_LDI with the scale factor tsamp represented by T].

Algorithm:
```
    out  = in(previous)*tsamp + out(previous)
```

INT_Z

This integrator is the z-transform of 1/s --> T/(1-z^-1)

Algorithm:

```
    out(present)  = in * tsamp + out(previous)
```

LN

The natural logarithm is calculated using an eight term truncated series:

```
    ln(in) = ln(1+x) = x - x^2/2 + x^3/3 - x^4/4 + x^5/5 - x^6/6
                              + x^7/7 - x^8/8
```

In order to increase accuracy at the ends of the range of the input the following compression approach is applied:

```
    if in > 1.375, in = in/2  and  out = ln(in) + ln(2);
    if 0.1353 <= in < 0.6875, in = 2*in and  out = ln(in) - ln(2);
    if 0.6875 <= in <= 1.375, out = ln(in);
```

The percentage accuracy varies, with the highest error of 0.003% in the input range of 0.32 to < 2.0, and a highest error of 0.9% in the input range below 0.32.

MINUS

This difference junction provides the difference between two inputs.

Algorithm:

```
    out = pos - neg
```

MULT

The multiplier performs a multiplication of two inputs.

Algorithm:

```
    out = ina * inb
```

NOISE

This white noise generator uses simple accumulator overflow nonlinearity on integer multiply for random number generation. The output is scaledto +/- 1.0.

PLL_SQR

This phase-locked loop contains a first order loop filter and square wave VCO. The output of the PLL is a +/-1.0 amplitude square wave. The center parameter determines the output frequency for zero input. The gain parameter determines the frequency deviation with maximum input. Lag and lead position the filter pole and zero respectively, and filgain sets the loop filter gain.

PULSE

This is a rectangular pulse generator. Its duty cycle and levels for mark and space may be parameterised. Mlevel is the signal level when marking for mark sample intervals, and slevel is the signal level when spacing for space sample intervals.

QUAD_OSC

This sinewave oscillator produces two sampled sinewave outputs, 90 degrees out of phase, at a frequency specified as a fraction of the sample rate, fs. It calculates each sine of x, using the first 5 terms of the Maclaurin series:

sin(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!

RTS_IN

One of the four rts pins is used as an input (SPROC must be in master mode). If the rts pin is a logic zero then the cell output is set to zero, else the cell output is set to non-zero.

RTS_OUT

One of the four rts pins is used as an output (SPROC must be in slave mode). If the cell input is zero then the rts pin is set to a logic zero, else the rts pin is set to a logic one.

SCALER

The SCALER cell, scales the input to yield between 0 to 8 bits shift left or right, on the output. This is equivalent to a signal level shift in the range of -48 dB to +48 dB in 6 dB steps. The following formula applies:

out = in * 2^shift

For values of shift not in the specified range out is forced equal to in.

SER_IN

The serial input reads data from one of the two serial hardware ports (siport0 or siport1) on the SPROC device.

SER_OUT

The serial output places data into one of two serial hardware ports (soport0 or soport1) on the SPROC device.

SINE

Sine is calculated as pi/2 times the input using the first 5 terms of the Maclaurin series:

sin(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!
where x = in.(pi/2)

SINE_OSC

The sinewave oscillator produces a sampled sinewave output at a frequency specified as a fraction of the sample rate, fs. It calculates sine of x using the first 5 terms of the Maclaurin series:

sin(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!

SINK

The sink accumulates a series of input samples (size determined by the length parameter) into a block of data RAM. The block is stored beginning at symbolic location 'instance_name.outvector'.

SINKRD

The SINKRD accumulates a series of input samples (size determined by the length parameter) into a block of data RAM. The block is stored beginning at symbolic location 'instance_name.outvector'. A reset input is available: if ≥=0.5 the cell is held in reset otherwise the cell can capture a series of input samples. The done output is zero if the cell is reset or capturing input samples, else the done output is one. Reset is only effective when the sink block is full.

SOURCE

The source repetitively reads an array of user specified sample values. The samples must be contained in a file, one sample per line, within the working directory, before scheduling. Source reads the samples one at a time from an array in data RAM, and the number of samples is specified by the length parameter. The array's position in RAM begins at symbolic location 'instance_name.invector'. Values of the sample data must be in the range from -2.0 to < 2.0 fixed point, but values can also be represented in hexadecimal and signed integer notation.

STEO_IN

The stereo input reads stereo data from one of the two serial hardware input ports (siport0 or siport1) on the SPROC device, and provides two outputs for signal processing.

STEO_OUT

The stereo output sends stereo data, from two signal paths, to one of two serial hardware output ports (soport0 or soport1) on the SPROC device.

SUM2-SUM10

The summing junctions add multiple inputs. All inputs are functionally equivalent. Care must be taken to scale inputs before summing, to avoid overflow, on the output.

TRANSFNC

The transfer function cell allows the user to directly specify a transfer function in either s or z domain. Each instance of the cell requires a transfer function file within the working directory, a .tff file, before scheduling. This file is identified with the spec parameter (warning: do not use the design name as the transfer function file name). For sdomain coefficients a bilinear mapping to z coefficients is performed. Additionally the .tff file may contain the frequency at which s and z map exactly, the "critical frequency".

UCOMPRES

The U Law compression cell requires a left-justified linear input value. This value is converted to a left-justified, 8 bit U law encoded value. The 8 bit output code has the format:

PSSSQQQQ

where
P = sign bit
SSS = 3 bit segment code
QQQQ = 4 bit quantization code.

The linear input value is a fixed point, 14-bit, left-justified, two's complement representation for integer values (-8159 <= in <= +8159) with the leftmost bit interpreted as the msb. The rightmost 10 bits are ignored.

UEXPAND

The U Law expansion cell requires a left-justified, 8 bit U law input format. This code is converted to an integer in the +/- 8031 range as a left justified 14 bit value. The 8 bit input code has the format:

PSSSQQQQ

where
P = sign bit
SSS = 3 bit segment code
QQQQ = 4 bit quantization code.

The output value is a fixed point, 14-bit, left-justified, two's complement representation for integer values (-8031 <= in <= +8031) with the leftmost bit interpreted as the msb. The rightmost 10 bits are zero.

VCO_SQR

This voltage controlled oscillator generates a square wave output with a +/-1.0 volt amplitude. The center parameter determines the output frequency for zero input.

Algorithm:

    output frequency = (center + gain*in) * sample rate

VOLTREF

The voltage reference provides a constant output voltage.

OrCAD and VIEW*logic* Graphical Design Interface

OrCAD

When you select SPROCview from the development system main menu, the development system shell invokes the OrCAD tool, DRAFT. You use the DRAFT tool to create a signal flow block diagram of your design.

The graphical design interface supports OrCAD software version 4.04.

You can access the DRAFT tool by using the standard methods provided with the OrCAD. However, if you are capturing a design for use with the development system, always access the DRAFT tool by invoking it from the development system main menu.

VIEW*logic*

The VIEW*logic* option for the SPROCview interface supports entry of schematics via Workview, a product of VIEW*logic* Systems, Inc.

SPROClab development system tools are invoked directly from Workview via customized menus. The user can create a schematic design for the application and invoke SPROClab tools to build, load, run, and debug the design using the customized VIEW*logic* menus.

Filter Design (SPROCfil)

If your signal processing design includes one or more filters, you create a data file, called a filter data file, that defines the detailed specifications and coefficient data for each filter. A parameter in each filter cell instance entered on the signal flow block diagram identifies the name of the filter data file to use with that filter.

When you use the SPROCbuild utility to convert the signal flow block diagram into code and generate a chip configuration file, the utility reads the filter data file for each filter cell instance and generates the appropriate code to implement the filter as specified. The generated code uses the coefficients from the filter data file and a cascade of special filter cells to implement the filter. The special cells are provided in the SPROCcells function library, but reserved for internal use by SPROCbuild.

The SPROCfil filter design interface helps you create filter data files that specify the coefficients and processing order to use in implementing a filter design. The filter design interface provides an interactive design environment that lets you define your filter using a graphical representation of the filter shape. Other tools in the filter design interface automatically generate the coefficients corresponding to the filter design, and write these coefficients to the filter data file.

Filter Types

The filter design interface supports design of the following major categories of digital filters:

- Infinite Impulse Response (IIR) or recursive filters, and
- Finite Impulse Response (FIR) or nonrecursive filters.

In the IIR category, there are four familiar analog types of filters:

- Butterworth,
- Chebyshev I,
- Chebyshev II (or inverse Chebyshev), and
- Elliptic function (for Cauer parameter).

In the FIR category, there are two filter types:

- Optimal Chebyshev approximation, commonly referred to as the Equiripple or Parks-McClellan-Remez (PMR) design, and
- Kaiser window design.

You can use these types to design lowpass (LP), highpass (HP), bandpass (BP), and bandstop (BS) filters.

Transfer Functions

SPROCbuild can implement user-defined transfer functions of two types:

- s-domain transfer functions
- z-domain transfer functions

when generating code and creating a SPROC chip configuration file.

The SPROCcells function library includes a transfer function cell that permits you to include transfer functions in your signal processing designs. When you place this cell in a diagram, you specify a parameter that names the file defining the transfer function.

SPROCbuild uses z-domain transfer functions directly and automatically converts s-domain transfer functions into z-domain transfer functions. The build implements the transfer function as a cascade of 1st-order or 2nd-order sections using the coefficients you define.

Converting Block Diagrams (SPROCbuild)

SPROCbuild provides a set of software modules that automatically converts your design into SPROC description language (SDL) code, then uses that code to generate a configuration file for the SPROC chip and a table of symbolic references to chip memory locations. To create these files, the utility uses files produced by the SPROCview graphical design interface, the SPROCcells function library, and the SPROCfil filter design interface in the development system, and user-defined cells and transfer functions of the proper form created outside the development system.

SPROCbuild includes three modules:

- **MakeSDL**
- **Schedule**
- **MakeLoad**

Each module performs a unique function in the process of converting your signal flow block diagram and associated files into SDL code and then into a SPROC chip configuration file and a symbol file for your design.

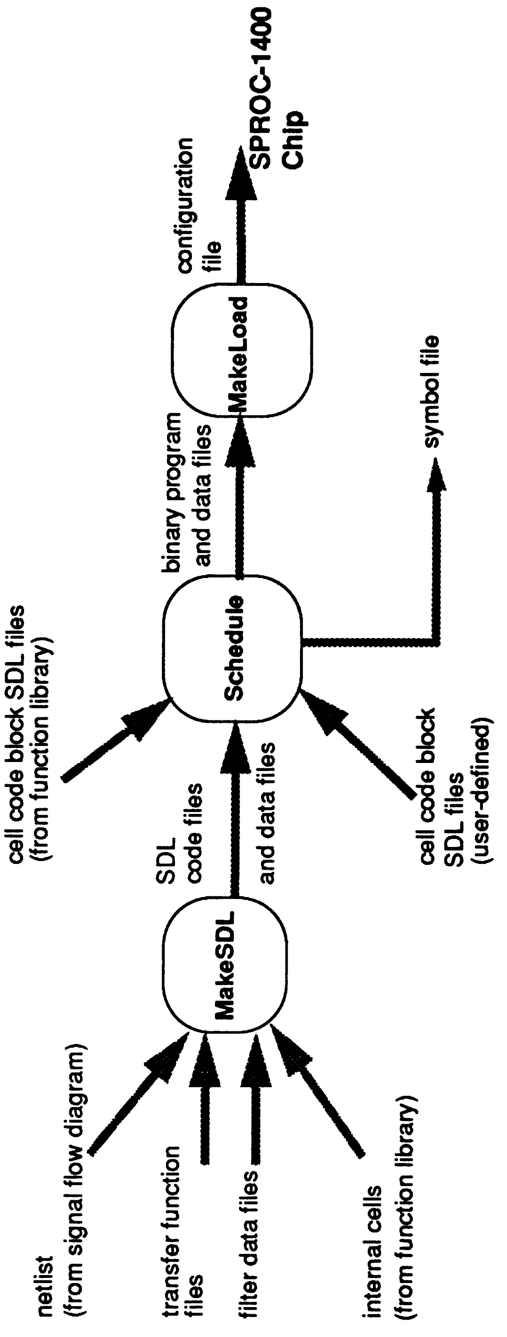
The Conversion Process

The conversion process comprises the sequential execution of all modules of the SPROCbuild utility. Each module performs its specific function in the process and produces an output file (or files) required by the next module. The general process is as follows:

1. The **MakeSDL** module integrates the output from the graphical design interface with data files from the filter design interface and user-defined transfer functions to produce a partial code package containing SDL code and data files. The module also generates instances of certain special cells that are included in the SPROCcells function library but reserved for internal use.

2. The **Schedule** module takes the partial package produced by **MakeSDL** and adds the code blocks for the cells used in the design (from the function library or user-defined cells) and any additional data files. Then the module schedules the program according to on-chip resource availability and produces binary format program and data files for the design. It also produces a table of symbolic references to chip memory locations.

3. The **MakeLoad** module takes the assembly language program and data files produced by **Schedule** and packages them into a configuration file for downloading to the chip.


netlist (from signal flow diagram)
transfer function files
filter data files
internal cells (from function library)
MakeSDL
SDL code files and data files
cell code block SDL files (from function library)
cell code block SDL files (user-defined)
Schedule
binary program and data files
symbol file
MakeLoad
configuration file
SPROC-1400 Chip

Overview of the Conversion Process

SPROClink Microprocessor Interface (SMI)

The SPROClink microprocessor interface (SMI) is a set of components you use in developing microprocessor applications that include the SPROC chip as a memory-mapped device. With the components of SMI, you can create microprocessor applications that separate the logic processing tasks that run best on a microprocessor from the computation-intensive real-time signal processing tasks that run best on the SPROC chip. By partitioning the design in this way, you increase the performance and efficiency of the complete application by increasing the performance and efficiency of the separate portions of the application.

The SPROC chip communicates with the microprocessor at high speed via the chip's parallel port. The SPROC chip appears as a memory mapped device occupying 16K bytes of microprocessor memory space. Refer to the appropriate *SPROC Programmable Signal Processor Data Sheet* for details on memory mapping the SPROC chip.

SMI supports applications using either Motorola-type (little endian) and Intel-type (big endian) byte ordering.

SMI Components

SMI includes the following components:

- A symbol translator (SymTran)

  The symbol translator uses the symbol file produced in the SPROClab development system and generates a data structure that mirrors the symbol file. This data structure allows for external C references to SPROC chip memory addresses. The symbol translator also produces a code file that allows you to locate the variables in the data structure at the appropriate SPROC chip memory map locations.

- The SPROC C function library (*sproclib.c*)

  The SPROC C function library provides the source code and header files for basic functions required to access the SPROC chip from a microprocessor. The library includes the SPROC chip load, reset, and start functions, and it includes the data conversion functions required for the microprocessor to correctly access and interpret the 24-bit fixed-point data type native to the SPROC chip.

Relationship to Other Products

SMI is delivered as a part of the SPROClab development system software. However, the components of SMI are not like other SPROClab software tools; you do not invoke them from the development system environment. The components of SMI provide source code and header files, in ANSI C, that you use outside of the SPROClab development system, in an embedded systems development environment. By accessing these files using the tools in your embedded systems development environment, you can create microprocessor applications in C that include the SPROC chip. Note that such applications require that you build a hardware memory-mapped system architecture.

The SMI Development Process

The process required to develop a microprocessor application that includes one or more SPROC chips as memory mapped devices requires work that must be done in the SPROClab development system, work that must be done in your embedded systems development environment, and other prerequisites.

In the SPROClab development system, you must:

- Create, debug, and tune the signal processing design using the SPROClab development system tools.
- Run SPROCbuild to produce the block file that the microprocessor application will use to load the signal processing design onto the SPROC chip.

In your embedded systems development environment, you must:

- Translate the symbol file into the data structure needed to provide microprocessor access to SPROC chip memory addresses.
- Copy the block file, the data structure, and all relevant file sections from the SPROC C function library into your applications work area.
- Create the microprocessor application.

In addition, you must also map the SPROC chip(s) into the microprocessor's memory.

Note that aspects of the microprocessor application depend on output from the signal processing design development process. If you develop the portion of the microprocessor application that deals with the SPROC chip in parallel with the signal processing design, be sure that you understand the relationship between the two processes, and the dependencies described in this manual. Otherwise, changes made to the signal processing design may affect work done on the microprocessor application.

Downloading and Debugging (SPROCdrive Interface - SDI)

The SPROCdrive interface (SDI) is the command-driven monitoring interface you use to load, run, and debug a signal processing design on the SPROC chip. SDI is part of the SPROClab development system ssoftware that executes on the development system PC.

The Hardware Link

In order to access the chip, SDI requires a hardware interface between the PC and the SPROCboard evaluation board where the chip is mounted. The SPROCbox interface unit provides that hardware link. It provides communications protocol conversion and relays user commands from SDI in a form that is appropriate for use by the chip.

The SPROCbox interface unit connects to the PC through a serial I/O port. The communication rate on this port can be 1200, 2400, 9600, or 19200 baud, depending on the setting of dip switches in the interface unit. SDI automatically determines the correct communications rate to match the rate selected by the interface unit DIP switches. The default setting is for 9600 baud.

The interface unit connects to the chip through the access port on the SPROCboard evaluation board. Communication over the access port is high-speed and synchronous.

Inputs to SDI

To load, run, and debug designs, SDI uses the following inputs:

- Your commands, including arguments, options, and switches
- The load file created by the "MakeLoad" utility.

  The load file includes the program and data generated from your signal-flow block diagram. The load file is written in Motorola S-record format to ensure data integrity.

- The symbol file created by SPROCbuild

  The symbol file includes the complete symbolic names of each input, output, and other node represented on your signal-flow diagram, and it maps these symbols to a specific memory address in the chip's memory.

  When you issue a command to read or modify the value of a symbol, SDI accesses the symbol file and converts the symbol name you enter into the actual on-chip memory address. SDI only supports the use of symbol names at the user interface. When communicating with the chip, SDI uses direct memory references.

Outputs from SDI

Because SDI is an interactive monitoring interface, most "outputs" from SDI are temporary values displayed on the development system PC or written to volatile chip memory. Changes made using SDI to modify the values of symbols do not effect the load file or the original signal-flow block diagram. They are only temporary changes to chip memory values. Whenever the chip is reinitialized, all memory values revert to the original values in the load file, i.e., those saved on your signal-flow diagram.

However, SDI can produce two types of output files:

- A log file

  The log file (*sdi_log.cmd*) comprises a complete record of all commands you enter during the SDI session; it is generated automatically and saved when you exit SDI.

You can execute the log file under SDI to entirely recreate the previous session. You can also use the file to help you update your signal-flow block diagram into a final "debugged" version by using the file as a reference that lists all the modifications you made to various design values.

- A capture file

  A capture file is a data file that stores a series of values for a specified symbol. You create a capture file by using the capture command and specifying the symbol name for the address at which to start and the number of memory locations to display and write to file.

  You can use the capture file with user-supplied tools to plot the values you captured and evaluate the performance of your design.

SDI Command List Summary:

SDI includes commands to provide the following functionality:

assemble/disassemble
break point commands
capture
comment
dos
halt/step/go
help
history
load
mode
pause
probe
read
repeat
show
start
test
verify
write

Overview of SDL

SPROC description language (SDL) is the language used to create high-level descriptions of arbitrarily complex signal processing systems to be implemented on the SPROC programmable signal processor.

SDL is a block-oriented language that supports hierarchical designs. Blocks may be of the following types:

- primitive
- hierarchical

Primitive Blocks

Primitive blocks, also called *asmblocks,* contain hardware-specific coding analogous to the firmware in a microprocessor system. Primitive blocks are written in assembly language. They may not contain references to other blocks.

Code for signal processing functions is written at the primitive level. These primitive blocks comprise the SPROCcells function library. They are optimized for the hardware, and efficiently implemented to extract maximum performance from the SPROC chip. Other primitive blocks include the "glue" blocks, or *phantoms,* required to provide control and synchronization functions for the multiple general signal processors (GSPs) on the SPROC chip.

Hierarchical Blocks

Hierarchical blocks contain references to other blocks, either primitive or hierarchical. The *sequence* (i.e., firing order) and *partitioning* (i.e., allocation over the GSPs and insertion of phantom blocks) of the referenced blocks in a hierarchical block is automatically determined.

A hierarchical block that is not referenced by any other block is a top-level block. There must be one and only one top-level block in a design.

Block Structure

A block contains a block name, block definition, and block body. The block name identifies the block for reference by hierarchical blocks. The block definition contains the following:

- an optional list of parameters
- a port list declaring the block's input and output signals
- optional general declarations for wires, variables, symbols, aliases, time zones, compute lines, and ports
- optional verify statements
- optional duration statements (primitive blocks only)

The block body contains references to other blocks (hierarchical blocks only) or assembly lines (primitive blocks and manual blocks only).

Instruction Format (Program RAM)I

Total Width	24 bits
Opcode	6 bits
Operand	15 bits
Address mode	3 bits, eight modes

Data Format

Total Width	24 bits
Range fractional	-2 to +1.999999762
Code	QQ.22, 2's complement with 22 bit fraction

Compiling SDL Files

SDL files are compiled by the SPROCbuild utility in the SPROClab development system. The utility includes three modules:

- The **MakeSDL** module

  The **MakeSDL** module prepares a top-level SDL file that completely describes the signal processing design using the netlist of the signal flow block diagram, primitive blocks from the function library, and other code and data files

- The **Schedule** module

  The **Schedule** module takes the top-level SDL file and breaks the file apart based on the resource and synchronization requirements of the blocks within the file. Resource requirements include program memory usage, data memory usage, and GSP cycles. Synchronization requirements include the determination of how and when blocks communicate data, and whether a block is asynchronous and independent of other blocks in the design.

  After breaking up the file to accommodate resource and synchronization requirements, the **Schedule** module partitions the file by blocks and locates the blocks to execute on the multiple GSPs on the SPROC chip using a proprietary partitioning algorithm. The module inserts phantom blocks as necessary to control the synchronization of the GSPs as they execute the design.

  Then the **Schedule** module generates a symbol table file that lists the physical RAM addresses on the SPROC chip for all the parameters, variables, and other elements in the design.

- The **MakeLoad** module

  The **MakeLoad** module converts the partitioned SDL file into a binary configuration file to run on the chip.

Basic Instruction Set

The following table lists the basic instruction set for the GSPs in the SPROC signal processor.

Basic GSP Instruction Set

OPCODE	OPERAND TYPE	DESCRIPTION
add	source	Add without carry. Load operand into ALU and sum with contents of accumulator. Result is stored in the accumulator
adc	source	Add with carry.
and	source	AND contents of accumulator with operand. Result is stored in accumulator.
asl	none	Arithmetically shift the accumulator contents 1 bit to the left and store the result in the accumulator. The most significant bit (msb) is shifted into the carry bit C and a zero is shifted in the least significant bit (lsb) of the accumulator.
asr	none	Arithmetically shift the accumulator contents 1 bit to the right and store the result in the accumulator. The lsb is shifted into the carry bit C, and the msb is held constant, (sign extended).
clc	none	Clear carry bit of status register.
cmp	source	Compare operand with accumulator contents and update the status register. Accumulator is unmodified by a compare instruction.
djne	source	Test loop flag, jump not equal to zero to specified operand address, then post decrement loop register.
jmp	source	Unconditional jump to operand address in the program RAM. Execution continues from the operand address.

Basic GSP Instruction Set (Continued)

OPCODE	OPERAND TYPE	DESCRIPTION		
jxx	source	Jump on condition code true.		
		xx	**CONDITION**	**TRUE CONDITION**
		cc	Carry Clear	~CF
		cs	Carry Set	CF
		lf	Loop Flag Set	LF
		mf	Multiplier Overflow Flag Set	MF
		ne	ZF Clear	~ZF
		ov	Overflow	OF
		si	Sign	SF
		eq	same as ZE	ZF
		ge	>=	(OF & SF) | (~OF & ~SF)
		ze	Zero/Equal	ZF
		le	<=	(~OF & SF) | (OF & ~SF) | ZF
		gt	>	~ZF & ((OF & SF) | (~OF & ~SF))
		lt	<	(~OF & SF) | (OF & ~SF)
		wf	Wait Flag Set	WF

Basic GSP Instruction Set (Continued)

OPCODE	OPERAND TYPE	DESCRIPTION
ldr	source	Load destination register (r) with operand.
ldy	source	Alias for mpy
mac	source	Load Y register of multiplier with operand value and execute the multiply/accumulate operation which adds the multiplication result to the contents of the M register. There is a two cycle latency before the result is available. The X register can be loaded with a new value during this two cycle period.
mpy	source	Load Y register of multiplier with operand value, and execute the multiplication operation, placing the result in the M register. There is a two cycle latency before the result is available. The X register can be loaded with a new value during this two cycle period.
nop	none	No operation.
not	none	Perform a one's complement of accumulator. Result is stored in the accumulator.
ora	source	OR contents of accumulator with operand. Result is stored in accumulator.
rol	none	Rotate accumulator contents left 1 bit through carry.
ror	none	Rotate accumulator contents right 1 bit through carry.
sec	none	Set carry bit of status register.
str	destination	Store contents of register (r) at destination address.

Basic GSP Instruction Set (Continued)

OPCODE	OPERAND TYPE	DESCRIPTION
sub	source	Subtract without carry. Load operand into ALU register and subtract from accumulator. Result is stored in the accumulator register.
subc	source	Subtract with carry.
xor	source	Exclusive OR contents of accumulator with operand. Result is stored in accumulator.

SPROCsim - Simulator

SPROCsim ssoftware simulates the physical SPROC chip, facilitating the development of custom cell libraries. SPROCsim implements a virtual SPROC using advanced software techniques. The virtual SPROC computes results which are identical to the physical SPROC, but performs these computations under the control of a command driven user interface. In contrast with the actual hardware, SPROCsim provides complete access to all registers, flags, memory, and internal status information. The virtual SPROC supports single stepping, continuous execution, and breakpoints.

Breakpoints can be set to occur whenever a particular instruction is executed, whenever a data location is accessed, whenever a range of data locations are accessed, or when a register or memory location takes on a particular value.

Command scripting allows the creation of command sequences, much like a batch file, useful in a particular simulation environment as well as allowing unattended operation.

Input/output to the virtual SPROC's serial ports and parallel port is performed using files. These files can be created using a text editor and viewed using a standard plotting package.

Source level debugging provides a correspondence between the SPROC Description Language (SDL) and the actual memory locations. These allow the user to refer to variable names with the names present in the actual design instead of as raw hexadecimal addresses.

All values are displayed in text windows. These text windows allow related information to be displayed in non-overlapping windows using ASCII characters for display.

The simulator is available as an optional addition to SPROClab development system.

APPLICATION INDEX

Section 4
Applications

Developing a Natural Logarithm Cell
Part 1 of 2
by Chester Nowicki

Introduction

A SPROC library cell was required for the natural logarithm function. This application note, part 1 of a two-part set, illustrates how a natural log cell was devised and shows implementation tradeoffs and results.

For the fixed point architecture of the SPROC general signal processors (GSPs), the cell would be constrained to operations in a 24-bit fixed point binary number system, with 22 bits following the binary point and two's complement representation. Computations were performed on a Sun SPARCstation SLC, using "C" language programs with 32-bit floating point variables. The final algorithm was run on SPROClab hardware, and the results were captured into a data file in order to compare actual performance to predicted performance.

Basic Approach

Realizeable input and output ranges impose important constraints upon a natural logarithm function for the SPROC library. The applicable two's complement QQ.22 number system affords a numeric range from +1.999999762 through -2.000000000. ln(1.999999762) establishes the top end of the input range by its argument, and the top end of the output range by its evaluation:

$$\ln(1.999999762) = 0.693147.$$

This result is shown to six decimal places. Solving the equation ln(in) = - 2.0 establishes the bottom end of the input range:

$$in = \exp(\ln(in)) = \exp(-2.0) = 0.135335$$

This result is also shown evaluated to six decimal places. Overall, the input range must therefore be limited to values of in, such that 0.135335 <= in <= 1.999999762. The span of the input range is less than 2.

A candidate for a mathematical generating function is the Maclaurin's series for ln(1+x), which converges for values of x in the interval (-1,+1). The series is represented as the infinite sum of terms:

$$\ln(1+x) = x - x^2/2 + x^3/3 - x^4/4 + \ldots -(-x)^k/k + \ldots.$$

Relative to input:

$$in = 1 + x \text{ (i.e., } x = in - 1).$$

For the SPROC input range $0.135335 <= in <= 1.999999762$, the transformation conveniently yields a range for x such that $-0.8646647 <= x <= 0.999999762$, which is a subset of the interval over which the Maclaurin's series converges.

Accuracy versus Execution Efficiency

Ideally, the natural logarithm routine should execute in as few cycles as possible with excellent accuracy. Limiting the implementation to a finite number of terms trades accuracy of the calculation against execution time. Additionally, for a polynomial series expression truncated to finite degree, the truncated series can be optimized by application of a Chebyschev polynomial to achieve an accuracy with least mean square error over the nominal interval $-1 < x < +1$.

Programs were written in the "C" language to simulate computational algorithms with print statements formatted to present results to six decimal digits. Figure 1 shows code for the basic truncated series, for the case of degree 9. Key points of this algorithm are the nested multiplication and the use of "minus x", that is:

$$x = in - 1 \text{ implies } (-x) = 1 - in$$

The truncated series can then be written as:

$$-(-x)(1 + (-x)(1/2 + (-x)(1/3 + (-x)(1/4 + \ldots + (-x)(1/k))\ldots.)$$

whereby terms in odd powers of x are positive and terms in even powers of x are negative. For notational convenience only, "x" is used to represent negative x or "minus x" in the "C" language programs.

The program in Figure 1 was extended for computations with truncated series of six through nine terms. The revised program, shown in Figure 2, further calculates the difference between the truncated series computation and the value for the built-in natural logarithm function; ideally, the difference would be zero for each argument. One line of code does the actual computation for each truncated series. The results are given in Table 1.

From Table 1, improvements can be seen at the bottom end of the argument range for an increasing number of terms (higher order) with the truncated polynomials; however, the top end is somewhat worse for nine terms than for eight terms. In the middle of the range, accuracy is clearly good to six decimal places. This center range accuracy corresponds to 20 binary places beyond the binary point for QQ.22 fixed point binary representation. For adjacent terms k and k+1 in the Maclaurin's series, the corresponding coefficients Ck and Ck+1 have this relationship:

$$Ck+1 = Ck * k/(k+1)$$

Because the ratio between the coefficients of these terms approaches unity for increasing k, contributions of larger valued arguments (magnitude of x nearing unity) at the ends of the ranges diminish less rapidly than those of smaller arguments (magnitude of x nearing zero) near the center of the range. For example, the argument x = 1/10 decreases its contribution by an order of magnitude for each term of increasing order, whereas the argument x = 3/4 decreases its contribution by a similar factor only after an increase in order by 8. That is:

$$(x)*(x\^{}1) = 0.1x \text{ for } x = 1/10$$

and: $(x)*(x\^{}8) = 0.100113x \text{ for } x = 0.75.$

Consequently, improvement in accuracy could be expected to be small for each term added beyond order 9 in the truncated series. In fact, the reversal in sign of the coefficient of the ninth order term accounts for the slightly worse accuracy at the top end of the input range for the ninth order truncated series than for the eighth order truncated series. At the bottom end of the range, the arguments x with negative signs actually tend to reduce the error in accuracy. Two other means for further improvement would be considered next, Chebychev correction polynomials and range compression.

Chebyshev Improvement

Table 2 lists Chebyshev polynomials, designated Vn(x), for degrees 0 through 10. Each polynomial can be applied against a truncated series of the given degree to yield a revised truncated polynomial expression that improves overall accuracy over the interval (-1,+1) in a least mean square error analysis. For degrees 0 and 1 the correction polynomials are trivial.

To create a Chebyshev corrected truncated series of degree n, the Chebyshev polynomial of degree n+1 is multiplied by a constant, then the result is subtracted from the truncated series of degree n+1, that is:

fc(x) = f(x) - c*Vn+1(x)

where:

fc(x) is the Chebyshev corrected truncated series of degree n
f(x) is the truncated series of degree n+1
c is a constant chosen such that terms of degree n+1 cancel
Vn+1(x) is the Chebyshev polynomial of degree n+1

Example Chebyshev Series Correction

For a truncated series of degree 6, V7(x) is applied:

fc(x) = x - x^2/2 + x^3/3 - x^4/4 + x^5/5 - x^6/6 + x^7/7
-[-7cx + 56cx^3 - 112cx^5 + 64cx^7]

Because the 7th order terms are to drop out:

x^7/7 - 64cx^7 = 0

c = 1/(7 * 64) = 1/448

Thus by substitution and factoring:

fc(x) = -(-x)(65/64 + (-x)(1/2 + (-x)(5/24 + (-x)(1/4 + (-x)(9/20
+ (-x)(1/6))))))

For a given value of x, the calculation uses "minus x" (i.e., -x) in a nested product. Using "x" to represent (-x) results in the expression for calc0 shown in the program in figure 2.

A review of the results in Table 1 under the headings Cheb7, Cheb8 and Cheb9 reveals that improvement in accuracy occurs at the top and bottom ends of the input range at the expense of significantly reduced accuracy in the center. Note that each Chebk column in the table represents the results of calculations using the Chebyshev corrected, truncated series of order n, for which n + 1 = k. Comparison of the Cheb7, Cheb8 and Cheb9 results to those for truncated series with seven, eight, and nine terms reveals that each Chebyshev corrected truncated series yields accuracy at the ends of the input range slightly better than that for the uncorrected truncated series of the next higher order.

Range Compression

With the accuracy of calculations using truncated series orders of magnitude better in the center of the input range (approaching 1.0), it was feasible to improve accuracy towards the ends of the input range by application of the additive properties of natural logarithms. The relevant properties are as follows:

$$\ln(x) = \ln(2/2 * x) = \ln(2 * x * 1/2)$$

$$\ln(x) = \ln(2) + \ln(x/2) = \ln(2x) + \ln(1/2) = \ln(2x) - \ln(2)$$

The strategy required involves doubling low end numbers or halving high end numbers, followed by the natural log computation upon the resulting number and the addition of a natural log correction.

Reinspection of the results in Table 1 yields the observations that top end accuracy decreases markedly for arguments above 1.35, and bottom end accuracy decreases marked below 0.7. Half of 1.35 is almost 0.7; equivalently, twice 0.7 is almost 1.35. These two numbers will be the range break points.

The program in Figure 3 was devised to evaluate the effects of compressing the end range input values and adding a correction for each of the previously illustrated truncated series computation schemes. After some adjustments in the choices of low_number and high_number, the results in Table 3 were produced.

Table 3 shows dramatically improved accuracy for natural logarithms computed at the top end, as well as towards the bottom end, of the input range. Even though improved results also occur for the Chebyshev-corrected truncated series, overall best performance is evident in the column for the eight term (8th order) truncated series. Table 4 expresses the error results of Table 3 as percentage error by dividing each error in Table 3 by the corresponding actual natural logarithm, and multiplying by 100.

Cell Algorithm and Code

The decisions established above generally define requirements upon the algorithm for the natural logarithm cell. Part 2 of this two-part application note series spells out the algorithm, relates general information relevant to producing in-line and subroutine forms of code, goes line-by-line through the in-line code, and highlights the differences between the in-line code and the subroutine code. The name of the final cell is ln.sdl [the sdl extension refers to SPROC Description Language].

SPROC Cell Performance

With the ln.sdl cell installed in a SPROClab environment, a source file was presented to the cell for both in-line and subroutine instantiations, and the output captured to a file. The arguments in the source file were essentially the same as those used in Table 1, Table 3 and Table 4, but reflect the effects of binary approximation to decimal numbers arising from use of 24 bit fixed point numbers. These arguments and the captured results were used to determine the accuracy of the cell achieved in SPROC hardware, relative to the Sun SPARCstation SLC "built-in" natural logarithm function. Table 5 gives the arguments, SPROC calculated natural logarithm, the error (SPROC natural logarithm calculation - Sun natural logarithm calculation) and the error expressed as a percentage. Comparison of the entries in the Error column of Table 5 with those in the eight terms column of Table 3 shows almost identical results. Comparison of the calculated percentage errors in Table 5 to those in the eight terms column of Table 4 also shows pretty close agreement in results.

Raw error in the range from 0.35 through 1.999999762 was less than 0.000007. Overall, percentage error in the natural logarithm cell calculations was under 0.03 in the argument range from 0.25 through 1.999999762. For argument 0.15, the percentage error was 0.65. These results agreed extremely well with the computed results from simulations with the Sun SPARCstation SLC.

Summary

The need for a natural logarithm function for the SPROC cell library led to the work illustrated in Part 1 and Part 2 of this two-part application note set. In this first part of the set a basic approach and steps in defining an algorithm for the function were given. Programs in the 'C' language for assessing accuracy of truncated series, with or without Chebyshev correction and with or without a range compression technique, were recounted and the results presented. In the second part of the set, the resulting algorithm was described, and its translation into code discussed in line by line detail. Information on how to write a SPROC cell was related for both in-line and subroutine code forms.

Error achieved by the implemented cell over the range from 0.35 through 1.999999762 was within 0.000007 units, as a difference between the SPROC calculated natural log and the "built-in" function for a Sun SPARCstation SLC. This agreed almost identically with computations run solely on the SPARCstation.

Table 1. Error for Truncated Series Calculations

_arg	6 terms	7 terms	8 terms	9 terms	Cheb7	Cheb8	Cheb9
2.00	-0.076481	0.066377	-0.058623	-0.169734	0.064144	-0.057647	0.052054
1.95	-0.054673	0.045089	-0.037838	-0.111552	0.046444	-0.038644	0.032606
1.90	-0.038354	0.029974	-0.023835	-0.071664	0.032206	-0.024707	0.019476
1.85	-0.026346	0.019451	-0.014611	-0.044887	0.021096	-0.014875	0.011004
1.80	-0.017675	0.012285	-0.008687	-0.027328	0.012745	-0.008275	0.005844
1.75	-0.011545	0.007525	-0.004989	-0.016113	0.006766	-0.004133	0.002930
1.70	-0.007314	0.004451	-0.002755	-0.009161	0.002766	-0.001782	0.001450
1.65	-0.004474	0.002529	-0.001454	-0.004995	0.000361	-0.000661	0.000810
1.60	-0.002628	0.001371	-0.000728	-0.002594	-0.000813	-0.000316	0.000597
1.55	-0.001471	0.000704	-0.000343	-0.001273	-0.001092	-0.000395	0.000544
1.50	-0.000778	0.000338	-0.000150	-0.000584	-0.000778	-0.000638	0.000501
1.45	-0.000384	0.000150	-0.000060	-0.000247	-0.000130	-0.000870	0.000403
1.40	-0.000174	0.000061	-0.000021	-0.000094	0.000636	-0.000987	0.000239
1.35	-0.000070	0.000021	-0.000007	-0.000032	0.001352	-0.000945	0.000035
1.30	-0.000025	0.000006	-0.000002	-0.000009	0.001895	-0.000746	-0.000168
1.25	-0.000007	0.000002	-0.000000	-0.000002	0.002190	-0.000426	-0.000331
1.20	-0.000002	0.000000	-0.000000	-0.000000	0.002203	-0.000039	-0.000421
1.15	-0.000000	0.000000	0.000000	-0.000000	0.001941	0.000350	-0.000424
1.10	-0.000000	-0.000000	-0.000000	-0.000000	0.001440	0.000679	-0.000340
1.05	-0.000000	-0.000000	-0.000000	-0.000000	0.000766	0.000899	-0.000189
1.00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000977	-0.000000
0.95	0.000000	0.000000	0.000000	0.000000	-0.000766	0.000899	0.000189
0.90	0.000000	-0.000000	-0.000000	-0.000000	-0.001440	0.000679	0.000340
0.85	0.000000	0.000000	0.000000	-0.000000	-0.001941	0.000350	0.000424
0.80	0.000002	0.000000	0.000000	-0.000000	-0.002203	-0.000039	0.000421
0.75	0.000011	0.000002	0.000001	-0.000001	-0.002186	-0.000425	0.000331
0.70	0.000042	0.000011	0.000003	-0.000004	-0.001878	-0.000741	0.000170
0.65	0.000133	0.000041	0.000013	-0.000012	-0.001290	-0.000925	-0.000029
0.60	0.000362	0.000128	0.000046	-0.000027	-0.000448	-0.000920	-0.000215
0.55	0.000886	0.000352	0.000142	-0.000045	0.000632	-0.000668	-0.000321
0.50	0.002001	0.000885	0.000397	-0.000037	0.002001	-0.000091	-0.000254
0.45	0.004244	0.002069	0.001022	0.000092	0.003865	0.000970	0.000136
0.40	0.008563	0.004564	0.002464	0.000598	0.006748	0.002876	0.001139
0.35	0.016628	0.009625	0.005642	0.002101	0.011793	0.006435	0.003378
0.30	0.031392	0.019627	0.012421	0.006016	0.021312	0.013395	0.008216
0.25	0.058193	0.039124	0.026610	0.015487	0.039883	0.027467	0.018691
0.20	0.107143	0.077184	0.056213	0.037572	0.076724	0.056625	0.041683
0.15	0.199059	0.153262	0.119201	0.088925	0.151617	0.118937	0.093587

Table 2. Chebyshev Polynomials

n	Vn(x)
0	1
1	x
2	2x^2 - 1
3	4x^3 - 3x
4	8x^4 - 8x^2 + 1
5	16x^5 - 20x^3 + 5x
6	32x^6 - 48x^4 + 18x^2 - 1
7	64x^7 - 112x^5 + 56x^3 - 7x
8	128x^8 - 256x^6 + 160x^4 - 32x^2 + 1
9	256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x
10	512x^10 - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1

Table 3. Error for Truncated Series Calculations with Compression

_arg	6 terms	7 terms	8 terms	9 terms	Cheb7	Cheb8	Cheb9
2.00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000977	0.000000
1.95	0.000000	0.000000	0.000000	0.000000	-0.000389	0.000957	0.000097
1.90	0.000000	0.000000	0.000000	0.000000	-0.000766	0.000899	0.000189
1.85	-0.000000	-0.000000	-0.000000	-0.000000	-0.001120	0.000806	0.000271
1.80	-0.000000	-0.000000	-0.000000	-0.000000	-0.001440	0.000679	0.000340
1.75	0.000000	0.000000	0.000000	0.000000	-0.001717	0.000525	0.000392
1.70	0.000000	0.000000	-0.000000	-0.000000	-0.001941	0.000350	0.000424
1.65	0.000001	0.000000	-0.000000	-0.000000	-0.002105	0.000159	0.000434
1.60	0.000002	0.000000	0.000000	-0.000000	-0.002203	-0.000039	0.000421
1.55	0.000005	0.000001	0.000000	-0.000001	-0.002231	-0.000236	0.000387
1.50	0.000011	0.000002	0.000001	-0.000001	-0.002186	-0.000425	0.000331
1.45	0.000022	0.000005	0.000001	-0.000002	-0.002068	-0.000596	0.000258
1.40	0.000042	0.000011	0.000003	-0.000004	-0.001878	-0.000741	0.000170
1.35	-0.000070	0.000021	-0.000007	-0.000032	0.001352	-0.000945	0.000035
1.30	-0.000025	0.000006	-0.000002	-0.000009	0.001895	-0.000746	-0.000168
1.25	-0.000007	0.000002	-0.000000	-0.000002	0.002190	-0.000426	-0.000331
1.20	-0.000002	0.000000	-0.000000	-0.000000	0.002203	-0.000039	-0.000421
1.15	-0.000000	0.000000	0.000000	-0.000000	0.001941	0.000350	-0.000424
1.10	-0.000000	-0.000000	-0.000000	-0.000000	0.001440	0.000679	-0.000340
1.05	-0.000000	-0.000000	-0.000000	-0.000000	0.000766	0.000899	-0.000189
1.00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000977	-0.000000
0.95	0.000000	0.000000	0.000000	0.000000	-0.000766	0.000899	0.000189
0.90	0.000000	-0.000000	-0.000000	-0.000000	-0.001440	0.000679	0.000340
0.85	0.000000	0.000000	0.000000	-0.000000	-0.001941	0.000350	0.000424
0.80	0.000002	0.000000	0.000000	-0.000000	-0.002203	-0.000039	0.000421
0.75	0.000011	0.000002	0.000001	-0.000001	-0.002186	-0.000425	0.000331
0.70	0.000042	0.000011	0.000003	-0.000004	-0.001878	-0.000741	0.000170
0.65	-0.000025	0.000006	-0.000002	-0.000009	0.001895	-0.000746	-0.000168
0.60	-0.000002	0.000000	-0.000000	-0.000000	0.002203	-0.000039	-0.000421
0.55	0.000000	0.000000	0.000000	0.000000	0.001440	0.000679	-0.000340
0.50	-0.000000	-0.000000	-0.000000	-0.000000	0.000000	0.000977	-0.000000
0.45	-0.000000	-0.000000	-0.000000	-0.000000	-0.001440	0.000679	0.000340
0.40	0.000002	0.000000	0.000000	-0.000000	-0.002203	-0.000039	0.000421
0.35	0.000042	0.000011	0.000003	-0.000004	-0.001878 -	0.000741	0.000170
0.30	0.000362	0.000128	0.000046	-0.000027	-0.000448	-0.000920	-0.000215
0.25	0.002001	0.000885	0.000397	-0.000037	0.002001	-0.000091	-0.000254
0.20	0.008562	0.004563	0.002464	0.000598	0.006748	0.002876	0.001139
0.15	0.031392	0.019627	0.012421	0.006016	0.021312	0.013395	0.008216

Table 4. Percentage Error for Truncated Series Calculations with Compression

arg	6 terms	7 terms	8 terms	9 terms	Cheb7	Cheb8	Cheb9
2.00	0.000000	0.000000	0.000000	0.000000	0.000000	0.140888	0.000000
1.95	0.000001	0.000001	0.000001	0.000001	-0.058199	0.143315	0.014502
1.90	0.000002	0.000002	0.000002	0.000002	-0.119294	0.140128	0.029426
1.85	-0.000003	-0.000003	-0.000003	-0.000003	-0.182018	0.130962	0.044120
1.80	-0.000001	-0.000003	-0.000004	-0.000004	-0.244987	0.115589	0.057907
1.75	0.000017	0.000004	0.000003	0.000002	-0.306732	0.093904	0.070079
1.70	0.000049	0.000003	-0.000003	-0.000008	-0.365717	0.065904	0.079896
1.65	0.000165	0.000021	-0.000001	-0.000020	-0.420280	0.031751	0.086660
1.60	0.000471	0.000082	0.000014	-0.000047	-0.468674	-0.008308	0.089669
1.55	0.001183	0.000232	0.000045	-0.000122	-0.509012	-0.053949	0.088228
1.50	0.002757	0.000606	0.000136	-0.000282	-0.539155	-0.104765	0.081674
1.45	0.006033	0.001461	0.000360	-0.000618	-0.556577	-0.160350	0.069314
1.40	0.012619	0.003334	0.000897	-0.001270	-0.558007	-0.220328	0.050403
1.35	-0.023473	0.007155	-0.002225	-0.010562	0.450500	-0.314867	0.011754
1.30	-0.009438	0.002470	-0.000656	-0.003434	0.722366	-0.284369	-0.064144
1.25	-0.003210	0.000697	-0.000158	-0.000917	0.981473	-0.190772	-0.148315
1.20	-0.000852	0.000151	-0.000025	-0.000181	1.208547	-0.021483	-0.231144
1.15	-0.000149	0.000025	0.000003	-0.000018	1.388538	0.250225	-0.303346
1.10	-0.000018	-0.000003	-0.000004	-0.000005	1.510837	0.712856	-0.357144
1.05	-0.000003	-0.000003	-0.000003	-0.000003	1.569376	1.843384	-0.387090
1.00	n/a	n/a	n/a	n/a	n/a	n/a	n/a
0.95	-0.000002	-0.000002	-0.000002	-0.000002	1.492794	-1.753507	-0.368200
0.90	-0.000012	0.000001	0.000002	0.000004	1.366726	-0.644879	-0.323072
0.85	-0.000170	-0.000020	-0.000001	0.000017	1.194069	-0.215196	-0.260874
0.80	-0.000992	-0.000173	-0.000030	0.000098	0.987165	0.017498	-0.188870
0.75	-0.003889	-0.000858	-0.000195	0.000394	0.759898	0.147654	-0.115117
0.70	-0.011901	-0.003141	-0.000842	0.001202	0.526408	0.207852	-0.047545
0.65	0.005750	-0.001503	0.000401	0.002094	-0.439948	0.173197	0.039068
0.60	0.000303	-0.000055	0.000008	0.000063	-0.431353	0.007668	0.082498
0.55	-0.000001	-0.000003	-0.000003	-0.000003	-0.240873	-0.113651	0.056935
0.50	0.000000	0.000000	0.000000	0.000000	-0.000004	-0.140889	0.000001
0.45	0.000001	0.000003	0.000003	0.000004	0.180336	-0.085086	-0.042625
0.40	-0.000245	-0.000045	-0.000010	0.000021	0.240399	0.004257	-0.045999
0.35	-0.004041	-0.001065	-0.000284	0.000410	0.178849	0.070619	-0.016152
0.30	-0.030036	-0.010596	-0.003792	0.002256	0.037243	0.076403	0.017839
0.25	-0.144364	-0.063858	-0.028636	0.002671	-0.144364	0.006587	0.018326
0.20	-0.532017	-0.283544	-0.153095	-0.037142	-0.419249	-0.178699	-0.070791
0.15	-1.654706	-1.034568	-0.654735	-0.317106	-1.123383	-0.706047	-0.433052

Table 5. SPROC Results

Argument	Natural Log	Error	% Error
1.999999762	0.693146706	-0.000000	-0.000051
1.849999905	0.615185499	-0.000000	-0.000014
1.799999952	0.587786436	-0.000000	-0.000034
1.750000000	0.559615612	-0.000000	-0.000031
1.699999809	0.530627966	-0.000000	-0.000033
1.649999857	0.500775099	-0.000000	-0.000020
1.599999905	0.470003605	0.000000	0.000007
1.549999952	0.438254833	-0.000000	-0.000015
1.500000000	0.405465603	0.000000	0.000122
1.449999809	0.371564627	0.000001	0.000323
1.399999857	0.336474895	0.000003	0.000821
1.349999905	0.300097942	-0.000007	-0.002192
1.299999952	0.262362719	-0.000002	-0.000575
1.250000000	0.223143339	-0.000000	-0.000095
1.199999809	0.182321548	0.000000	0.000083
1.149999857	0.139761925	0.000000	0.000076
1.099999905	0.095310211	0.000000	0.000124
1.049999952	0.048790216	0.000000	0.000200
1.000000000	0.000000000	0.000000	n/a
0.949999809	-0.051293373	0.000000	-0.000238
0.899999857	-0.105360508	0.000000	-0.000158
0.849999905	-0.162518978	0.000000	-0.000039
0.799999952	-0.223143339	0.000000	-0.000122
0.750000000	-0.287681341	0.000001	-0.000254
0.699999809	-0.356672049	0.000003	-0.000888
0.649999857	-0.430784464	-0.000001	0.000308
0.599999905	-0.510825396	0.000000	-0.000076
0.549999952	-0.597836733	0.000000	-0.000059
0.500000000	-0.693146944	0.000000	-0.000034
0.449999809	-0.798507690	0.000000	-0.000054
0.399999857	-0.916290522	0.000001	-0.000062
0.349999905	-1.049818993	0.000003	-0.000324
0.299999952	-1.203926802	0.000046	-0.003834
0.250000000	-1.385896921	0.000397	-0.028669
0.199999809	-1.606974125	0.002465	-0.153143
0.149999857	-1.884698868	0.012422	-0.654785

Figure 1 - C Program for Truncated Series Natural Logarithm Calculation

```
#include <stdio.h>
#include <math.h>
main()
{

    int constant, count;
    float in, out, x, calc;

 /* This program calculates natural logs of values piped from a source file. A
truncated series is used.
*/

    for (count = 0; count < 31; count++)

    {

        scanf("%f", &in)                    /* get input argument */
        x = 1 - in;                         /* transform in to "minus x" */
        calc = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/
                6.0 + x*(1.0/7.0 + x*(1.0/8.0) + x*(1.0/9.0))))))));
        out = log(in);                      /* use built-in function for comparison */
        printf("%2.9f % 2.9f\n", calc, out);

    }

}
```

Figure 2 - C Program for Accuracy of Natural Logarithm Series Calculations

```
#include <stdio.h>
#include <math.h>
main()
{

int count;
float in, out, x, calc6, calc7, calc8, calc9, calc0, calca, calcb;

/* This program is used to produce a set of results representing the differences
between natural log calculations and the built-in natural log function.

    Truncated series and Cbebyshev corrected truncated series are used.
    Arguments for in are looped to range from 2.0 through 0.15.
*/

    printf("_arg___6 terms___7 terms___8 terms___9
    terms____Cheb7_____Cheb8_____Cheb9\n\n"); /* headings */

    in = 2.05;                          /* initial value */

    for (count = 0; count < 38; count++)

    {

        in = in - 0.05;                 /* argument calculation */
        out = log(in);                  /* built-in function evaluation */
        x = 1.0 - in;                   /* "minus x" */

    calc6 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/
    6.0))))))-out;

    calc7 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/6.0
    + x*(1.0/7.0)))))))-out;

    calc8 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/6.0
    + x*(1.0/7.0 + x*(1.0/8.0))))))))-out;

    calc9 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/6.0
    + x*(1.0/7.0 + x*(1.0/8.0) + x*(1.0/9.0))))))))-out;

    calc0 = (-x)*(65.0/64.0 + x*(1.0/2.0 + x*(5.0/24.0 + x*(1.0/4.0 + x*(9.0/20.0 +
    x*(1.0/6.0))))))-out;

    calca = (-x)*(1.0 + x*(17.0/32.0 + x*(1.0/3.0 + x*(3.0/32.0 + x*(1.0/5.0 + x*(5.0/
    12.0 + x*(1.0/7.0))))))) + 1.0/1024.0 - out;

    calcb = (-x)*(255.0/256.0 + x*(1.0/2.0 + x*(37.0/96.0 + x*(1.0/4.0 + x*(1.0/80.0
    + x*(1.0/6.0 + x*(11.0/28.0 + x*(1.0/8.0)))))))) - out;

    printf("%5.2f%10.6f%10.6f%10.6f%10.6f%10.6f%10.6f%10.6f\n", in, calc6,
    calc7,calc8, calc9, calc0, calca, calcb);

    }

}
```

Figure 3 - C Program for Accuracy of Natural Logarithm Series Calculations that Incorporate a Compression Technique

```
#include <stdio.h>
#include <math.h>
main()
 {

    int count;
    float in, out, x, calc6, calc7, calc8, calc9, calc0, calca, calcb;
    float correction, low_number, high_number, ln_two;

/* This program is used to produce a set of results representing the differences
between natural log calculations and the built-in natural log function.

    Truncated series and Chebyshev corrected truncated series are used. Argu-
    ments for in are looped to range from 2.0 through 0.15 The input range is
    compressed at top and bottom to improve accuracy. */

    printf("_arg___6 terms___7 terms___8 terms___9
    terms____Cheb7_____Cheb8_____Cheb9\n\n");

    ln_two = log(2.0);
    low_number = 0.6875;
    high_number = 1.375;
    in = 2.05

    for (count = 0; count < 38; count++)
    {

      in = in - 0.05;                /* initial value */
      out = log(in);                  /* built-in function evaluation */
      x = 1.0 - in;                  /* "minus x" */
         correction = 0.0;           /* middle range, uncompressed */
      if (in < low_number)
      {
       x = 1.0 - in*2.0;             /* compress low end towards center */
         correction = -ln_two;       /* log(arg) = log(2*arg) - log(2)
   */
      }
      else if (in >= high_number)
      {
        x = 1.0 - in/2.0;               /* compress high end towards center */
            correction = ln_two;        /* log(arg) = log(arg/2) + log(2) */
        }

    calc6 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/
    6.0))))))-out + correction;

    calc7 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/6.0
    + x*(1.0/7.0)))))))-out + correction;

    calc8 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0 + x*(1.0/6.0
    + x*(1.0/7.0 + x*(1.0/8.0))))))))-out + correction;

    calc9 = (-x)*(1.0 + x*(1.0/2.0 + x*(1.0/3.0 + x*(1.0/4.0 + x*(1.0/5.0
     + x*(1.0/6.0 + x*(1.0/7.0 + x*(1.0/8.0) + x*(1.0/9.0))))))))-out + correction;

    calc0 = (-x)*(65.0/64.0 + x*(1.0/2.0 + x*(5.0/24.0 + x*(1.0/4.0 + x*(9.0/20.0 +
    x*(1.0/6.0))))))-out + correction;

     calca = (-x)*(1.0 + x*(17.0/32.0 + x*(1.0/3.0 + x*(3.0/32.0 + x*(1.0/5.0 + x*(5.0/
    12.0 + x*(1.0/7.0))))))) + 1.0/1024.0 - out + correction;

     calcb = (-x)*(255.0/256.0 + x*(1.0/2.0 + x*(37.0/96.0 + x*(1.0/4.0 + x*(1.0/80.0
    + x*(1.0/6.0 + x*(11.0/28.0 + x*(1.0/8.0)))))))) - out + correction;

    printf("%5.2f%10.6f%10.6f%10.6f%10.6f%10.6f%10.6f%10.6f\n", in, calc6,
    calc7, calc8, calc9, calc0, calca, calcb);

    }

}
```

Developing a Natural Logarithm Cell

PART 2 of 2

by Chester Nowicki

Introduction

This is part 2 of a two-part application note set concerning development of a natural logarithm cell. Part 2 highlights cell programming/coding conventions and techniques as applied to an algorithm for the natural logarithm. Considerations relevant to a subroutine version of this cell are also discussed.

Cell Algorithm

The decisions established in Part 1 of this application note set generally define requirements upon the algorithm for the natural logarithm cell. Specifically, the requirements are:

1. 8th order, eight term truncated Maclaurin's series
2. input range bottom limiting
3. top end input range compression
4. bottom end input range compression
5. correction for compression action.

In terms of cell operation, data flow conventions, SPROC general signal processor (GSP) architecture, the GSP instruction set, and constraints in the use of on-chip program and data random access memory (RAM) come into play. Input arguments and output arguments are presented as data stored in memory locations that correspond to wires in a schematic diagram. Refer to the *SPROCcells Function Library Reference Manual* for the icon of the natural logarithm cell, denoted ln.

By convention, the cell will be labeled ln.sdl, where sdl stands for "SPROC Description Language." The input wire, pin number 2, and the output wire, pin number 1, correspond to the input/output (I/O) argument list for the cell, which will be denoted "in" and "out", respectively. No parameters appear with the cell, because none are required for this function. With these points clear, the algorithm outline follows:

```
Read "in" (cell input wire)
Initialize correction to zero
If in < bottom, then in = bottom
If in <= low_number,
then
        in = in*2
        correction = -ln(2)
else if in >= high number,
then
        in = in/2
        correction = ln(2)
Save correction till end
Take two's complement of in
Add 1.0                                 // result is -x
Multiply 1/8 times (-x)
Add 1/7 to product
Multiply result by (-x) Add 1/6 to product
Multiply result by (-x)
Add 1/5 to product
Multiply result by (-x)
Add 1/4 to product
Multiply result by (-x)
Add 1/3 to product
Multiply result by (-x)
Add 1/2 to product
Multiply result by (-x)
Add 1.0 to product
Multiply result by (-x)
Subtract final product from correction
Write result to "out" (cell output wire)
```

A number of practical considerations apply to this, as well as to any algorithm, to be coded. They are briefly described in the following few paragraphs.

CONSTANTS: All in-line constants are limited to 15 bits, which may end up either right justified or left justified in the target data bit field (usually 24 bits). Constants are incorporated into program instructions as immediate operands, placed within a designated source bit field. All program instructions occupy one location in program RAM. For example, 1/4 = 0.25 is represented in a 15 bit field as 000100000000000; when left-justified in a target 24-bit field, the result is

000100000000000000000000, corresponding to 00.0100000000000000000000 and equating exactly to 0.25.

VARIABLES: A variable occupies one location in 24 bit data RAM. As wires, inputs and outputs are presented as variables to provide the full 24-bit dynamic range of the QQ.22 fixed point binary number system. When "constants" are needed with more than 15 bits of precision, variables must be used. For example, 1/5 = 0.2 is represented as 00.0011001100110011001100, which equates to 0.1999999809, decimal. If 0.2 is loaded as a constant left justified in a 24-bit field, the result is 00.00110011001100000000, which equates to 0.1999951172. Therefore, to achieve best accuracy, constants such as the coefficients 1/3, 1/5 and 1/7 should be set up as cell variables.

SYMBOLS: An in-line constant may be equated to a symbol. This improves readability of the code, but it must be remembered that constants are loaded as immediate arguments (either left or right justified), with only 15 bits of representation. By contrast, variables are treated as direct arguments for access to their contents; if the name of a variable is submitted in an immediate argument, the address of that variable is accessed. Symbols can also be used to initialize variables (24 bits).

TWO'S COMPLEMENT NOTATION: Negative values are represented in two's complement notation. This means that loading of the negation of a variable must be done either through a subtraction, or via a one's complement followed by an integer addition of 1. With symbolic constants, the two's complement can be indicated by preceding the symbol with a minus sign, in which case the scheduler does the computation as a step in the generation of machine code.

OPERAND TYPES: Ordinarily, concern about operand types is unnecessary; however, the cell programmer must explicitly select the appropriate multiplier output register segment to correspond with the intended operand types. For fixed point operands, the MH register must be designated, because it provides an offset to adjust for the implied final position of the binary point in the product. For strictly integer operands, the MHI or MLI register must be designated. For mixed type multiplications the MHI or MLI register should be designated, but the programmer must understand how the product is formed. For example, consider the multiplication of 0.125 by 6. The product of this multiplication remains within the 24-bit representational field (no overflow), and can therefore be interpreted as a fixed point result read out of the MLI register.

CONSTANT EXPRESSION EVALUATIONS: Consider a value for a coefficient, such as 1/7. How does the cell programmer conveniently represent such a value? Realizing that a fixed point computation is implied, he could carry out the division and use the fixed point decimal representation of the result. Alternatively, the scheduler will evaluate constant expressions, but operand type must be explicitly indicated and the result of each evaluation must lie in the interval [-2.0,+2.0]. Symbolic constants also may be used as arguments in expressions.

In view of the points just made, the following allocations were initially decided for application to the natural logarithm cell algorithm:

Variables

inv3 = 1.0/3
inv5 = 1.0/5
inv6 = 1.0/6
inv7 = 1.0/7
ln_two = 0.6931471814
mln_two = -0.6931471814

Symbolic constants

inv2 = 0.5 = 1.0/2
inv4 = 0.25 = 1.0/4
inv8 = 0.125 = 1.0/8
low_number = 0.6875
high_number = 1.375
bottom = 0.135375977

It should be noted that each of the symbolic constants, including bottom, need no more than 15 bits for accurate representation. The value for bottom, however, is a compromise to eliminate a variable. Although the natural logarithm of bottom is -1.999699357, the error in the natural logarithm calculation at the very bottom of the input range is greater than the difference achieved in representing bottom in 24 bits as a variable (the value would then be 0.135335207).

General Comments

A few general comments about writing cells are appropriate, prior to description of the actual code for the natural logarithm cell.

The listing for ln.sdl shows the structure and organization applied by STAR Semiconductor for cell source code. For convenience only, the listing does not show the CVS header with comments related to version control. The first section provides cell name, and brief descriptions of the cell's function, arguments, parameters and algorithm. The second section accommodates the in-line code version of the cell. When a subroutine form is applicable, a third section provides the subroutine call, and a final section accommodates the body of the subroutine. These last three sections occur in forms generally called "blocks" and specifically declared asmblock, callblock and subrblock, respectively, in the three instances illustrated in the cell listing.

The block declaration consists of one logical line that includes the cell name, a parameter list, and an argument list. The parameter list, contained in braces, establishes initialization values (for other than inputs and outputs) that the scheduler assigns when it produces machine code. The argument list, contained in parentheses, lists all variables used for cell input and output wires. In both cases, list entries are separated by commas; however, a semicolon is used to separate the argument list into an input list and an output list, in that order.

After the block declaration comes symbol definitions and variable definitions with initial variable assignments. [Note: Although in general the parameter list may be used to define initial values that can be applied to variables, none are needed for the natural logarithm cell. The parameters present values that the scheduler will assign if the schematic containing the cell does not contain explicit alternatives. For cells that include subroutine versions, the statement "%subr=default" is a mandatory component in the asmblock and subrblock parameter lists. Refer to the *SPROClab User's Guide* for further information regarding parameters.] Other statements may appear among or following these definitions; these are described in the *SPROC Description Language Reference Manual.* Comments demarcated by two methods may be incorporated. One method is the "C" language scheme of /* comment */. The other method is a double slash, (i.e., //) with the actual comment following it within the same line.

Star's conventions include a "Registers used" comment, to list specialized usage of any GSP registers in the cell implementation. Finally, a duration statement is required prior to the code itself. The duration statement gives the worst case number of instructions executed in any one pass through the cell, from its beginning to its end. By convention, the duration statement provides, as a comment, the total lines of code (instructions) in the subject block of the cell. [Note: In the cases of callblocks and subrblocks, register usage for both the callblock and subrblock are provided only in the callblock. Also, the duration statement for the subrblock always contains an argument of 0, whereas the duration statement for the callblock should contain the combined actual duration for the callblock and subrblock. Again by convention, the duration statement in the callblock includes total lines of code as a comment.]

Actual code instructions appear at the end of the block, and are contained between a "begin" statement and an "end" statement. Each line may include comments. Also, blank lines are permitted. Mnemonics for instructions are lower case, while register names in operands are upper case.

Cell Code: In-line

Examination of the ln.sdl cell listing reveals one major difference from what was presented in an earlier section. The algorithm for calculation of the truncated series is nested only for the last 7 terms. The reason for this difference will become clear in the discussion of details about the cell's in-line code.

The parameter list in the asmblock declaration contains no parameters, except "%subr=default," which is required for the scheduler to establish that a subroutine form might exist for use as an alternative to the in-line code. The argument list in the asmblock declaration shows one input and one output, respectively designated "in" and "out". Variables and symbols following the declaration meet criteria previously described.

The register usage comment cites specific usage of the X register. This is explained in subsequent detail. Lastly, before the code itself, the duration statement shows a duration of 47 instruction cycles, and 52 lines of code.

Following the begin statement, the first instruction reads the input argument presented to the cell. The second instruction sets the X register to a zero value initial correction, as noted in the register usage comment. This value of correction will apply if the value of "in" lies within the range not subject to compression.

The third instruction tests the A register for bottom of the range compliance against an immediate argument, and results in a jump to label "inrange" if the result is true. Failure of the test results in loading the A register with bottom as a replacement for the out of range input value, followed by a jump to label "low". This accomplishes the bottom range limiting function given in the algorithm. To save a cycle, the jump skips the next two instructions, which are not necessary with the input known to be less than the test case for that pair.

Like an "if-then" construct, the aforementioned compare instruction tests the contents of the A register against low_number to establish whether to compress the value towards the center of the range. The arithmetic shift left is equivalent to a multiply of the contents of A by 2, and minus ln(2) then replaces the correction held in the X register. The subsequent jmp instruction to label "not_high" avoids the next test and jump in order to save a cycle, when A is known to contain a number less than high_number.

The instruction comparing the contents of A to high_number and the following jump less than is an "else-if" construct, that establishes whether to compress a high end value towards the center of the range. Failure of the test is followed by an arithmetic shift right; this is equivalent to division of the contents of the A register by 2. ln(2) then replaces the correction held in the X register.

All of the above if-then, else-if decisions converge at the instruction following the label "not_high" so that the value of correction in register X gets stored to the variable named "correction." This frees the X register for its next use, cited in the register usage comment. To get minus in, the next two instructions effects a two's complement as a one's complement upon the contents of the A register followed by addition of an integer 1 [not 1.0]. Adding 1.0 to the result yields 1.0 - in, which is the needed "(-x)" [not to be confused with register X].

At label t8, "(-x)" is loaded into register X one time, for use in all subsequent multiplies. The first multiplication times inv8 forms the core of the 8th order term. In current hardware, the mpy instruction takes 3 cycles to execute. This means a latency of 2 additional instruction cycles precedes the presentation of the product in the multiplier output register. In general, the X register may be reloaded during the latency period; if no other useful instruction can be executed during the latency period, then one or more nop instructions must be inserted. [Note: It is possible to read the previous contents of the product register during the latency period; however, some caution is suggested for this practice, in case a single cycle multiply instruction is offered in future products]. At label t7, the core of the 7th order term, inv7, is placed in the A register, and a nop instruction follows. The fixed-point product, now available from the MH register, is added to the contents of the A register, which then are multiplied times the contents of the X register.

At label t6, inv3 is loaded into the A register; an arithmetic shift right divides it by 2 to yield the value of inv6. This procedure eliminates the need for a separate variable for the inv6 coefficient. The addition of the now available product to the contents of the A register prepares the way for the next nested multiplication operation. The contents of A are again multiplied by the contents of X.

Except for operand in the lda instruction, the nested multiplication procedure repeats identically from labels t5 to t1. At t1, the procedure is modified in a way to save a cycle that otherwise would be lost. If the nested product scheme were to end by finally adding 1.0 to the next to the last product, the following instruction sequence would be required:

```
t1:   lda 1.0
      nop
      add MH // 1.0 + (-x)(1/2 + (...
      mpy A // (-x)(1.0 + (-x)(1/2 + (...
      nop
      lda correction // correction in A
      sub MH // correction - (-x)(1.0 + (-x)(1/2 + (...
```

In this sequence of seven instructions, the result is equivalent to that achieved in the six instructions before the last, as given in the ln.sdl listing for the in-line code.

The final instruction stores the result, contained in the A register, to the variable corresponding to the cell output wire. The end statement closes the in-line code segment.

Cell Code: Subroutine

For almost all intents and purposes, the implementation of the natural logarithm algorithm in the subroutine form of the cell is identical to that described in the cell code discussion for the in-line cell. Therefore this section deals with general features unique to subroutine forms of cells, and with specific features relevant only to the differences between the in line and subroutine versions of the ln.sdl cell. These differences are best reviewed while examining the ln.sdl cell listing.

As for the asmblock segment for the in-line form, the parameter list in the callblock declaration contains only "%subr=default." This is required for the scheduler to verify that a subroutine form exists for use as an alternative to the in-line code. The argument list in the callblock declaration similarly shows one input and one output, respectively designated "in" and "out". Of the two variables that follow, only "correction" serves a purpose identical to that for the in-line version. It should be understood that it is strictly necessary to have among the callblock variables those unique to the call; this is to guarantee re-entrant code function for the subrblock.

The variable "return" is specifically for establishing the return address for the subroutine; its contents are equated in the callblock code to the address of the location just beyond the jump instruction, which invokes the actual subroutine code. By convention, the return variable is presented first in the list of variables, such that its address is the lowest in the block of variables set up for the callblock by the scheduler program.

The callblock code section is prefixed with a begin statement, and ends with an end statement, as it does for the asmblock as well as the subrblock. In general, the callblock code serves two purposes: (1) to support exchange of arguments (cell input and output) and (2) to set up management hooks for the subroutine. The first instruction and the last instruction, lda in and sta out, clearly address the first purpose. These two instructions are identical to the first and last instructions in the asmblock of the in-line form. The second and third instructions address the second purpose. As indicated in the register usage comment, the B register is designated for the base address of variables in the callblock. In this case, the address of the block is that of the variable "return," and is denoted by the expression "#return." With this address in the B register, subroutine code can access callblock variables through indirection. The jump instruction always has as its argument the form "cellname.$start", where the double quotes must be included in the operand. For ln.sdl, cellname is ln.

Inspection of the subrblock in the ln.sdl listing reveals that the subrblock declaration contains only the name of the cell, and no parameters or arguments. The variable and symbol list consists of the remaining variables and symbols in the in-line form that were not declared in the callblock; these meet criteria previously mentioned to support the algorithm. For those variables that were declared in the callblock, namely "return" and "correction," symbols alone are defined as shown in the listing, and for convenience, are repeated here:

```
symbol return=0, correction=1;
```

With the base address of the two callblock variables contained in the B register, the sum of the contents of B and the value of one of these designated symbols yields the address of the corresponding variable in the callblock. Thus, the variable can be accessed using indexed instructions for B register plus offset. In the code segment of the subrblock, three instructions of this type can be found, they are:

```
stx [B+correction]
lda [B+correction]
jmp [B+return]
```

The first two instructions ensure that the correction is unique to the instantiation of the callblock that invoked the subroutine. The third instruction similarly guarantees return to the address, symbolically designated "postamble", for that same callblock instantiation.

In terms of lines of code, the callblock adds two instructions and the subrblock adds one instruction as overhead for the call, the return, and the use of callblock variables by the subrblock code. Looking at the duration statement in the callblock, one sees "4+46". The 4 comes from the callblock and the 46 from the subrblock, yielding a total of 50 instruction cycles. Compared to the duration of 47 for the asmblock, the duration of the subroutine version reflects the addition of the three overhead instructions.

Summary

Based upon results of work recounted in Part 1 of this application note set, an algorithm was presented for a natural logarithm cell. General considerations for cell programming were related, and specific requirements for the natural logarithm cell were addressed. Actual code generated as an implementation of the algorithm was reviewed line by line for the in-line code form. A small modification to the algorithm for code and cycle time reduction was explained. Finally, differences applicable to the subroutine code form were presented.

Listing 1 - ln.sdl Code

```
/************************************************************

                         ln.sdl

************************************************************

Function:

   This code calculates the natural logarithm of the input using these
   first 8 terms of a series:

   ln(in) = ln(1+x) =

      x - x^2/2 + x^3/3 - x^4/4 + x^5/5 - x^6/6 + x^7/7 - x^8/8

Arguments:

   fixed in 0.135375977 <= in < 2.0 // input value
   fixed out -2.0 <= out <= 0.693115235 // resulting calculation

Parameters:

   none

Algorithm:

   The calculation range is restricted by hard limiting to a lowest value
   of 0.135375977, and reduced for greater accuracy at the ends by a
   divide by two at the top end, or a multiply by two at the bottom end. A
   correction is added to the natural log calculation.

   If in > 1.375, in = in/2 and correction = ln(2); if 0.135375977 <= in <
   0.6875, in = 2*in and correction = -ln(2); if 0.6875 <= in <= 1.375,
   correction = 0.0.

   The algorithmic calculation of ln is a nested product of sums:

   ln(1+x) =

   -(-x) - (-x)(-x)(1/2 + (-x)(1/3 + (-x)(1/4 + (-x)(1/5 + (-x)(1/6 +
      (-x)(1/7 + (-x)/8))))))

   Since in = 1 + x, x = in - 1 and -x = -(in - 1) = -in + 1.
```

```
Notes:

1)  Variables are used to guarantee 24 bit representation and precision
    for those constants 1/n that are not represented in 13 binary places.

2)  The calculation is carried out for 0.13535375977 <= in < 2.0,
    sinceln(0.135375977) = -2.0; for in< 0.135375977, in is forced to
    0.135375977.

*/

/*******************/
/* inline code */
/*******************/

asmblock ln { %subr=default } (in; out)

variable fixed ln_two = 0.693147181;
variable fixed mln_two = -0.6931471814;
symbol low_number = 0.6875;
symbol high_number = 1.375;
symbol bottom = 0.135375977;

variable fixed correction;
symbol inv8=0.125;                  // 1.0/8
variable fixed inv7 = 1.0/7;
variable fixed inv5 = 0.2;          // 1.0/5
symbol inv4 = 0.25;                 // 1.0/4
variable fixed inv3 = 1.0/3;
symbol inv2 = 0.5;                  // 1.0/2

           /* Registers used:

fixed X             initially holds correction; then holds (-x)

           */

duration 47;                        // total lines of code: 52

begin
     lda in
     ldx #0                         // initial correction
     cmp #bottom                    // antiln(-2.0) threshold at
                                       bottom of range
     jge inrange lda #bottom        // use antiln(-2.0) for below
                                       range values
```

```
        jmp low                             // skip next test to save a cycle
//
inrange:
        cmp #low_number
        jgt not_low
low:    asl                                 // multiply by two
        ldx mln_two                         // correction is ln(1/2) = -ln(2)
        jmp not_high                        // skip next test to save a cycle
not_low:
        cmp #high_number
        jlt not_high
        asr                                 // divide by two
        ldx ln_two                          // correction is ln(2)
not_high:
        stx correction                      // store final correction
        not
        add #1                              // two's complement
        add #1.0                            // -in + 1.0 = -(in - 1.0) = -x
t8:     ldx A                               // -x into X
        mpy #inv8                           // (-x)(1/8)
t7:     lda inv7
        nop
        add MH                              // 1/7 + (-x)(1/8)
        mpy A
t6:     lda inv3                            // 1/3 in A
        asr                                 // 1/3 divided by 2 = 1/6
        add MH                              // 1/6 + (-x)(1/7 + (-x)(1/8))
        mpy A
t5:     lda inv5
        nop
        add MH                              // 1/5 + (-x)(1/6 + (-x)(1/7 + (-x)
                                               (1/8)))

        mpy A
t4:     lda #inv4
        nop
        add MH                              // 1/4 + (-x)(1/5 + (-x)(1/6 + (-x)
                                               (1/7 +
                                            // (-x)(1/8))))

        mpy A
t3:     lda inv3
        nop
        add MH                              // 1/3 + (-x)(1/4 + ...
        mpy A
```

Section 4: Applications

```
t2:   lda #inv2
      nop
      add MH                         // 1/2 + (-x)(1/3 + ...
      mpy A
t1:   lda correction                 // correction into A
      sub X                          // correction - (-x) in A
      mpy MH                         // (-x)(-x)(1/2 + (...
      nop
      nop
      sub MH                         // correction - (-x) - (-x)(-x)(1/2
+
                                     // (... + (-x)(1/8)))))) in A
      sta out
//
end

/*******************/
/* subroutine call */
/*******************/

callblock ln { %subr=default } (in; out)

variable hex return=postamble;
variable fixed correction;

           /* Registers used:

hex B               base address of variables in callblock
fixed X             initially holds correction; then holds (-x)

           */

duration 4+46;                       // total lines of code: 4 + 51

begin
      lda in
      ldb #return                    // provide for passing return_spot
      jmp "ln.$start"
postamble:
sta out
end
```

```
/*******************/
/* subroutine body */
/*******************/

subrblock ln {} ()

symbol return=0, correction=1;
variable fixed ln_two = 0.693147181;
variable fixed mln_two = -0.6931471814;
symbol low_number = 0.6875;
symbol high_number = 1.375;
symbol bottom = 0.135375977;

symbol inv8 = 0.125;                 // 1.0/8
variable fixed inv7 = 1.0/7;
variable fixed inv5 = 0.2;           // 1.0/5
symbol inv4 = 0.25;                  // 1.0/4
variable fixed inv3 = 1.0/3;
symbol inv2 = 0.5; // 1.0/2

duration 0;

begin

     ldx #0                          // initial correction
     cmp #bottom                     // antiln(-2.0) threshold at bottom
                                        of range

     jge inrange
     lda #bottom                     // use antiln(-2.0) for below-range
                                        values

     jmp low                         // skip next test to save a cycle
//
inrange:
     cmp #low_number
     jgt not_low
low: asl                             // multiply by two
     ldx mln_two                     // correction is ln(1/2) = -ln(2)
     jmp not_high                    // skip next test to save a cycle
not_low:
     cmp #high_number
     jlt not_high
     asr                             // divide by two
     ldx ln_two                      // correction is ln(2)
not_high:
     stx[B+correction]               // store final correction
```

Section 4: Applications

```
        not
        add #1                      // two's complement
        add #1.0                    // -in + 1.0 = -(in - 1.0) = -x
t8:     ldx A                       // -x into X
        mpy #inv8                   // (-x)(1/8)
t7:     lda inv7
        nop
        add MH                      // 1/7 + (-x)(1/8)
        mpy A
t6:     lda inv3                    // 1/3 in A
        asr                         // 1/3 divided by 2 = 1/6
        add MH                      // 1/6 + (-x)(1/7 + (-x)(1/8))
        mpy A
t5:     lda inv5
        nop
        add MH                      // 1/5 + (-x)(1/6 + (-x)(1/7 + (-x)
                                       (1/8)))
        mpy A
t4:     lda #inv4
        nop
        add MH                      // 1/4 + (-x)(1/5 + (-x)(1/6 + (-x)
                                       (1/7 +
                                    // (-x)(1/8))))
        mpy A
t3:     lda inv3
        nop
        add MH                      // 1/3 + (-x)(1/4 + ...
        mpy A
t2:     lda #inv2
        nop
        add MH                      // 1/2 + (-x)(1/3 + ...
        mpy A
t1:     lda [B+correction]          // correction into A
        sub X                       // correction - (-x) in A
        mpy MH                      // (-x)(-x)(1/2 + (...
        nop
        nop
        sub MH                      // correction - (-x) - (-x)(-x)(1/2 +
                                    // (... + (-x)(1/8))))) in A
        jmp [B+return]
//
end
```

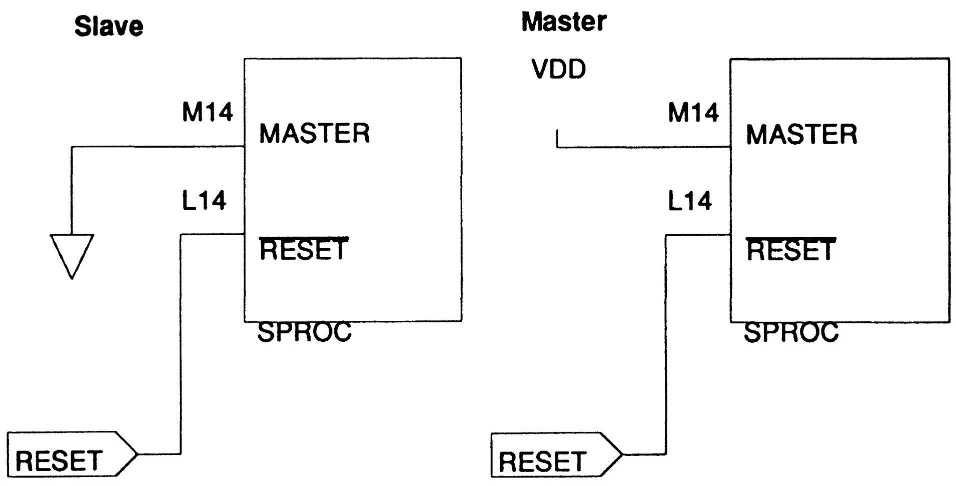
SPROC I/O Operation

by Frank Sgammato

This paper will describe the SPROC, the available options and selections for a variety of applications. It will discuss configuration registers, modes, and I/O.

SPROC Modes of Operation

The SPROC can be configured into one of two operating modes, master or slave, by controlling the input pin [MASTER]. If slave mode is to be used, three additional input pins must be driven - - MODE[2:0]. These pins control the width of the parallel port (in slave mode) and may be changed dynamically in operation. The section on memory models provides further details.



In master mode, the SPROC is the "ACTIVE" device. Initialization is achieved by the SPROC automatically, as described in the section on boot. In slave mode, power-on initialization is achieved with the aid of a controlling microprocessor (MPU/MCU). The parallel port of the SPROC is the likely path to interface on

MPU/MCU with the SPROC memory. The MPU/MCU interface is discussed in further detail in the parallel port section. The access port (serial interface) may also be used, as described in more detail in the access port section. The code and data blocks are loaded to the internal caches of the SPROC. In slave mode, referred to as the coprocessor mode, the SPROC is "PASSIVE" on the bus.

Boot (Master Mode)

Internally to the SPROC reside two programs, "**Self-Boot**" and "**Break**". Self-boot is only used by master SPROCs. The second section may be used anytime during operation regardless of mode (refer to the "**Break**" section).

Self-boot is a program which permanently resides in the SPROC ROM. It will be executed only upon reset and only in master mode. Immediately after reset, the code bus for the GSPs is disconnected from the normal code RAM (000h to 3FFh) and connected to the internal ROM bus. The assembly code contained in the boot section instructs one GSP to perform all the data movement. In the case where slave SPROCs are connected to the master, that same GSP will also load their memory spaces.

Note: Since the GSP does not depend upon the code RAM for its instructions, it may copy data to that area along with data RAM.

Boot loading is performed in blocks with **bus width** = **8 bits** (though all words are 24 bit), **MSB first**, and wait states = 7. The protocol is block size first, block destination followed by data. Blocks are read from an external memory device (accessed with CS high) and copied to SPROC addresses with CS low until a block size of zero is encountered. The last instruction of boot commences operation.

Break (Master or Slave)

Break is a program which permanently resides in SPROC ROM, and can be executed at any time by issuing a halt command. This forces the GSPs to disconnect from the code RAM and reconnect to the ROM (similar to boot) and start executing instructions at the beginning of the break section. The code in the break section directs the GSPs to copy their registers and load their registers to predetermined locations in data RAM.

This mechanism may be automatically used by the SPROClab development system under SPROCdrive and can be used for debugging. This feature may be enabled or disabled. If enabled, 15 data locations are reserved for each GSP. With the feature disabled, the user regains the use of the those locations (for the final design).

An example of a SPROCdrive command sequence is as follows:

```
halt <CR>
read A G3 F7 <CR>
```

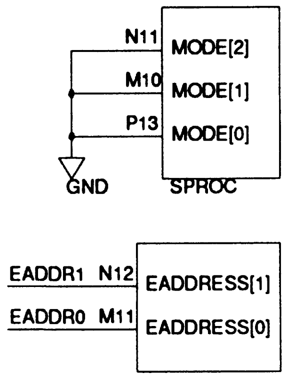
The contents of the GSP#3 accumulator will be displayed on the development system's screen in fixed point format with seven decimal places.

```
halt <CR>
read  *G* h <CR>
```

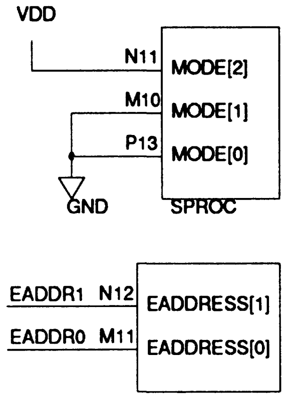
This will display the register contents for all GSPs in hexadecimal format.

Memory Models of the SPROC (Slave Mode)

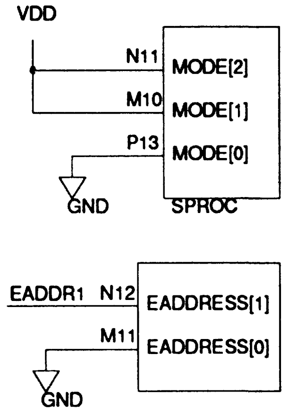
In slave mode the bus width and byte order are determined with the **MODE[2:0]** pins. When 24-bit bus is selected, the transfer requires only one write (or read) and no extra address lines are needed. If a 16-bit bus is used, the transfer requires two writes (or reads) and one additional address bit (EADDR1) is required. When 8-bit mode is used, transfers require three writes (or reads) and two additional address bits (EADDR1, EADDR0) are required.



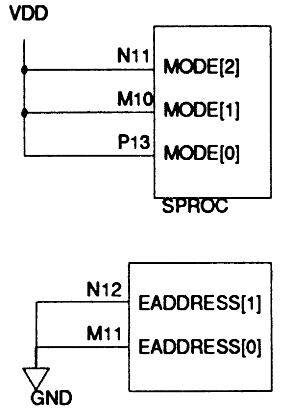
**8-bit Mode
MSB first**



**8-bit Mode
LSB first**



**16-bit Mode
LSB first**



**24-bit Mode
LSB first**

For the various modes, the following is a representation of SPROC parallel port transfers for standard external bus widths of 8 bits, 16 bits and 32 bits.

				EADDR 0	EADDR 0
SPROC Port Pins	D23-D16	D15-D8	D7-D0		
32 bit Bus Pins:	D31-D24	D23-D16	D15-D8		
24 bit Mode	MSB	MID	LSB	0	0
16 bit Bus Pins:		D15	D0		
16 bit Mode,LSB First		MSB	MID	1	X
16 bit Mode,LSB First		LSB	0's	0	X
16 bit Mode,MSB First		LSB	0's	1	X
16 bit Mode,MSB First		MSB	MID	0	X
8 bit Bus Pins:					
8 bit Mode,LSB First		*	0's	0	0
8 bit Mode,LSB First			LSB	0	1
8 bit Mode,LSB First			MID	1	0
8 bit Mode,LSB First			MSB	1	1
8 bit Mode,MSB First			MSB	0	0
8 bit Mode,MSB First			MID	0	1
8 bit Mode,MSB First			LSB	1	0
8 bit Mode,MSB First		*	0's	1	1

Key:	MSB - Most Significant Byte
	MID - Middle Byte
	LSB - Least Significant Byte
	X - Don't Care

*Note: This condition wastes a bus transfer cycle and is not necessary.

Parallel Port (Slave Mode)

The SPROC parallel port is a flexible and efficient interface for system designers. The main section of the parallel port is **14 address and 24 data lines, READ, WRITE, and Chip Select.** Interfacing with a SPROC is as simple as interfacing a microprocessor with a static RAM. The SPROC will be a memory-mapped peripheral on the microprocessor's bus.

Timing of data transfers will vary according to the bus width. If a bus width of 24 bits is chosen, the data transfer requires only one write (or read). Refer to the section on **"Memory Models"**. If the selected bus width requires multiple writes to the SPROC, the port circuitry reformats the data into 24-bit words. Likewise when the microprocessor requests data, the parallel port hardware separates the 24-bit word into the required widths.

Watch Dog Timer (Slave Mode)

A watch dog timer (WDT) is provided as a safety feature of the parallel port interface circuitry. To enable the WDT a bit must be set by writing to location 4FFh bit 16. This can be performed by the microprocessor or the SPROC.

With **"8-bit width"** and **"MSB first"** selected, a complete word transfer over the parallel port will take three writes (or reads). If the last byte, the LSB is not written or read within **160 master clock cycles** (indicating an incomplete transfer), the WDT will initiate a soft reset of the parallel port interface circuitry. A similar situation is true for 16-bit mode.

Note: If in "MSB first" mode, the microprocessor reads only the LSB (last byte), the time-out will not occur.

Soft Reset of Parallel Port (Slave Mode)

A soft reset of the parallel port occurs for a variety of reasons. One was discussed above, another is when external circuitry causes both read and write pins to be low. A soft reset clears the flags (RTS, RD, WR, BUSY).

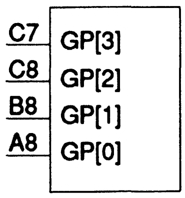
Parallel Port (Master Mode)

In master mode, the bus width can be configured for 8, 16, or 24 bits with 18 address lines (four more than in slave mode). The bus width can dynamically change in the application by software. The parallel port may also interface with devices of varying speeds requiring different timing for each. This is accomplished with the control register located at 4FFh.

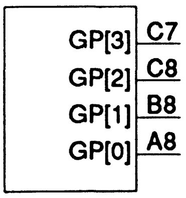
Special Note: In master mode the SPROC powers on in 8 bit mode and can be reconfigured at any time thereafter.

GPIO (Master or Slave)

The GPIO pins are bidirectional lines which may be used in either master or slave modes. The direction and level may be changed at any time by software, and monitored as well. The register for setting the direction and level is located at 4FEh. The register for reading pin levels is 4FCh.



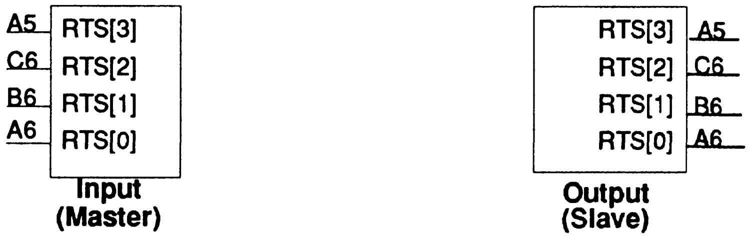
Input



Output

RTS (Master or Slave)

The **RTS** (request to send) pins are **unidirectional** lines. The direction is mode dependent. In master mode the RTS pins are input only, and output for slave mode.



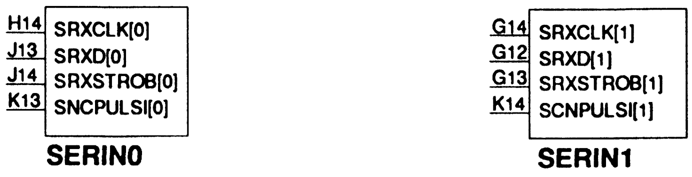
In **master mode** the RTS lines can **only be tested** at location 4FCh. In **slave mode** the RTS lines can be **set or cleared** at 4FEh. This pin can be used for master/slave signalling.

Serial Inputs (Master or Slave)

Two serial input ports support a four wire interface and are extremely versatile, eliminating the need for external hardware. The lines are clock, data, strobe and sync. Each port can work independently of each other. The data flow manager works in conjunction with the input ports to interface with the central memory unit.

Clock lines can be passive or active. Passive indicates the clocks must be supplied by an external device and active means the SPROC supplies the clocks. An internal rate may be selected by writing to 410h or 411h. The value written to these locations will allow an internal counter to divide the rate of the main oscillator. Clock waveform may be continuous or gated (burst mode) and may be inverted.

Data streams may vary in length. The allowable lengths are 8, 12, 16, and 24 bits. The sequence is selectable for **msb** or **lsb** first. The received msb may be inverted to convert offset binary representation to two's complement.



Strobe lines are used to indicate the data is valid. The types of strobes are long, short, inverted, format 2, and format 3.

Sync lines are used by the data flow manager to direct incoming packets to the proper locations within the input FIFO. These lines are used by the data flow manager to align the samples in the FIFO buffers.

The setup for the serial input operation may be written to locations 443h and 44Bh for input ports siport0 and siport1. This is also referred to as the port configuration.

Data Flow Manager - Input (Master or Slave)

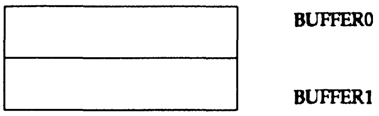
The Data Flow Manager (DFM) is responsible for much of the data movement within the SPROC. This relieves the GSPs from having to perform this function. The DFM must be notified of all data structures, timing and synchronizing requirements of all the GSPs. The DFM may be reset (405h) without the need for the port to be reconfigured. In the case of the serial input ports, the DFM must know the following:

	Locations
FIFO Starting ADDR	442h and 44Ah
Buffer Length (samples per packet)	440h and 448h
Index Length (buffers in FIFO)	441h and 449h

Some examples of FIFO structures:

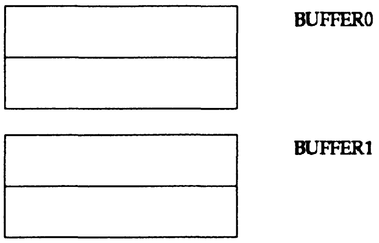
Input Port siport0; 1 sample per packet, 2 buffers in FIFO

Buffer Length	1
Index Length	2
FIFO Start ADDR	assigned by the scheduler
FIFO size	2
Trigger Location	800h (Port 0)



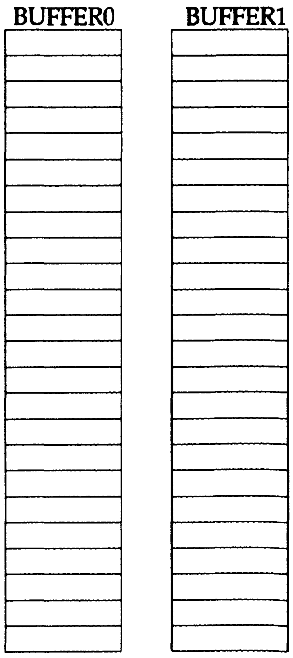
Input Port siport1; Stereo, 2 buffers in FIFO

Buffer Length	2
Index Length	2
FIFO Start ADDR	assigned by the scheduler
FIFO size	4
Trigger Location	801h (Port 1)



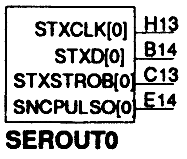
Input Port siport1; Stereo, 2 buffers in FIFO

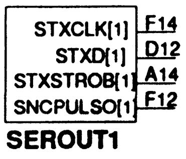
Buffer Length	24
Index Length	2
FIFO Start ADDR	assigned by the scheduler
FIFO size	48
Trigger Location	801h (Port 1)



Serial Out (Master or Slave)

Two serial output ports support a four-wire interface and have the same flexibility as the serial input ports. The configuration registers are located at 453h and 45Bh. In addition, an 8-bit decimation register will allow a resampling at a slower rate. The decimation registers are located at 454h and 45Ch.





Data Flow Manager - Output (Master or Slave)

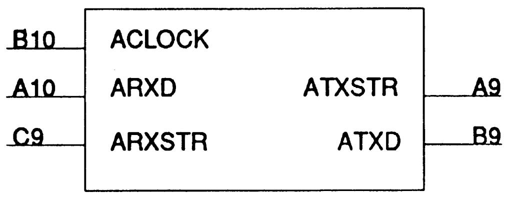
The DFM is responsible for the output data movement. The FIFO structures are much simpler than those of the input. The DFM must know:

	Location(s)
FIFO Length	451h and 459h
FIFO Start ADDR	452h and 45Ah
DFM Reset	405h
DFM Restart	455h and 45Dh

The output DFM is also responsible for monitoring the trigger bus. A register for each output port is available. The locations for these mask registers are 456h and 45Eh.

Access Port Usage (Master or Slave)

The access port is a full duplex, serial interface of synchronous receive and transmit pairs. It has the capability of non-intrusively monitoring and changing the SPROC's code, data and control space. This may also be used in the target system. This port provides added adaptability and control of the SPROC without disturbing the operation. Some precautions must be taken to insure proper operation.



Access Port

In master mode, the SPROC on power-on reset will load its code and data through the parallel port. The access port may be written or read any time after initialization has taken place. In slave mode, the SPROC must be initialized by an external device. This can be accomplished through the host interface (parallel port) or the access port (serial).

Access port receive data are of two types: read commands and write commands. Input transfers are valid only while the receive strobe is enabled. For the SPROC 1400-5, a read command consists of an extended address field only, whereas a write command consists of an extended address field followed by a data field. The extended address field has length 16 bits with msb equal to a "1" for a read command, and "0" for a write command. Three bits following the msb are reserved. The next 12 bits specify the SPROC's internal address for the read or write for only the write command, a final 24 bits (msb first) specifies the data to write to the selected SPROC's address. Note: the three reserved bits must be zero.

In response to a read command only, the SPROC sends a 24-bit data word over the transmit line of the access port while the transmit strobe is enabled. This will be the word read from the location specified in the read command, and is presented msb first.

The access command 09425A5A5Ah will write the value of 5A5A5Ah to location 942h. The access command 8BEFh will cause the access port to transmit the contents of 0BEFh. The access port can read or write data once every 70 master clocks with parallel port usage, and once every 35 clocks with no parallel port usage.

Designing High Performance, Cost Effective Digital Motion Controllers with the SPROC-1400

by Kirk Brisacher

Introduction

Mechanical motion is part of everyone's life. The demanding applications of motion control in consumer, automotive, industrial, instrumentation, and computer peripheral products continue to grow. There exists a need for highly flexible, accurate and cost effective motion control systems development. The growing computational power of digital microprocessor technology and the decreased cost of mechanical shaft and linear encoders has brought a tremendous increase in the application of DC motors servo systems to modern motion control systems.

Digital control loops are by far more adaptive and natural in the environment of today's complex digital based products, in comparison to the analog control loops of yesteryears. Software programmable digital filtering offers a more flexible product design cycle, as well as effective dynamic compensation for complex motion control.

Digital filters eliminate many of the manufacturing problems associated with component tolerance and drift that have plagued analog compensation techniques. They have the added benefit of increasing the level of integration in silicon and reducing the component count on the printed circuit board which decreases the system size and increases reliability. These are important product design issues in an industry like computer peripherals, where a short design cycle and high production yield is of paramount importance.

The objective of this application is to demonstrate a design and development philosophy and technique using the SPROC-1400 and SPROClab that will allow a system designer to quickly design, develop, and test cost effective, compact, reliable, single or tightly coupled multi-axis digital motion controllers. The concepts illustrated in these notes are applicable to a wide variety of DC motors including brush, brushless, stepper, linear and rotary voice coil motors.

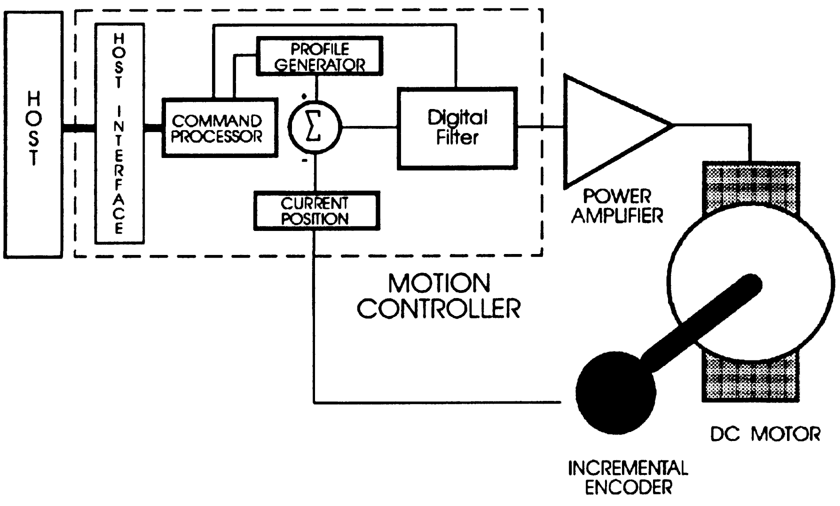


Figure 1. A Digital Motion Control System

Digital Motion Control Systems

A block diagram of a closed loop digital motion control system is shown in Figure 1. Closed loop control systems compare their actual output (position) with a desired command input (position) and use the difference between them to regulate the system's output. The advantages of closed loop control systems over open loop systems are increased accuracy of the system's response to a position or velocity control command and increased bandwidth (speed). The disadvantage of closed loop control systems is that they can become unstable and oscillate. The ease with which one can effortlessly design and test digital filters using the SPROC-1400 and SPROClab makes it possible to quickly and correctly compensate a motion control system and increase the overall system stability.

Hardware Design Overview

Controller Component

The controller is the heart of the motion control system. It closes the position control loop and performs the necessary calculations required to provide dedicated motor control. The controller either outputs a plus/minus analog voltage, commutation drive signals or PWM signal to the amplifier component. Additionally, the controller component does velocity and positional profiling and interfacing to the outside world via anything from a thumbwheel to a host computer. The controller must support high speed real-time processing to avoid feedback loop delays which introduce phase lag, and yet have numerical precision to support accurate positional, velocity profiling and filtering calculations. The controller should allow easy interfacing with analog or digital devices to minimize the number of glue chips required.

Figure 2 illustrates the flexibility of the SPROC-1400 as the heart of the motion controller. There are four subsystems within this generalized motion controller, a processor, a host interface, a feedback sensor input signal and control output signal.

The SPROC-1400 provides the processing power. It calculates the correct velocity or position profile, monitors the feedback sensor and implements one or more digital filters. As is apparent from the diagram very few additional glue components are required to make a master SPROC operational. The processor subsystem consists of the SPROC, a clock generator, a 8x16K PROM and decoder. The clock generator provides the master clock for the SPROC chip and the sampling clocks for A/D converters, D/A converters and internal event triggers. A 8x16K PROM and a simple decoder are all that is required to allow the SPROC to act as a master and self-boot.

The Host Interface Subsystem can be as simple as an 8-bit register file to allow the transferring of commands to, and receiving status back from, the SPROC motion controller. A interrupt request line can be created by using one the the SPROC's GP[0:3] general purpose I/O lines. The host can inform the SPROC that it requires service by latching a request thru the RTS[0:3] inputs. The SPROC can easily communicate with 8/16/24/32 bit data buses.

The Feedback Sensor Signal Subsystem provides position, velocity, or error information for input to the digital filter algorithm. In Figure 2, two types of input are shown, serial and parallel. The input serial port is built into the SPROC and is capable of excepting 8/12/16/24 bits in either MSB or LSB order. This allows easy hook-up to serial A/D converters. The parallel path allows the attachment of quadrature decoder chips, as well as parallel A/D converters.

The Control Output Signal Subsystem drives the amplifier which in turn results in motor motion. In Figure 2, two types of control output signals are shown, serial and parallel. The output serial port is built into the SPROC and is capable of sending 8/12/16/24 bits in either MSB or LSB order. This allows easy hook-up to serial D/A converters. The parallel path allows easy attachment to digital PWM input amplifiers, as well as parallel D/A converters for analog input amplifiers.

The procedure recommended in this application note for designing the other components of closed loop systems that employ the SPROC-1400 is as follows:

- Choose a motor to drive the required load. It is desirable to choose a low armature inertia motor that can develop a high torque to quickly accelerate and decelerate the load. Motors of this type have a high torque constant Kt. It is common practice to choose a motor based on the torque developed by the motor to the moment of inertia of the motor armature, coupling and the load.

- Choose a digital incremental encoder with quadrature outputs to monitor the position of the motor's shaft based on the encoder resolution and accuracy required for the application. Analog encoder can also be easily interfaced to the SPROC-1400 and used for positional and velocity feedback.

- Choose an amplifier to drive the motor. The amplifier must be capable of supplying the power required by the motor for your load conditions. Linear amplifiers are satisfactory for low power, however, switching supplies are recommended for high power applications because of their improved efficiency and size.

- Connect the motor, amplifier, encoder and controller according to the interconnect specifications of each component.

- Using SPROClab, create your required digital filter design using SPROCview and SPROCfil. After you are satisfied with your design, begin testing your filter using the procedure described under "Testing Your Motion Control System".

Optionally, construct a model of the open loop transfer function of the motion system using Laplace Transforms. A Bode plot showing the phase margin and gain margin of the open loop system can then be drawn from the open loop transfer function. Choose the desired closed loop response. The desired phase margin and gain crossover frequency for the digitally filtered system can then be determined from the desired closed loop response.

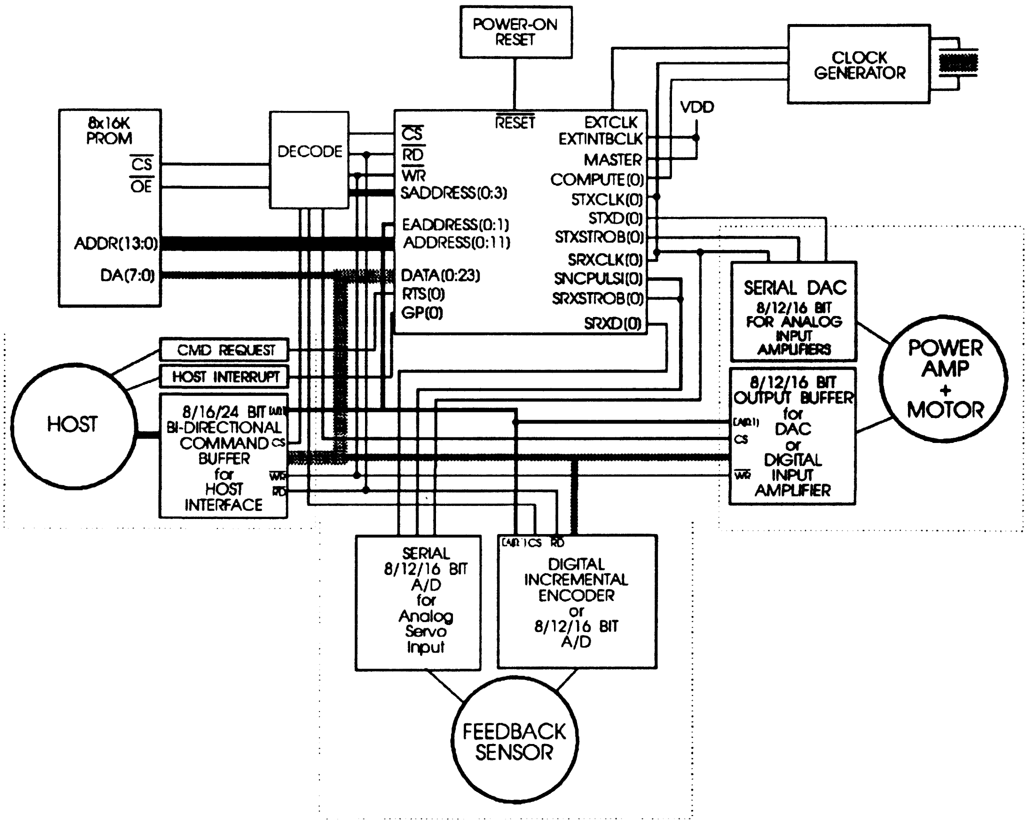


Figure 2. Realization of Motion Controllers

Software Overview

The software required for SPROC-1400 motion controllers is developed with SPROClab. As an example, using the SPROClab development system we initially implemented a standard first order, lead-lag compensation filter. This filter is commonly used in motion control systems. The filter, pictured in Figure 4A, was graphically designed and drawn with SPROCview, a signal flow editor in conjunction with the SPROCcell function library. Notice how few cells were required to complete the design, which took approximately 10 minutes to input into SPROClab. The digital filter was designed and developed without having to write a single line of source code. To generate the required object module for this filter took less than 1 minute to compile and link using SPROCbuild.

Notice that the initial filter contains the pulse cell. The pulse cell was selected as it is an important element for testing the filter's performance by analyzing the motion controller system's response to a step function. Step response provides an excellent vehicle to test the system bandwidth and stability. The encoder cell provides the positional information from the encoder and is so labeled. The minus summing junction provides the error term, i.e., the difference between the projected position and the actual position.

If a particular motion system design appeared to be suffering from a bad resonance, possibly due to the combination and interaction of the amp/motor/load, a sine wave cell can replace the pulse cell to determine the fundamental resonant frequency (Figure 4B.). Notice that a feed forward loop was added to bypass the digital filter and the encoder amplifier gain was set to zero so the system is essentially in an open loop configuration. Once the resonant frequency has been isolated, a notch filter can be designed with SPROCfil and added to the signal flow diagram with SPROCview. The resulting system is shown in Figure 3. Once the resonance has been eliminated the step response testing can continue. Although a system can appear stable by adjusting the filter constants, it is strongly recommended to determine the open loop phase margin and gain to determine the allowable limits of compensation that can be added without causing marginal system instabilities.

As an added example, Figure 5 shows an implementation of a finished PID (Proportional Integral Derivative) filtering system. Notice that the minus summing junction cell is now connected to a profile generator cell, as well as the encoder input cell. The SPROC motion controller, under host control, can now accept and execute positional and velocity profile commands.

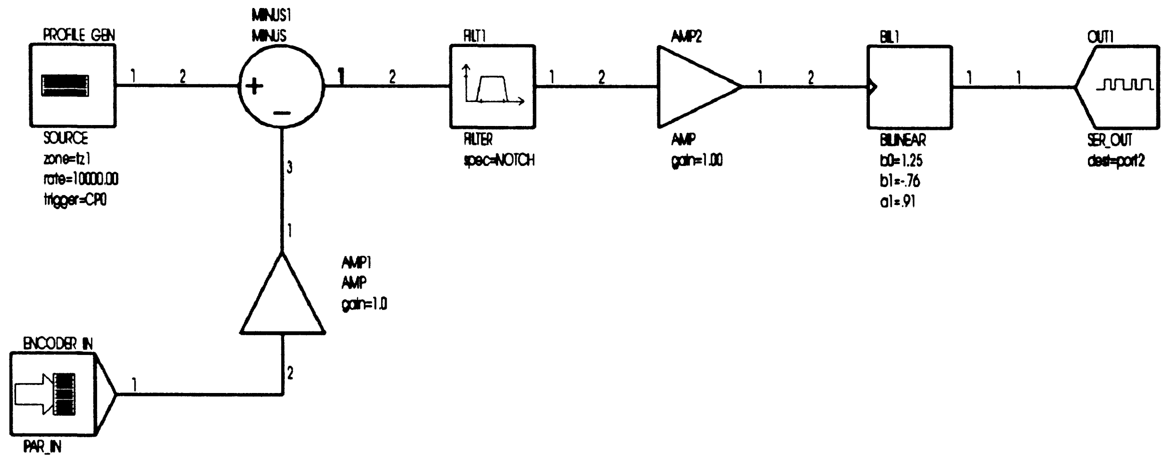


Figure 3. Notch Filter + First Order Lead-Lag Compensation Filter

First Order Lead-Lag Compensation Filter

$$H(Z) = K \cdot \frac{(Z - A)}{(Z - B)}$$

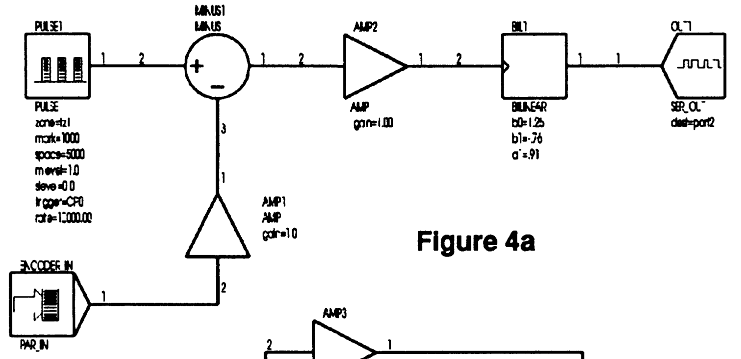


Figure 4a

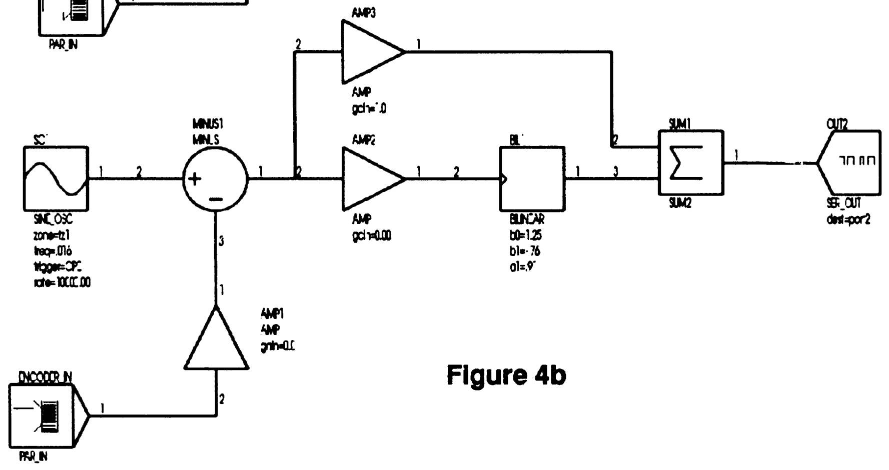


Figure 4b

Proportional Integral Derivative (PID) Compensation Filter

$$U(t) = K_p \cdot e(t) + Kd \cdot de(t)/dt + K_i \cdot \int e(t)dt$$

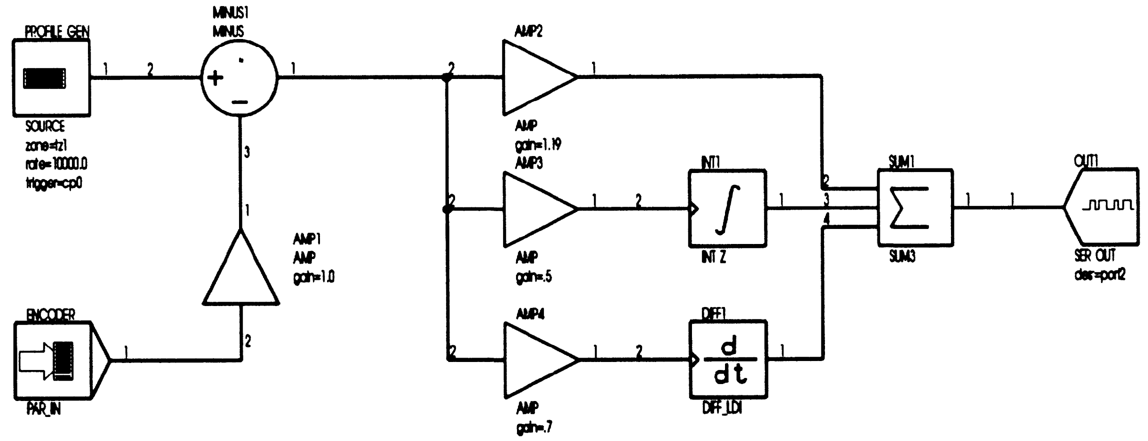


Figure 5. Proportional Integral Derivative (PID) Compensation Filter

Testing your Motion Control System

After building a SPROC-1400 motion controller and connecting it to your DC motor, optical encoder, and amplifier you are now ready to optimize the motor's performance for the given load situation. This involves adjusting the digital filter coefficients to achieve maximum system bandwidth, as well as minimizing system errors and oscillations.

The system performance is optimized by testing its step response. This is where the SPROClab development environment excels. Using the built-in pulse cell, one can inject a step response profile and monitor the actually encoder position with the SPROC probe in real-time. The step size, in encoder counts, should be small enough so that the amplifier is operating in its linear range.

The analog motor's response to the step input is observed by converting the encloder position into an analog signal with the SPROC probing DAC connected to an oscilloscope. One of four types of responses will be observed; underdamped, overdamped, critically damped, and unstable. These responses are shown in Figure 6. The filter coefficients should be adjusted until the critically damped response is observed. Typically, a response with fast risetime, minimum overshoot, and fast setting time is desired.

Once the step response is optimized, the system bandwidth can be approximated from

BW = .35/Tr
BW = System Bandwidth in HZ
Tr = Risetime between 10% and 90% of step response in sec.

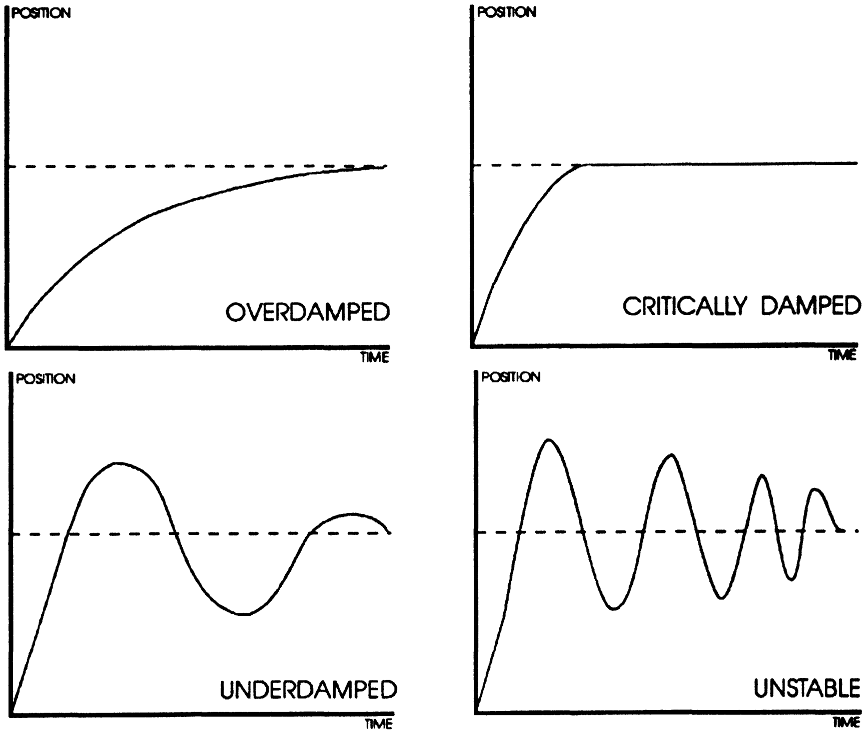

POSITION
OVERDAMPED
TIME
POSITION
CRITICALLY DAMPED
TIME
POSITION
UNDERDAMPED
TIME
POSITION
UNSTABLE
TIME

Figure 6

Implementation of an Adaptive Line Echo Canceller

by Dan Greenwood

Introduction

Telephone circuits are susceptible to a form of electrical echo, also known as line echo, which occurs due to mismatched impedances between various connections in the existing telephone system. A hybrid is a 2-to-4 wire circuit which is used to direct the speech signal flow between two signal paths. While hybrids have been implemented using integrated circuits, most higher end telephones still use two transformers for a hybrid so that there is an isolation between the two telephone systems.

Line echo occurs when a portion of the transmitted signal is reflected back to the transmitting system due to mismatched impedances in the telephone lines between the two end users. Since the impedance of the telephone system between the two callers depends on the distance of the telephone lines, it is impossible to insure a matched impedance for all cases. Historically, a method of insuring that the transmitted signal is not reflected is the use of echo suppressors. The echo suppressor is a voiced activated switch which opens the return path to prevent line echo from occurring. The disadvantage in this approach is that the telephone conversations are limited to being half-duplexed. That is only one person may speak at a single time. Echo suppressors are still commonly used in hands-free telephones and teleconferencing systems.

The advent of adaptive filtering has allowed the implementation of echo cancellers which attempt to remove the echo from the return path while not preventing other signals from passing through. Initially echo cancellers were implemented only at the central switching office of the telephone company. However, due to the advent of commercial digital signal processors, line echo cancellers may also be implemented at a cost effective price on the end subscriber's telephone. This has become a common practice especially in the areas of hands-free telephones, modems and teleconferencing systems.

This application note presents the implementation of an adaptive FIR filter configured as a line echo canceller. A new SPROC cell is developed which can be implemented on 50 MHz SPROC signal processor. A brief background on adaptive filters and echo cancellers is presented along with the system results.

Line Echo Canceller

The area of adaptive filtering has grown widely in recent years. Adaptive algorithms may be used in such applications as speech coding, system identification, active noise control, system modelling, and echo cancellation. The adaptive transversal (FIR) filter is the most commonly used in industry. This is due to its limited complexity, great flexibility, and an ability to insure system stability. Figure 1 shows a typical adaptive FIR filter scheme.

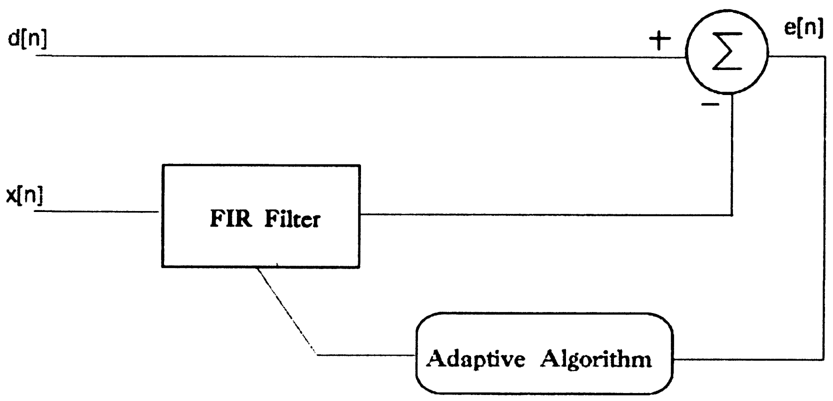


Figure 1. Typical Adaptive Filter Scheme

where:

x[n] = filter input
d[n] = desired or training signal
y[n] = filter output
e[n] = system error

Widrow's Least Mean Square (LMS) algorithm is the most common adaptive algorithm used. The LMS algorithm is implemented in the following manner:

e[n] = d[n] - y[n]
w[n+1] = w[n] + u*e[n]*x[n]

where u is the step-size which controls the speed of convergence of the algorithm. A larger u will increase convergence speed while perhaps causing system instability and excess mean square error after convergence. Smaller u values while requiring longer convergence times, result in a more stable system once convergence has occurred.

The implementation of a line echo canceller (LEC) consists of an adaptive FIR filter which attempts to adaptively model the telephone hybrid's response. Once the filter has converged, the line echo may be predicted from the known transmitted signal and then subtracted from the return signal thereby cancelling the echo in the return path. The adaptive line echo canceller set-up is shown in Figure 2. The system has two inputs and one output: x[n], d[n] and e[n] respectively. The input x[n] is the near-end speakers transmitted signal. The return signal from the hybrid is d[n] and is the LMS training signal. Finally the error signal e[n] is our return signal after echo cancellation.

A line echo canceller SPROC cell can be divided into two functional blocks: the FIR filtering and the LMS coefficient update. In order to partition this task more evenly among the available GSPs of the SPROC, it is advantageous to subdivide the LEC implementation into more than one sub-cell. The LECBLOCK cell consists of two separate cells: AFIRX and LMSX which share a common coefficient/data vector. The assembly files for these three cells are included for reference in listings 1 through 3. The LECBLOCK cell has two user specified parameters: mu and length. This allows the line echo canceller to be modified easily for different applications.

This application provides an example of one method of allowing two cells to share a common memory vector. The line echo canceller uses a hierarchical text approach to implement a single block in the signal flow schematic which contains more than one internal cell. The advantage in this method is that a complicated task may be subdivided into several tasks which may be partitioned onto more than one GSP of the SPROC. This is important since the SPROC compiler will not partition a single cell over multiple GSPs. Therefore by dividing the line echo canceller into separate functional blocks we can utilize more of the processing power of the SPROC.

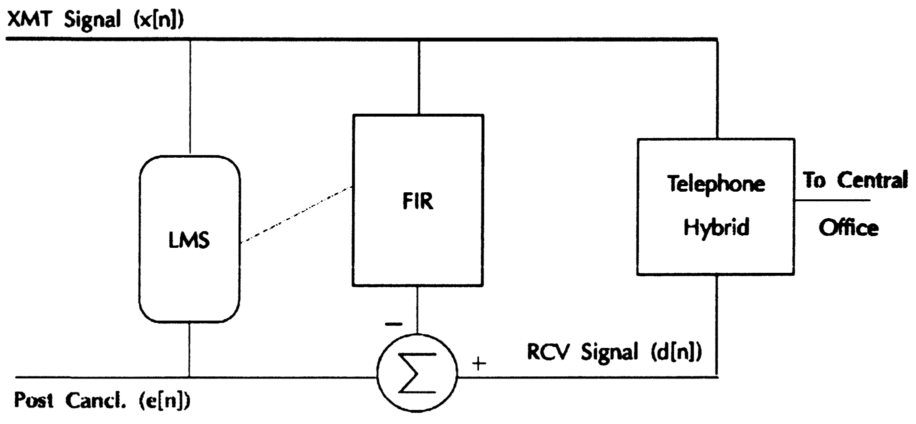


Figure 2. Adaptive FIR Configured as a Line Echo Canceller

The line echo canceller was implemented in two manners for testing. Initial testing was performed by using a NOISE generator cell as the transmitted signal. The white noise signal was fed though a bandpass IIR filter and amplifier to model a telephone hybrid response. The bandpass filter was designed using SPROCfil and has a passband between 300 and 3300Hz to be consistent with the telephone system. The amplifier gain was set to implement a 6 dB loss, i.e. gain =0.5, which is a common estimate of the return loss of a line echo signal. Figure 3 shows the signal flow diagram for this test implementation.

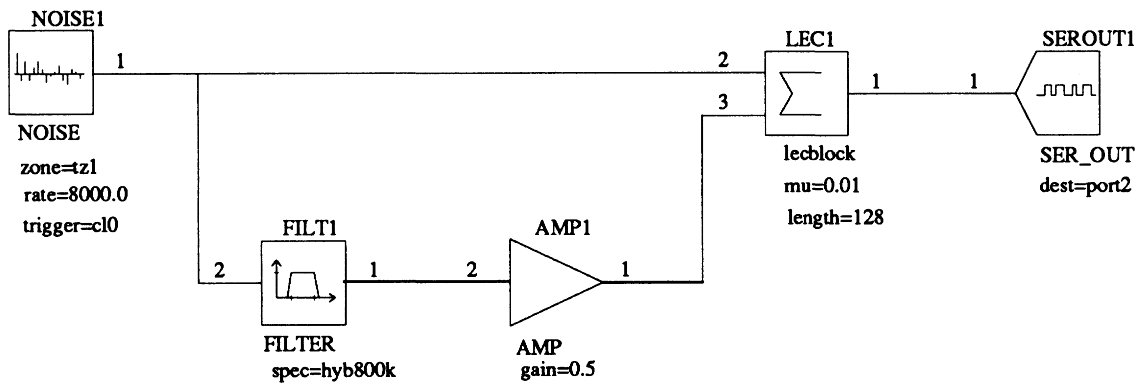

NOISE1
NOISE
zone=tz1
rate=8000.0
trigger=cl0
1
2
FILT1
FILTER
spec=hyb800k
1
2
AMP1
AMP
gain=0.5
1
2
3
LEC1
lecblock
mu=0.01
length=128
1
1
SEROUT1
SER_OUT
dest=port2

Figure 3

Figure 4 shows the signal flow diagram for an actual line echo canceller system which may be placed in a telephone system. The system utilizes both serial input channels and one serial output channel of the SPROC signal processor. The system may be placed in an existing telephone system by re-routing the signal from the near-end speaker's transmitter and also the received signal from the telephone hybrid. The serial output port is directed to the near-end speaker's receiver/speaker.

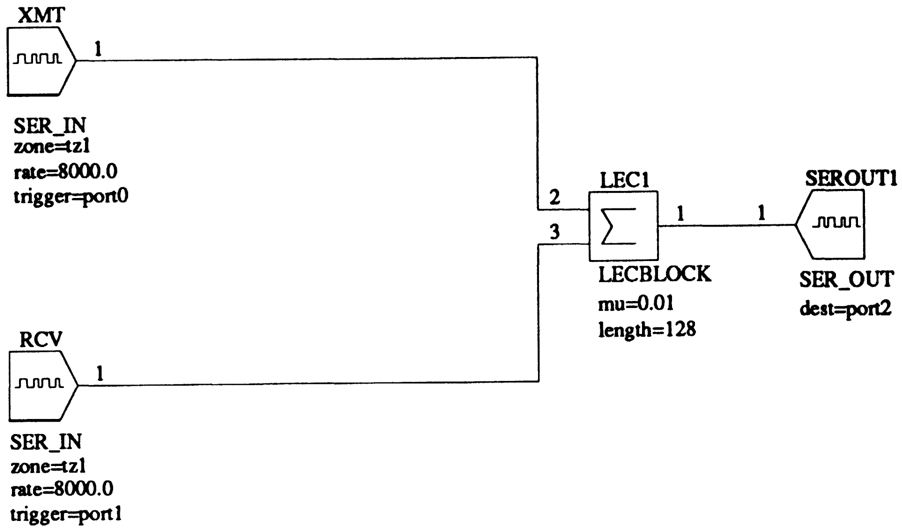


Figure 4

Summary

An adaptive line echo canceller cell was developed for the SPROC signal processor. The implementation divided the adaptive filter into stages which could better utilize the processing power of the SPROC. The line echo canceller was tested with an order of 128. The application tested using the 20 MHz SPROCboard at a reduced sampling rate. The overall cancellation using the white noise generator was over 25 dB.

The first test using the noise source as the transmitted signal can be implemented on 1.1 GSPs on the 50 MHz SPROC. The number of data locations used in the implementation was 351. The application required 169 program locations. Test two using actual serial inputs requires 1.0 GSP on the 50 MHz part. The data and program space required for this application were 282 and 123 respectively. The remaining GSP power may be used for other tasks including the implementation of control cells which may be used to control the line echo canceller adaptation.

This application demonstrates a method defining common data segments between separate cells. This provides a method of sharing large amounts of data between cells without placing separate wires between the cells. This provides a less complicated signal flow diagram which is also more flexible. The SPROClab development system allows the user to quickly determine the proper values for parameters such as the LMS step-size.

Listing 1.

```
/*******************************************************************
                              afirx.sdl                          *
* *****************************************************************/

asmblock afirx { %subr=default} ( xn ; yn )

//
duration 4*(%length-1)+10;      // lines of code = 11
//
begin
         lda xn                 // Areg = x[n]
         ldb #lec_vect          // Breg points to start of lec_vect
         sta [B+%length]        // store new x[n] in lec_vect

         ldl #%length-1         // initialize loop counter register
         ldd #1                 // initialize Dreg for decrement step
AFIR1:   ldx [B+L+%length]      // Xreg = x[n-L] oldest x value in vector
         mpy [B+L]              // Mreg = x[n-L]*w[n-L]
         djne FLOOP             // decrement Lreg
FLOOP:   ldx [B+L+%length]      // Xreg = x[n-L] & MPY complete
         mac [B+L]              // Mreg += x[n-L]*w[n-L]
         stx [B+L+%length+1]
         djne FLOOP             // check Lreg and loop if needed
//
         lda MH                 // Areg = MH = y[n] sta yn // save y[n]
// end of asmblock afir
         end
```

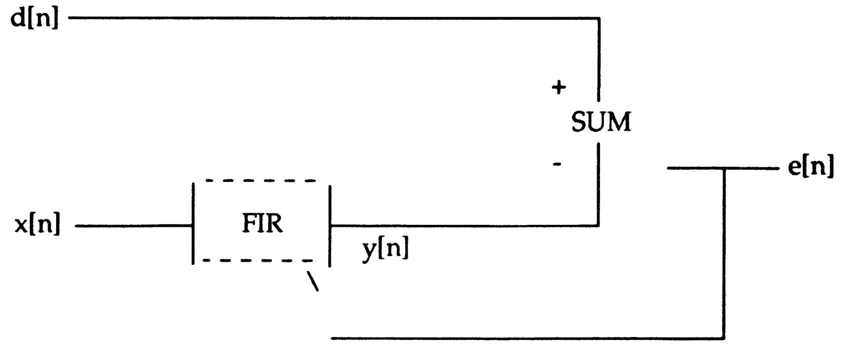
Listing 2

```
/*******************************************************************
*                           lmsx.sdl                               *
* *****************************************************************/

asmblock lmsx { %subr=default } ( dn, yn ;en )
//
// End of Filtering, Now calculate error e[n] = d[n] - y[n] (Mreg)
//
verify (%mu>= 0.0 && %mu <= 0.25), 'Specify step-size, 0.0 <= mu, 0.25';
//
duration 5*(%length-1)+14;      // lines of code = 17
```

```
//
begin
// End of Filtering, Now calculate error e[n] = d[n] - y[n] (Mreg)
//
         LDA dn                    // Areg = d[n] = training signal
         SUB yn                    // Areg = d[n] - y[n]
         STA en                    // store e[n] for output
// Calculate mu * e[n]
         ldx A                     // Xreg = e[n]
         mpy mu                    // Mreg = mu * e[n]
//
         LDB #lec_vect             // Breg points to start of lec_vect
         LDD #1                    // initialize Dreg for decrement step
//
// Perform Widrow's LMS update: w[n+1] = w[n] + mu * e[n] * x[n]
//
UPDATE:  ldl #%length-1            // re-initialize loop counter register
         ldx MH                    // Xreg = mu * e[n]
         mpy [B+L+%length+1]       // Xreg = x[n-L] * mu * e[n]
         lda [B+L]                 // Areg = w[n] current coef.
         djne ULOOP                // decrement Lreg
ULOOP:   add MH                    // Areg = w[n] + mu * e[n] * x[n-L]
         sta [B+L+1]               // store updated coef, w[n+1]
         mpy [B+L+%length+1]       // Mreg = mu * e[n] * x[n-L]
         lda [B+L]                 // Areg = w[n], current coef
         djne ULOOP
//
         add MH                    // Areg = w[n] + mu * e[n] * x[n]
         sta [B]                   // store last coefficient
// end of asmblock lms
         end
```

Listing 3

```
/******************************************************************
*                          lecblock.sdl                              *
* *****************************************************************

Function:

          Line Echo Canceller using user specified order Adaptive FIR
          Filter using Widrow's Least Man Square Algorithm (LMS).

Arguments:

Parameters:

          fix      xn
          fix      dn
          fixed  mu       0.0 <= mu <= 0.25   // LMS step- size
          int      length                     // adaptive filter length

Algorithm:
```



```
     e[n] = d[n] - y[n]

     Widrow's LMS Algorithm Coefficient Update

     w[n+1] = w[n] + mu * e[n] * x[n]

*/
```

```
/*********************/
/       * inline code            */
/*********************/

block lecblock { %subr=default, %mu, %length } ( xn,dn ; en )
// verify ( %mu >= 0.0 && %mu <= 0.25 ), 'Specify step-size, 0.0 <= mu ,
0.25';
//
variable fixed yn;
variable fixed mu = %mu;
variable fixed lec_vect[2*%length+2]=0.0;
//
begin
afirx afirxl
    {
    }
    (
    xn,
    yn
    );
//
lmsx lmsxl
    {
    }
    (
    dn,
    yn,
    en
    );
    end
```

FRACTIONAL SAMPLE RATE CONVERSION APPLICATION NOTE

by Stephen J. Sheslow

Abstract:

Digitally sampled analog signals at a known sample rate can be converted to a new sample rate using a fractional sample rate conversion method. This method uses the time delay properties of the Fourier Series to reconstruct the final sample rate desired using a Finite Impulse Response, FIR, reconstruction filter with filter coefficients related to the current time difference of the original sample and the desired final sample.

Introduction

Analog band limited signals used by digital systems are all sampled using an analog to digital conversion process. The original analog signal once converted to a sampled digital representation is now described by the set of digital samples and is only defined at these discrete time samples. The properties of a sampled analog signal represent a combination of the properties of the original analog signal and the sampling function. The sample rate is related to the spectral properties of the original continuous time signal transformed into the discrete time spectrum by the sample rate. The resulting discrete time signal can be used to reconstruct the original continuous time signal.

In many cases it is desired that the original sample rate be changed to a different final sample rate. The sample rate conversion between the two sample rates can be accomplished by using the original samples of the discrete time signal in the discrete time spectrum. One technique for discrete time sample rate conversion is fractional sample rate conversion. This technique uses a Finite Impulse Response, FIR, filter for reconstruction of a single output sample based on a finite set of the input samples. The coefficients of the FIR filter are modified for each desired output sample based on the time delay between the two samples.

The SPROC signal processor will be used for this sample rate converter. The SPROC signal processor's hardware architecture and the SPROClab signal flow diagram development tools are ideal for this sample rate conversion application.

The FIR filter used for reconstruction of the output sample requires a new set of coefficients for each output sample. The SPROC data flow manager allows real time access to the fast internal memory of the SPROC signal processor without interruption of current algorithm. The four parallel signal processors of the SPROC signal processor allow an FIR filter to be accomplished in decimated parallel blocks for shorter propagation times of output samples.

Sample Theory Basics

The conversion of analog signals into a series of discrete time samples requires some knowledge of the original analog signal's characteristics. The original signal should be band limited so it can be represented by a Fourier Series expansion of the original signal. The constraint on the bandwidth of the original analog signal to be sampled will allow the sample rate of the signal to be a finite number. A sampled signal has characteristics in the frequency domain that are related to the original bandwidth of the signal combined with the characteristics of the actual sampling function which will be discussed.

A technique to sample a continuous time signal into a discrete time signal is to multiply the continuous time signal by a periodic series of the delta function. The delta function, $\partial(x)$, is defined as one for the input zero and zero for all other inputs. The mathematical convolution of a peridodic delta function and a continuous time signal results in an infinite series of discrete time samples based on the frequency of the periodic delta function.

Sample Rate Conversion

A sampled continuous time signal at some original sample rate can be converted to a new sample rate using digital signal processing of the original discrete time signal. The original discrete time signal samples are used to generate new samples at the desired sample rate. One technique for the conversion of sample rates is to reconstruct the new periodic sample sequence as a shifted in time representation of the original sequence. The properties of the Fourier Series related to time shift are the basis of this method.

The time shift properties of the Fourier Series transform state that a sequence of time domain samples, shifted in time, is equal to the Fourier Series transform of the original unshifted signal samples multiplied by the exponential function evaluated at the time shift value. This property can be applied to sample rate conversion using a FIR filter for the exponential function multiplication.

The Fourier Series time shift property can be applied to a method for sample rate conversion between two arbitrary sample rates. The sample rate desired can be represented at each point as a time domain shifted version of the original sample value. The actual time shift between the two samples can then be used to determine the value of the exponential function needed to calculate the new sample rate. This technique is referred to as Fractional Sample Rate conversion.

Finite Impulse Filter Reconstruction

The original sampled signal data can determine the desired output samples using a finite amount of input samples. This finite set of original samples can be used to reconstruct the desired samples based on the addition of weighted original samples representing the current relative time delay of the desired samples. The desired sample at a known time is related to the original sample by a known time difference. This will determine the filter coefficient values, or weights, to be used for the current desired output of the conversion. The number of samples used for the Finite Impulse Response reconstruction will affect the filter length and therefore the hardware requirements.

The Finite Impulse Response, FIR, filter will introduce some error as related to the infinite filter theoretical response. To decrease the effect of using only a finite number of original samples for the desired sample calculation the reconstruction filter coefficients are multiplied by a window. The window will taper the values of the filter coefficients to minimize the effect of using only a finite number of filter taps, or coefficients, for the reconstruction. This window method is frequently used when a finite sample length results in a rectangular window.

A rectangular window of samples is a finite set of samples. This window will be zero on either side of its bounds and the sample value inside its bounds. A rectangular window results in a instantaneous change from zero to full value of the windowed input samples. A series of alternative window functions have been explored which allow the input samples to gradually increase to full value at the center and then decrease to zero at the lower and upper bounds of the sample window. By removing the step function of the rectangular window better reconstruction of the new desired sample will result.

Hardware Requirements

The algorithm discussed requires the FIR reconstruction filter to be propagated for each desired sample. Each new sample that is calculated is related to the original sampled signal and the current time delay for the desired sample. This time delay will determine the value of the filter coefficients for the FIR reconstruction filter. In order for the algorithm to be performed in real time, the FIR reconstruction filter must not take more than one sample period of the desired output frequency. The filter coefficients must be altered during the current calculation to prepare for the next output sample. This requirement of the total time duration of the filter coefficient update and output calculation of the filter allows the system to continue at the desired output sample rate.

SPROC Implementation

The STAR Semiconductor SPROC signal processor is ideally suited for Finite Impulse Response, FIR, reconstruction filter, Fractional Sample Rate, (FSR) conversion. The architecture of the programmable signal processor allows for the real time uninterrupted access to internal program and data memory during execution. This feature allows for the different filter coefficients to be written for the FIR reconstruction filter as the current filter coefficients are used by the filter. The data flow manager of the SPROC hardware allows this real time access to memory and permits the processor to continue execution of the current reconstruction filter as the new filter coefficients are supplied.

In addition to the ability of the SPROC signal processor to allow for real time access to program and data memory internal to the device, the SPROC signal processor has four parallel processing general purpose signal processors, GSPs, per single device. Each GSP executes its own process as scheduled by the software. The actual SPROC processor code is generated based on a signal flow graph representation of a particular algorithm. All the real time considerations and data variable requirements of the signal flow graph for a particular algorithm are handled by the SPROClab software development tools.

The general purpose signal processors, GSPs, of the SPROC signal processor have an architecture specialized for the mathematical processing associated with most signal processing algorithms. Each GSP has a 24-bit data path and a 56-bit multiply accumulator to minimize the effect of fixed point multiplication. Each GSP has a flexible address generation block for use in algorithms that use data arrays such as filters. The address generation is structured with a loop register, a base register, a frame register, and a decrement register.

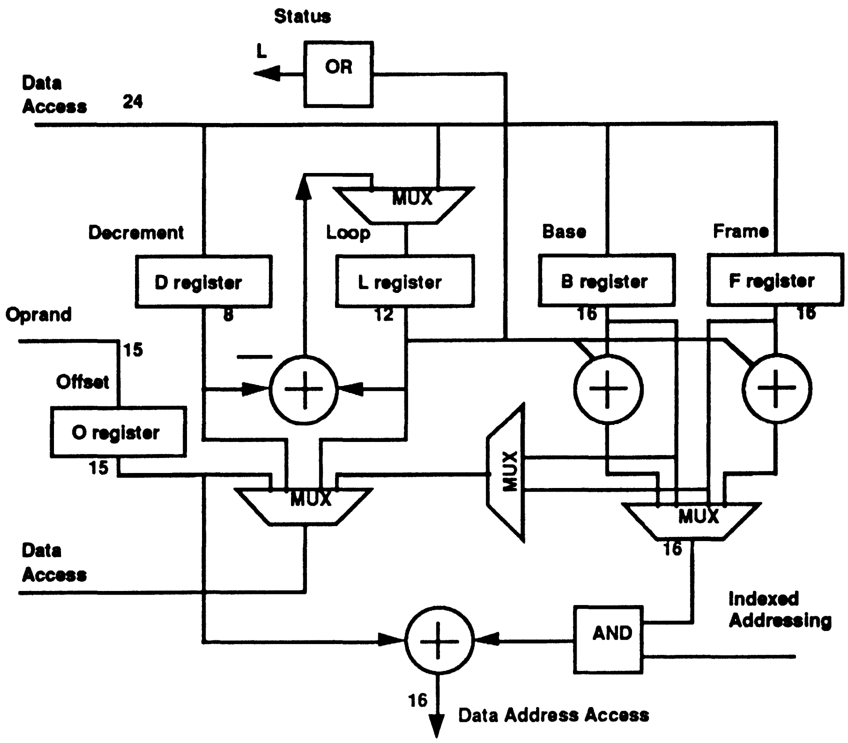


Figure 1

SPROC Firmware Structure

The SPROC signal processor firmware is developed based on the signal flow diagram of the fractional sample rate converter algorithm. The algorithm will accumulate the samples required for the output sample of the reconstruction FIR filter. When the output sample is required the SPROC signal processor will calculate the output of the filter and be loaded with the next set of filter coefficients. The coefficients are related to the current time delay of the desired output related to the original input. This time delay results in an index for the coefficients for the FIR reconstruction filter. The coefficient index is used to allow the next set of coefficients to be loaded into the internal memory of the SPROC processor. The firmware signal flow diagram is shown below:

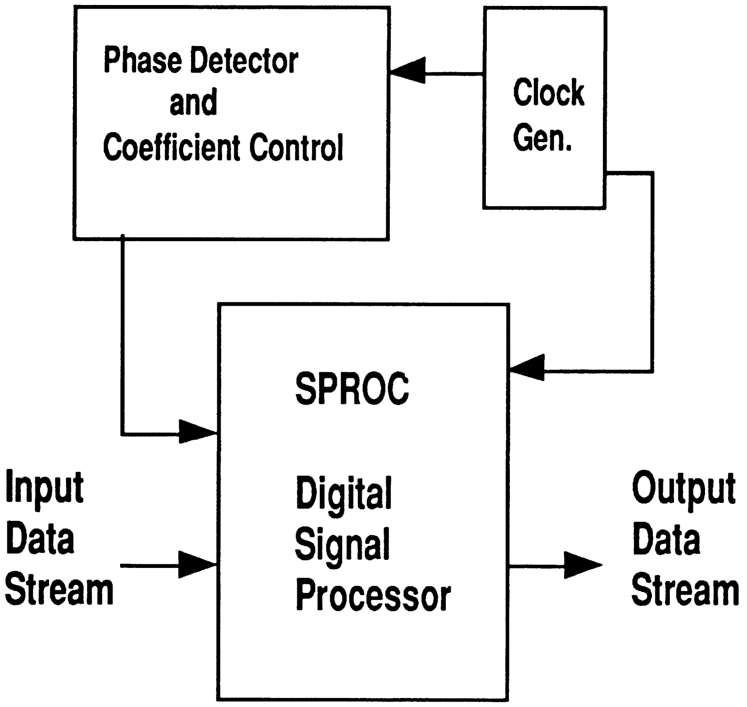

Phase Detector and Coefficient Control
Clock Gen.
SPROC Digital Signal Processor
Input Data Stream
Output Data Stream

Figure 2

The firmware for the sample rate conversion shows the input data entering a serial input block of the SPROCcells function library. The serial input block is all that is required for the input of data to the SPROC signal processor. The input of the signal flow diagram from the serial input block has a specified sample rate as a parameter for the SPROC software to create the application. This sample rate will generate a trigger for the signal flow to be calculated. The data collection routine for the sample rate converter as defined by the serial input block of the signal flow diagram will provide the data for the actual FIR filter of algorithm.

The firmware for the SPROC signal processor uses a block representation of an FIR filter for code generation of that filter. The FIR filter is therefore placed between the input and the output of the algorithm. The FIR filter block requires that a coefficient file be specified. The application requires the coefficient set to be changed for each output sample. The actual hardware address of the coefficient set is used by the microprocessor of the application to provide the coeffiecient set in the internal memory of the SPROC signal processor. This memory access permits uninterrupted execution of the FIR filter during coefficient updates.

The output of the filter internal to the SPROC signal processor is connected by the firmware to the serial output block. This block of the SPROC firmware is all that is required for the output of the filter algorithm to be output at the serial port of the SPROC signal processor. The serial output block requires only the port designation as a parameter for the block. This will then map the output of the signal flow diagram, representing the firmware, to the hardware of the serial port for output.

SPROC Processor Interface

The SPROC signal processor permits access to the internal memory of the device by use of the Data Flow Manager, DFM. The DFM controls the access to the data and address buses of the SPROC signal processor. The SPROC signal processor in a slave mode configuration appears to a microcontroller as a section of static ram. The SPROC signal processor therefore interfaces to a microcontroller easily and needs no interrupt handler or context save routines like other processors. The DFM permits access to the internal memory by both the serial input ports, both the serial output ports, the parallel port, the serial access port, and the on chip software directed probe. The probe on the SPROC signal processor can be assigned to any signal of the signal flow diagram to allow investigation while the SPROC signal processor is executing algorithms. Probing is performed without interrupting the GSPs.

The sample rate conversion algorithm will be accomplished using a micro controller to write the new filter coefficients into the filter space used by the SPROC signal processor. The microcontroller will calculate which set of coefficients represent the current time delay between the desired sample and the original sample rate set of samples. This calculation of time difference is cyclical for two related sample rates and therefore the set of coefficients is finite. The order of the set of coefficients is also cyclical. The microcontroller interface is concurrent to the calculation of the current output sample calculation. The microcontroller will write the coefficients directly into [illegible] SPROC signal processors internal data memory.

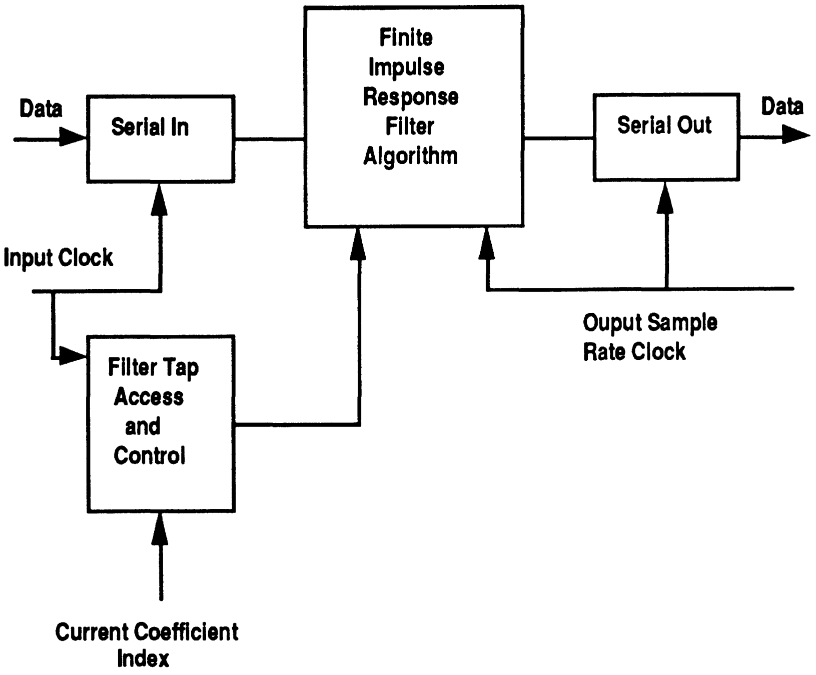

Data
Serial In
Finite Impulse Response Filter Algorithm
Serial Out
Data
Input Clock
Ouput Sample Rate Clock
Filter Tap Access and Control
Current Coefficient Index

Figure 3

Conclusion

The SPROC signal processor's hardware architecture has the data flow manager for independent access to the internal memory of the processor. The ability to access this internal memory during program execution allows for adaptive FIR filters to be updated while the filter is in operation. The architecture of the SPROC signal processor is therefore optimum for adaptive filters and filters that require larger coefficient set sample spaces. One such large filter coefficient set application is the fractional sample rate conversion reconstruction filter. This reconstruction FIR filter is propagated for each desired output sample and the filter coefficients are updated for the next sample concurrently via the parallel port of the SPROC signal processor.

The parallel architecture of the four general purpose signal processors, GSPs, allow for more processing power resulting in greater signal throughput. These mathematical accumulators of the SPROC signal processor permit large FIR filters to be realized in the parallelization of smaller FIR sections executing simultaneously to achieve the desired filter length. This parallelization of the GSPs allows filters that can run concurrently. It also would permit partitioning of large filters over many GSPs if needed for greater real time performance.

The ability to develop application specific firmware with the use of signal flow diagrams allows the designer to complete the application firmware quickly. The designer can concentrate on the signal flow of the algorithm and not the actual code required. The SPROClab development tools handle the actual SPROC signal processor code generation from the designer's signal flow diagram. This signal flow diagram once processed into a loadable file for the SPROC signal processor can be probed in real time during its execution based on the Data Flow Manager, DFM, of the SPROC hardware structure and the SPROCdrive software of the SPROClab development system.

Application of Linear Phase Filterbank to Frequency Shaping Digital Hearing Aids

by Brian M. Finn and Sen M. Kuo, Northern Illinois University

Introduction

As human life expectancy increases, environmental noise increases, and society's acceptance and desire for high sound pressure music increases, so does the risk for noise-induced sensorineural hearing loss. Hearing loss often associated with the geriatric population is now a problem for those of all ages.

At this time, hearing improvement offered to those with sensorineural losses often is in the form of an amplification system, with little or no adjustable compensation for specific spectral enhancements. As sensorineural hearing loss is often sloping, or perhaps notched, it is of practical significance to design a hearing aid with multiple filterbanks to provide useable gain at the frequencies where the ear is lacking sensitivity. As no two hearing impaired listeners have similar thresholds of hearing vs frequency and traditional frequency shaping has drawbacks of those associated with all analog filters, the motivation for a digital based multiple filterbank hearing aid is found.

Several ideal properties are thus approached in the initial design phases of such a system. As a filterbank is to be the spectral inverse of the ear's residual hearing sensitivity, sufficient bands must be implemented to allow custom magnitude modification. The filters are also to be linear phase to not produce any audible distortion; likewise, the group delay is to be minimized (<5ms) to not produce temporal distortion between the listener and his environment. Because of the linear phase requirement, symmetrical FIR filters must be implemented; however, a multiple series of FIR filters with sharp cutoffs necessary to prevent spectral leakage between bands requires large data buffers and significant processing power. The Interpolated FIR filter, with perfect symmetry, complementary outputs, and a minimum number of multiplications per sample is an elegant solution to the problems associated with the ideal properties of a digital filterbank hearing aid. INTERPOL, the interpolated FIR filter, has been implemented in the SPROC description language, as shown in Listing 1.

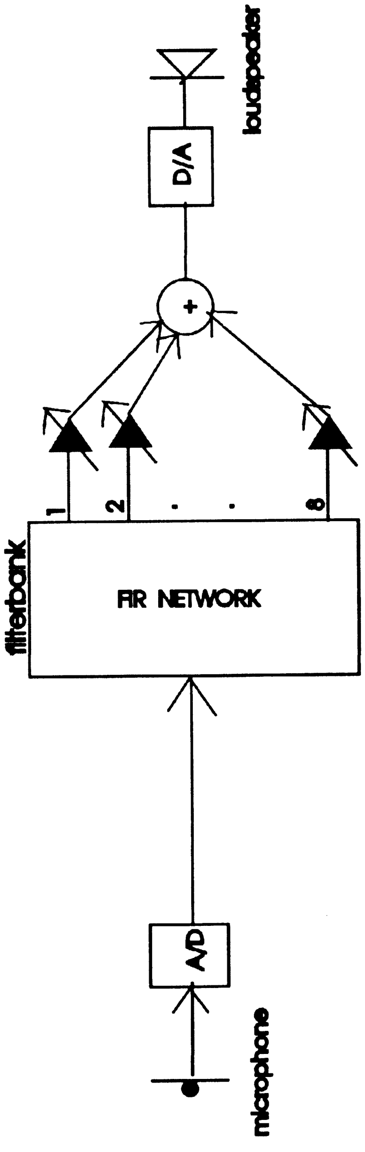
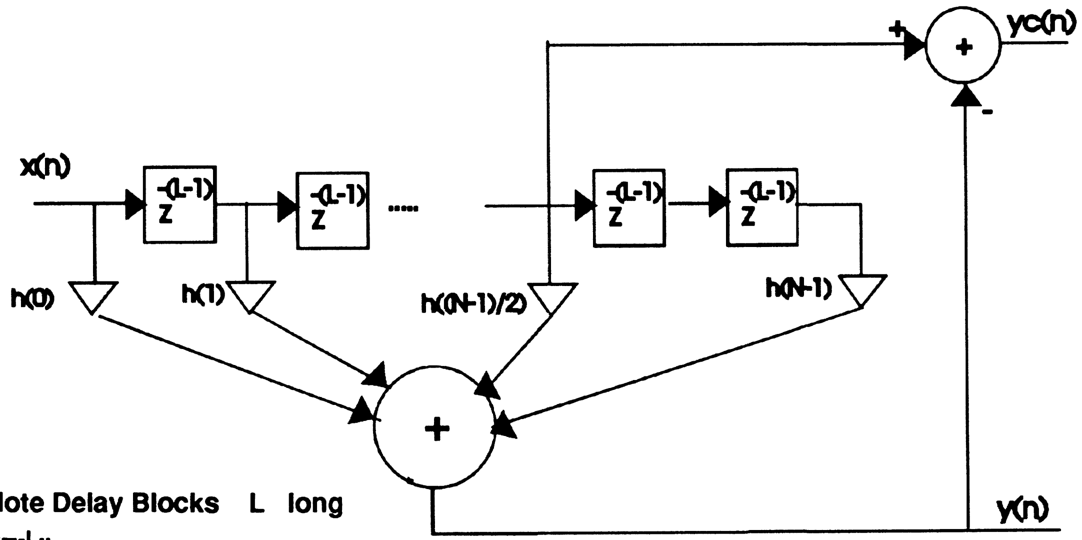


Figure 1

The first step in designing the filterbank is to determine the number of bands (L) needed for a specified frequency resolution, where each band is fn/L wide and fn denotes the Nyquist frequency. In this application eight bands were chosen as a balance between spectral resolution and computational resources. By interpolating a lowpass halfband filter, -50dB stopband attenuation, with an L factor (%zeroes in INTERPOL cell) of eight, five pass bands and four complementary pass bands are generated. This cell is analogous to padding the original seed impulse response with L-1 zeros between successive seed coefficients. The lowpass filter seed coefficients were generated in SPROCfil by use of the Parks-McClellan algorithm. In generating this "soft rolloff" filter only nine coefficients are required, as N the filter order (%coefs in INTERPOL cell) is always to be odd for symmetry. As stated one of the factors for choosing an interpolating FIR filter is that the complementary output yc(n) is computed by subtracting the transversal output y(n) from the input x((N-1)/2) in the input data buffer. This results in a highly efficient cell algorithm whose first stage requires coefficient data storage of N, data storage of N + (N-1)*(L-1) (%length in INTERPOL cell), and N multiplications per cycle. Figure 2 shows the FIR interpolating block diagram with both passband and complementary passband outputs.



Note Delay Blocks L long
"Z^{-L}"

Figure 2

From the first filter step, where pass bands are separated on an even/odd basis, sub-filter banks proceed to decimate the input signal spectrally so that individual gain may be applied to each frequency band in any arbitrary manner. Using the initial seed coefficient file, an interpolation factor of L= 4 is used to generate the next level of spectral decimation on the initial output. The complementary output is passed through a different seed filter with an interpolation factor of L=2, that splits the inner two even spectral bands from the outer two spectral bands. Because this seed filter was a lowpass halfband with a much narrower transition band, 13 coefficients are required. From here further interpolating sub-filter banks with either seed file decompose the signal into the final band-passes, with each successive filter requiring a lower order as the signal is being progressively decimated. A single output LP filter is required to remove the alias (9th) band from INTERPOL 3's output. Table 1 gives the seed filter coefficients, while Figure 3 shows the signal flow and filter magnitude transfer functions.

Table 1

Coefficient	Seed 1	Seed 2
h(0)	-0.01550278	-0.01074323
h(1)	-0.04962827	0.0250155
h(2)	0.03100575	-0.01608604
h(3)	0.289573	-0.07562522
h(4)	0.5	0.2192462
h(5)	0.289573	0.3014174
h(6)	0.03100575	0.5
h(7)	-0.04962827	0.3014174
h(8)	-0.01550278	0.2192462
h(9)		-0.07562522
h(10)		-0.01608604
h(11)		0.0250155
h(12)		-0.01074323

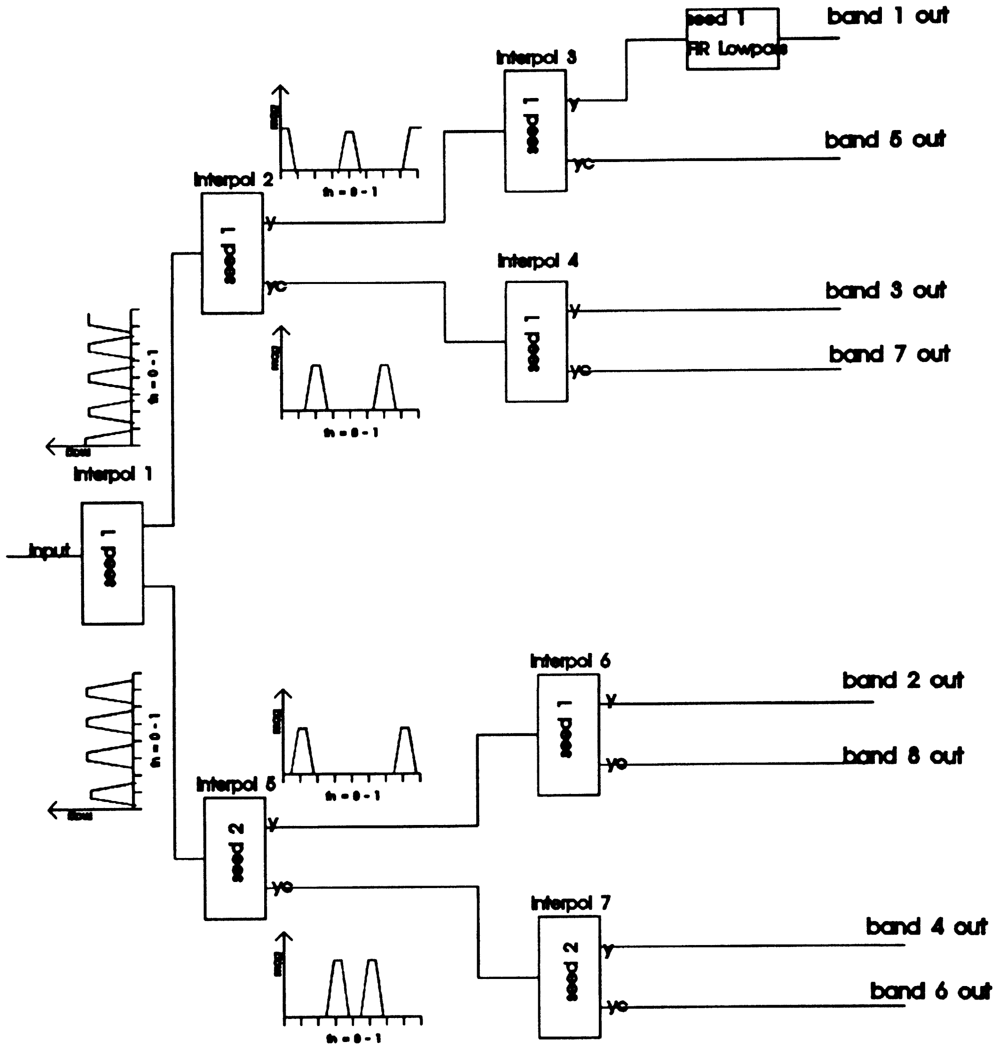

Input
seed 1
Interpol 1
Interpol 2
Interpol 3
Interpol 4
Interpol 5
Interpol 6
Interpol 7
seed 2
seed 1
FIR Lowpass
y
yc
band 1 out
band 5 out
band 3 out
band 7 out
band 2 out
band 8 out
band 4 out
band 6 out
mag
fn = 0 - 1

Figure 3

Following the signal paths through Figure 3 one sees that each path has a slightly different number of delays imposed by the transversal filter networks. Ideally it would be possible to add delays to the paths so that there was a common delay to each band, equal to the delay of the longest path. In this design, sampling at 11 KHz, only the path producing filterband 1 (6.2 ms) exceeds the ideal delay limit of 5 ms. Because of the minimal sampling rate used, delay issues are not addressed in this real-time implementation.

Once the filterbank has been realized, the SCALER cell is used to adjust the band level in +/-6 dB increments. The just noticeable difference hearing level is approximately 3 dB, so bit shifting in 6 dB increments is reasonable to provide magnitude shaping to the hearing impaired listener. It is noted that one of the most desirable qualities of the digital filterbank hearing aid is that the response can be adjusted in a real-time prescription fitting method. By writing the shift level into the scaler, a patient or audiologist has a method of adjustment to provide for the optimal frequency shaping curve.

In real time evaluation of the filterbank hearing aid, which is schematically represented in Figure 4, careful attention must be given to not saturate the D/A converters by imposing such a high gain in one or more bands that the output level exceeds their threshold of +/-3.0V. It was also found that the non-essential bands could be scaled below unity to prove for maximal shaping of the most essential bands.

Using the 20Mhz SPROC chip, a sampling frequency of 11,000 Hz was implemented with the external A/D clock input. At this rate and chip speed 3.9/4.0 GSP's were utilized, and 345 data and 594 code locations were needed.

This same application with the same sampling rate of 11,000 Hz when compiled for a 50 MHz SPROC requires 1.5/4.0 GSP's. Assuming the 50Mhz SPROC chip were used at 16,000 Hz sampling rate 2.1/4.0 GSP's would be utilized. This extra processing power could be used to increase the bandwidth of the filterbank to 8 KHz, perform amplitude compression signal conditioning, and implement an adaptive filter to perform acoustic echo and feedback cancellation.

The SPROClab development system is an excellent tool for what is termed as the "master hearing aid" in developing digital based clinical audiology. By using the immediate "write" commands, audiologists can quickly determine the optimal frequency curve for their patients. Once the prescription is fitted, the code and coefficients may be downloaded to the individual's hearing aid processor. Likewise as hearing sensitivity changes so may the code, in an equally convenient manner. This application has the advantage of providing a bank of linear phase FIR filters implemented on the parallel architecture of the SPROC signal processor, where most multi-band filters implemented are required to use IIR filters with non-linear phase.

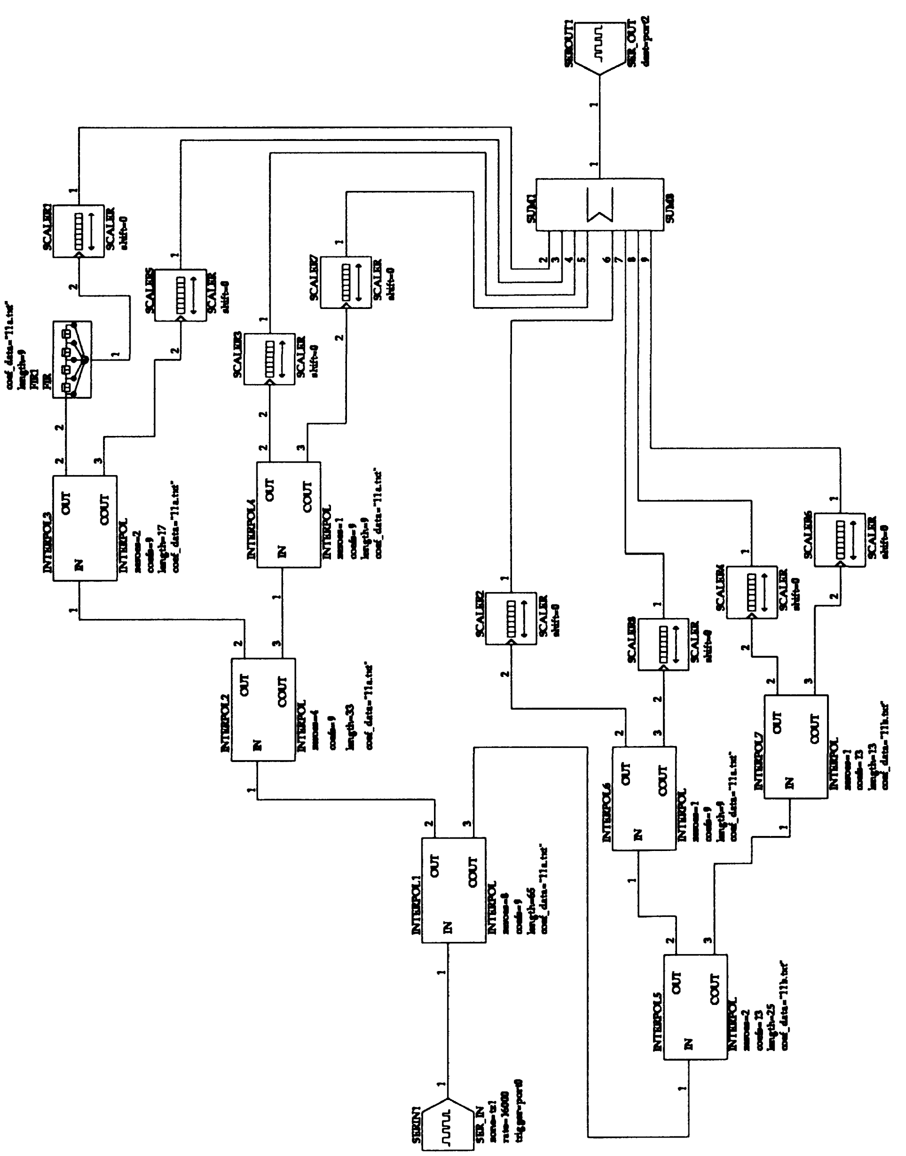

SERIN1
SER_IN
INTERPOL1
INTERPOL2
INTERPOL3
INTERPOL4
INTERPOL5
INTERPOL6
INTERPOL7
FIR1
SCALER1
SCALER2
SCALER3
SCALER4
SCALER5
SCALER6
SCALER7
SCALER8
SUM1
SUM8
SEROUT1
SER_OUT
IN
OUT
COUT

Figure 4

Listing 1

```
/*

   Interpolating symmetrical FIR filter

*/
asmblock interpol (%subr=default,%zeroes, %coefs, %length, %coef_data)
    (in;out,cout)

variable fixed coef_vec[%coefs] = %coef_data;
variable fixed data_vec[%length]=0.0;

duration 12+6*(%coefs-1)+4+3+3*(%length);
begin
        LDA in
        LDL #%coefs-1               //Initialize L register
        LDB #data_vec               //base of data
        STA [B]
        LDF #coef_vec               //base of coef
        LDD #1                      //set up decrement value
//
        LDA B                       //A reg = data_vect address
        ADD #%length-1              //A reg = length-1+data_vec add
        LDB A                       //B reg = A reg = end of data
        LDX [B]                     //X reg = x[n-length-1]
        MPY [F+L]                   //M reg = x[n-length-1]*h[n-coefs-1]
        DJNE LOOP1                  //decrement L reg by 1
//
LOOP1:  LDA B //A reg = B
        SUB #%zeroes                //decrement B reg to next valid data
        LDB A                       //replace B reg
        LDX [B]                     //place next x in X reg
        MAC [F+L]                   //Mult/Acc
        DJNE LOOP1                  //repeat loop
//
        LDB #data_vec
        LDA [B+%length/2-1]         //A reg = x[n-length/2-1]
        SUB MH                      //A reg = yc[n] = x[n-length/2-1]-
y[n]
//
        STMH out                    //filter output
        STA cout                    //complementary output

//
```

```
        LDL #%length-2
        LDB #data_vec
        LOOP2: LDX [B+L]                //data shift
        STX [B+L+1]
        DJNE LOOP2
//
end
```

Spectral Analysis with Applications on the SPROC-1400 Family of Signal Processors

Part 1 of 2

by Scott Andrews

Introduction

This two part application note will review the implementation of a DFT algorithm via the Goertzel approach versus that of a discrete filter bank. Additionally, performance and computation tradeoffs will be discussed in the framework of what the user of these estimation techniques is seeking from the spectrum. Finally, this note will describe a practical application of the theory in the rapid prototype and implementation of a DTMF tone decoder on the SPROC-1400. Part 1 of this note will focus on implementation while part 2 will review the theory of the Goertzel method and other nonparametric methods in use today.

Spectral analysis, also called spectral estimation, encompasses a set of techniques used to estimate/analyze the spectrum of a signal from incomplete observations, that is from finite data. The advent of the digital computer and fast techniques for analyzing the spectral content of a signal (based on the much earlier work of Gauss, Fourier, and Euler) has lead to widespread use of these techniques in a variety of disciplines, including geophysics, medicine, telecommunications, and industrial automation.

Parametric and Nonparametric Spectral Estimation

Spectral analysis by numerical methods remains an inexact science. Over the last two decades, a proliferation of techniques for estimating the spectrum of a signal have emerged. These techniques can be classified into two broad categories -- nonparametric (classical) and parametric. This application note will focus the discussion on nonparametric techniques of digital spectral analysis.

The difference between nonparametric and parametric spectral estimation is simple. Nonparametric methods assume no underlying model for the signal (other than the use of Fourier's general signal representation theorem), while parametric methods assume statistical models for the signal based on statistical assumptions made about the signal. Examples of nonparametric spectral estimation techniques include filter banks and Fast Fourier Transform (FFT) techniques somehow applied directly to the observed signal data. Linear Predictive Coding (LPC) based spectral estimation is an example of a parametric technique.

Filter Bank Methods

In filter bank methods, a parallel arrangement of fixed order IIR or FIR filters are implemented to provide suitable spectral resolution. The outputs of each of the filters are fed into square-law devices and then into integrators to obtain a measure of the energy in each of the spectral regions defined by the filters. This operation is illustrated in Figure 1.

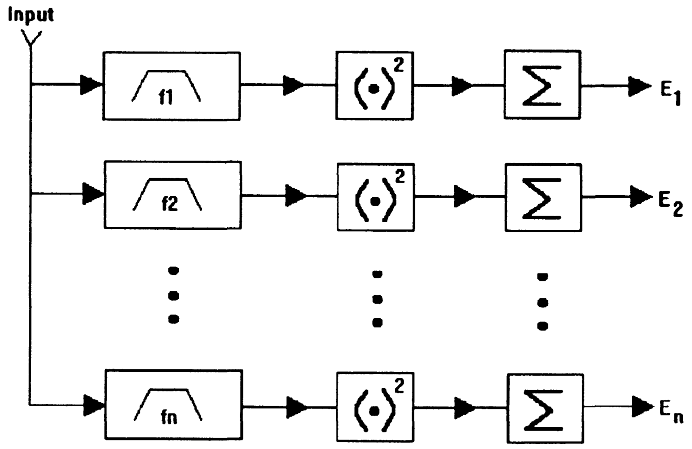


Figure 1: Filter Bank Power Spectral Estimation

All nonparametric spectral estimators can be thought of and derived in consideration of passing the signal through a parallel cascade of narrowband filters (filter bank) and accumulating and measuring output power. The variance of a spectral estimator formed thus is difficult to describe. Depending upon the types of filters used to construct the bank, an expression for the statistical characteristics of this type of estimator can be prohibitive to calculate. Resolution can more or less be chosen based upon the number of filters and the bandwidth of each of the filters. Figures 2 through 9 show simulations created with Matlab. The listings for all the simulations developed in Matlab and discussed in this application note appear in Listing 1. Figures 2 through 5 show, more or less, the variance of the filter bank method with 4th and 8th order Butterworth bandpass filters, and 16th order Yule-Walker derived filters when the input is white Gaussian noise with zero mean and unit variance. Figures 6 through 9 show the resolving power of these same filter banks for a tone of 1336 Hz (commonly used in telecom applications - - DTMF encoding/decoding specifically) and amplitude of 10 V found in the same zero mean, unit variance Gaussian white noise background. It is clear that filter design method and filter order have everything to do with resolving power.

Implementations on SPROC

The two spectral estimation applications presented below have the advantage that they are sample (stream) based, unlike FFT based approaches to broadband spectral estimation, which require all the data be present at the onset of computation (for this reason, FFT based approaches are often referred to as block or vector algorithms). In this sense, the implementations presented below are true real time spectral analysis methods. Specifically, these applications are geared toward tone detection, for a DTMF application which will be presented at the end of the note. It often occurs in practice that only a few well defined spectral features need be observed from the data. Toward this end, the designs which follow offer computationally attractive alternatives to broadband (look at everything) spectral estimation.

Second Order Elliptic Based Filter Bank for DTMF Tones

This design of Figure 10 was sketched and realized in less than one hour. This design fits into less than two GSPs of the 20 MHz part - - SPROC-1400-2 - -leaving two GSPs free for other functions. There are eight filters, one for each of the DTMF frequencies used in telecom. Each of the filters is a second order elliptic filter with four of the frequency responses shown in Figure 11. Code and data resource use were not too severe -- about 225 data locations and 786 code locations were used. Using the new version of the Schedule program reduces the code resource used via subroutines by a factor of six to seven for this design (due to the redundancies). Notable features of the design include a pulse generator used for the integration time constants (which can easily be adjusted on the fly), a bilinear

bilinear cell, bil1, used for a unit delay, and sample and hold circuitry used to hold the spectrum in place while another analysis is being performed. Each of the data sinks represent a data memory location which a controller could easily read at any convenient point in time.

Goertzel DFT Cell for Eight Real Frequencies

Filter banks are appropriate in some cases and not in others. It turns out that when specific resolution must be achieved, it can best be controlled through the use of a selective Fourier based approach, such as the Goertzel approach. Listing 2 displays the implementation of a Goertzel cell for analysis of up to eight real frequencies. The cell has one input, the signal, and nine outputs, eight for the calculated energies at each frequency and one to signal that the analysis is complete for a given number of samples.

Figure 12 contains a design in which the maximum energy value is chosen and an index indicating which frequency was chosen is output. A DOS file named *dftcoef.dat* is read into the cell at compile time. These are the a coefficients of the second order IIR filter of Figure 13 (i.e., the pole values) 2*cos(2pk/N). Changing the frequencies to be analyzed can be accomplished by changing the file before compile time or by changing the harm vector in memory at or beyond run time. A microcontroller or microprocessor could easily strobe through a large set of frequencies to piece together a global view of the spectrum if desired.

The total development effort for the cell was under one week. Developing under the SPROClab is unique in the sense that it allows developers to debug using the proven techniques of the analog world (oscilloscope, function generator, frequency counter, etc.) and/or digital techniques (reading/writing memory locations, simulating GSPs, register monitoring, etc.).

DTMF Application

Figure 14 contains a layout of a DTMF decoder which took less than one hour to sketch. Cells preceded by a $ are either user contributed (uploaded to the Star BBS), unreleased (being tested), or under development (soon to be tested and released). This design utilizes less than a full 20 MHz SPROC-1400-2. On a 50 MHz part, it is anticipated that multiple channels can effortlessly be decoded. Notable features of this layout are the use of two Goertzel cells - - one for the DTMF tones and one to compute second harmonics; a $twist block which computes twist and reverse twist according to AT&T specs; $max4 blocks which compute the max in each tone group and report the energy and the tone index at the output; $harm2 blocks which pick out the second harmonic from a list of harmonics given an address of the tone; and a $and5 gate which signals a valid tone if all inputs are true (logical 1). The data sinks are of length one and can be readily sampled by a system controller (they are DMA locations). The upper and lower sinks give the row and column decoded addresses, respectively, while the middle signals tone valid/invalid.

Summary

The implementations in this note were geared toward narrowband problems (i.e., tone detection). Algorithms such as the FFT are better suited for broadband spectral estimation. Proper care should be exercised when utilizing nonparametric spectral estimation techniques. If possible, it is always better to know what to look for and where to look in the spectrum for desired information. If this is known ahead of time, the more efficient techniques presented in this note may be more applicable and certainly far easier to implement.

Figure 2: 2nd Order Butterworth Bandpass Filter Bank PSD Estimate

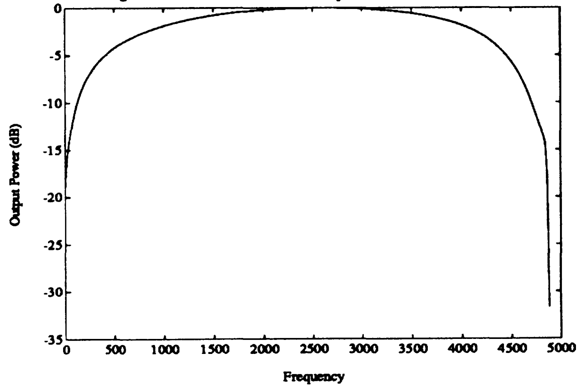


Figure 3: 4th Order Butterworth Bandpass Filter Bank PSD Estimate

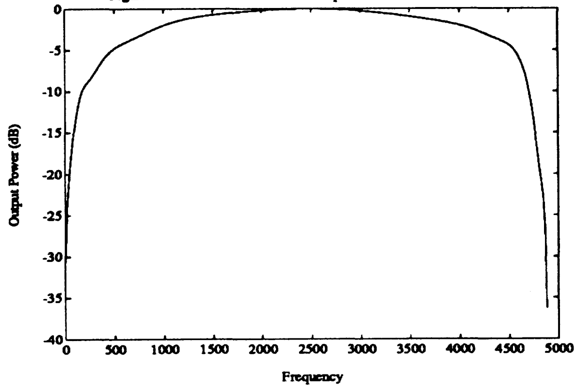


Figure 4: 8th Order Butterworth Bandpass Filter Bank PSD Estimate

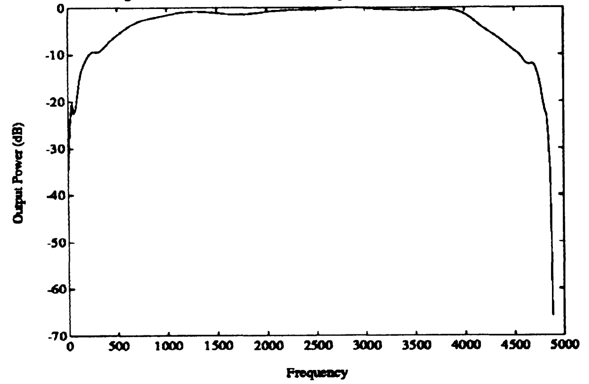


Figure 5: 16th Order Yule-Walker Bandpass Filter Bank PSD Estimate

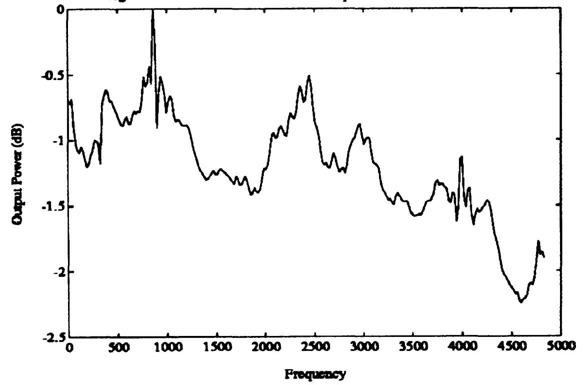


Figure 6: 2nd Order Butterworth Bandpass Filter Bank PSD Estimate

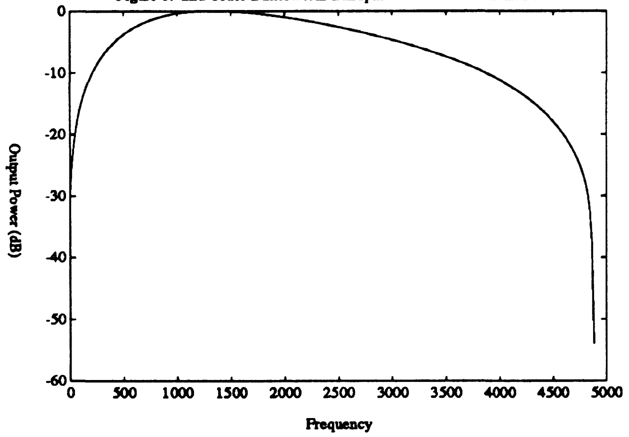


Figure 7: 4th Order Butterworth Bandpass Filter Bank PSD Estimate

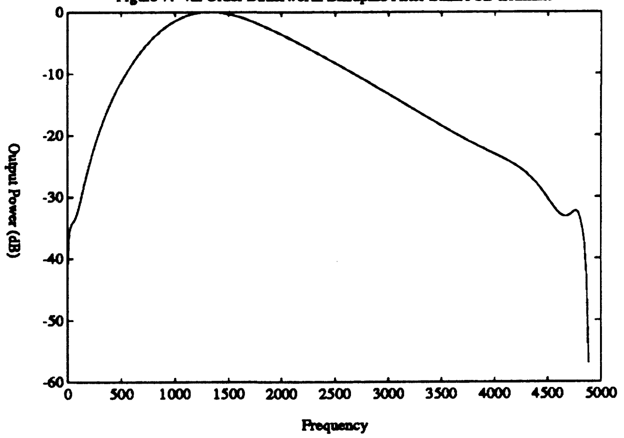


Figure 8: 8th Order Butterworth Bandpass Filter Bank PSD Estimate

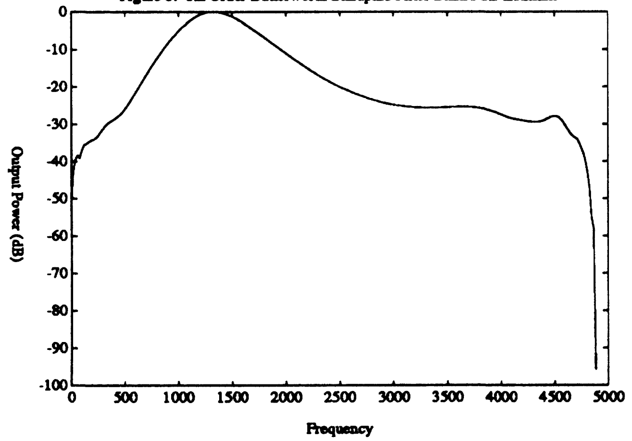
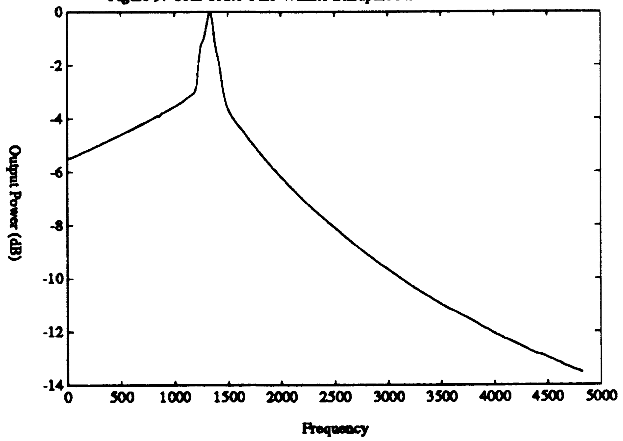


Figure 9: 16th Order Yule-Walker Bandpass Filter Bank PSD Estimate



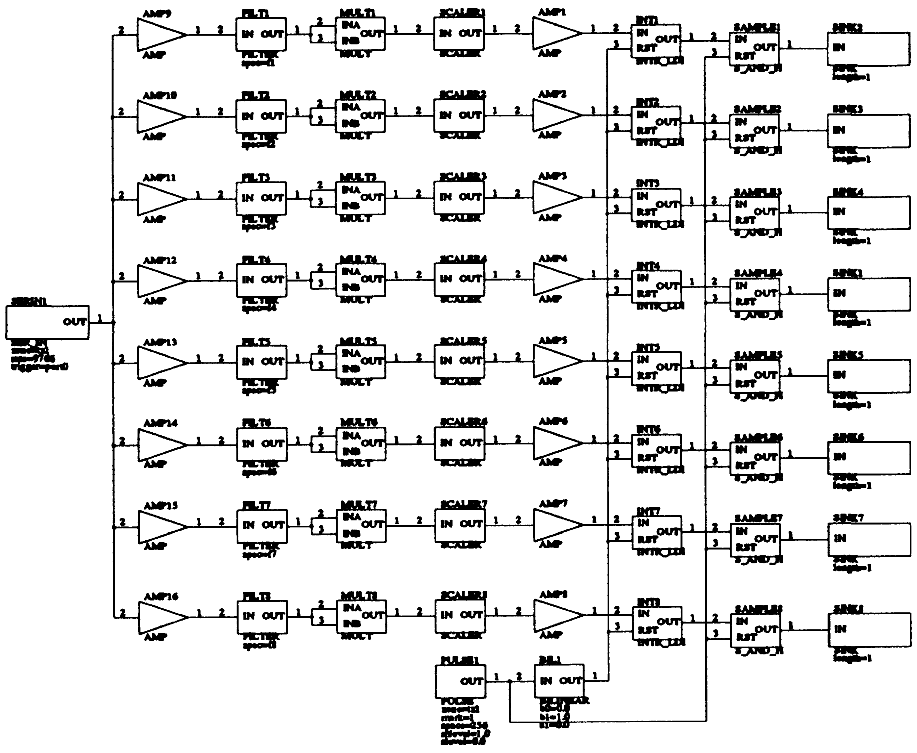


Figure 10. Second Order Elliptic Filter Bank

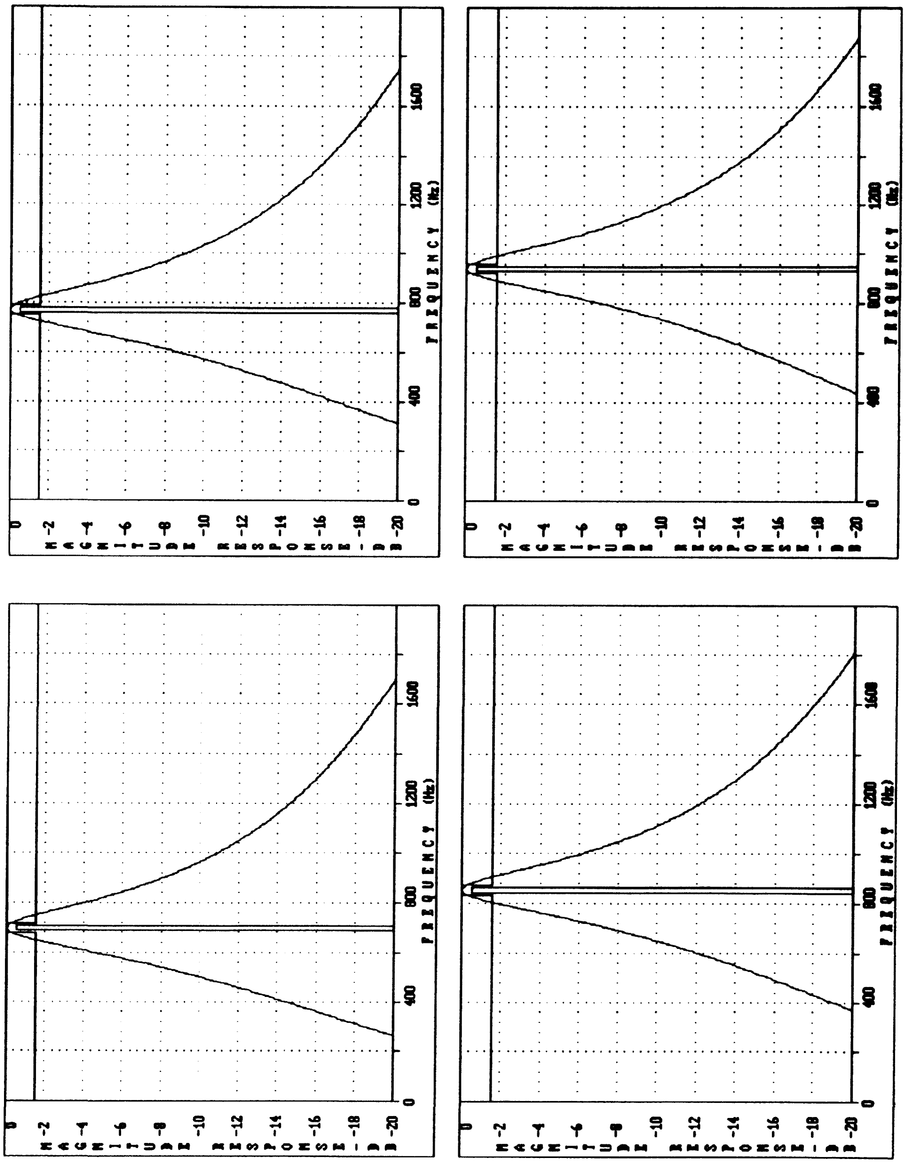


Figure 11. Filter Response Using SPROCfil

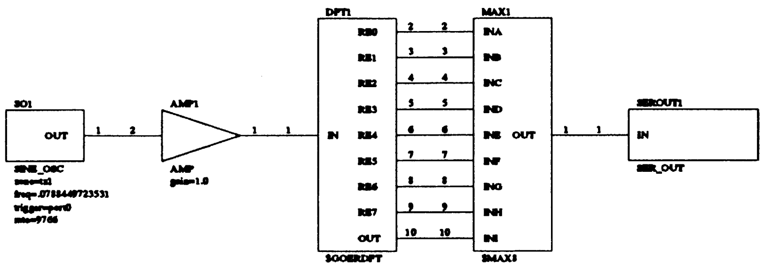


Figure 12. Goertzel DFT Cell

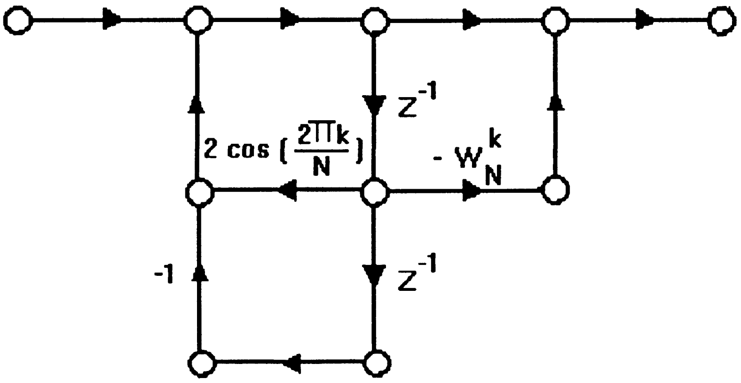


Figure 13. Second Order Goertzel Section

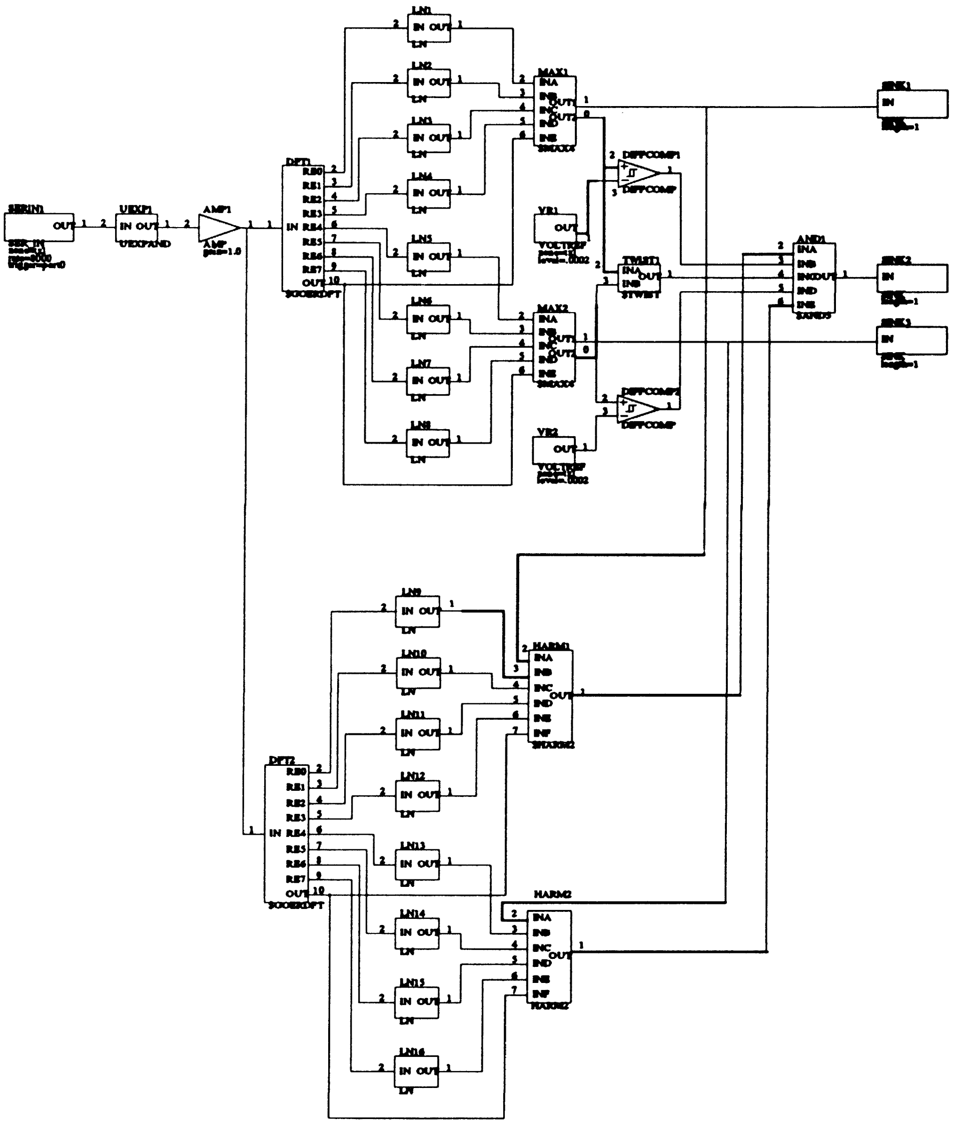


Figure 14. DTMF Decoder

Listing 1.

```
function y = goertzel(nsamp,nfreq,x)
%
%  This Pro-Matlab function implements the real form of the Goertzel
%  DFT algorithm.
%
%  Input parameters:  nsamp - Number of samples on which to run GDFT
%                 nfreq - Number of discrete frequencies
%                 x - Vector of samples of size 1:nsamp
%
%  Function output:   y - Vector of PSD estimates of size 1:nfreq
%
q = 2*pi/nfreq;
for k=1:nfreq
c(k) = cos(q*(k-1));   %  Form cosine vector
end
for k=1:nfreq
         c2 = 2*c(k);
         y2 = 0.0;
         y1 = x(1);
         for j=2:nsamp
           temp = y1;      %  Save y(n-1)
           y1 = c2*y1-y2+x(j); %  y(n-1) = 2cos(2pik/N)*y(n-2)-y(n-2)+x(n)
           y2 = temp;      %  Old y(n-1) -> y(n-2)
         end
         y(k) = c(k)*y1-y2;     %  X(k) = cos(2pik/N)*y(N)-y(N-1)
         y(k) = y(k)*y(k); %  Energy spectrum from real part of DFT
end
%
%  This Pro-Matlab program analyzes the various PSD estimators and
compares
%  them with the true PSD as well as computes MSE between the true PSD and
%  each of the estimators.
%
N = 256;              %  Number of input samples
Pb = 2;               %  Half of order for Butterworth filter bank method
Pyw = 16;             %  Filter order for Yule-Walker filter bank method
fs = 9766;            %  Sampling rate
f = 1336;             %  Tone frequency
dfb = 10;             %  Frequency step for Butterworth method
dfyw = 20;            %  Frequency stop for Yule-Walker method
a = 10;               %  Amplitude of tone
wp = hamming(N)';     %  Hamming window of size N
```

```
wbt = hamming(2*N-1)';  %  Hamming window of size 2N-1;
%
%  Generate tone:
%
t = (0:N-1)/fs;
x = zeros(1,N);

x = a*cos(2*pi*f*t);
%
%  Add noise ~N(0,1):
%
rand('normal');
e = rand(1,N);
x = x + e;
%
%  Butterworth or Yule-Walker based bandpass filter bank:
%
%yfb = bfiltbank(N,N,Pb,fs,dfb,x);
yfb = ywfiltbank(N,N,Pyw,fs,dfyw,x);
%
%  Goertzel DFT
%
yg = goertzel(N,N,x.*wp);
%
%  Window data for periodogram method of classical spectral analysis:
%
yp = x.*wp;
%
%  Set up for Blackman-Tukey method:
%
ybt = xcorr(x).*wbt;
%
%  Plot results
%
fscale = 0:(fs/2)/(N/2-1):fs/2;
fscale2 = 0:(fs/2)/(N-1):fs/2;
Pfb = 10*log10(abs(yfb)/max(abs(yfb)));
Pg = 10*log10(abs(yg)/max(abs(yg)));
Pp = 10*log10(((abs(fft(yp))).^2)/max((abs(fft(yp))).^2));
Pbt = 10*log10(((abs(fft(ybt))).^2)/max((abs(fft(ybt))).^2));
plot(fscale2,Pfb);
title('Figure 9:  16th Order Yule-Walker Bandpass Filter Bank PSD
Estimate');
```

```
xlabel('Frequency');
ylabel('Output Power (dB)');
pause;
plot(fscale,Pg(1:N/2));
title('Figure 16:  Goertzel DFT Spectral Estimate for Tone+Noise
Input');
xlabel('Frequency');
ylabel('Output Power (dB)');
pause;
plot(fscale,Pp(1:N/2));
title('Figure 10:  Periodogram for Gaussian White Noise Input');
xlabel('Frequency');
ylabel('Output Power (dB)');
pause;
plot(fscale2,Pbt(1:N));
title('Figure 13: Blackman-Tukey Spectral Estimate Tone+Noise Input');
xlabel('Frequency');
ylabel('Output Power (dB)');
pause;
plot(fscale2,Pfb,fscale,Pg(1:N/2),fscale,Pp(1:N/2),fscale2,Pbt(1:N));
title('Figure 19:  Pfb (B 4th Order), Pg, Pp, Pbt and True PSD');
xlabel('Frequency');
ylabel('Output Power (dB)');
pause;
hold;
truth;
%
%  Compute mean squared error estimate average and vs. frequency:
%
mse_fb = mean((Ph'-decimate(Pfb,2)).^2)
SEfb = (Ph'-decimate(Pfb,2)).^2;
mse_g = mean((Ph'-Pg(1:N/2)).^2)
SEg = (Ph'-Pg(1:N/2)).^2;
 mse_p = mean((Ph'-Pp(1:N/2)).^2)
SEp = (Ph'-Pp(1:N/2)).^2;
 mse_bt = mean((Ph'-decimate(Pbt(1:N),2)).^2)
SEbt = (Ph'-decimate(Pbt(1:N),2)).^2;
pause;
hold;
%
%  Display squared error of PSD estimators over frequency:
%
plot(fscale,SEfb,fscale,SEg,fscale,SEp,fscale,SEbt);
```

```
title('Figure 22:  Error for Pfb (Y-W 16th Order), Pg, Pp, Pbt Vs.
Frequency');
xlabel('Frequency');
ylabel('Squared Error');
%
% This Pro-Matlab program computes the True PSD of a tone (given by f)
% in a white noise background of unit variance.
%
N = 256;
fs = 9766;
f = 1336;
rand('normal');
e = rand(1,256);
var = std(e)*std(e);
fnorm = f/fs;
c2 = 2*cos(2*pi*fnorm);
Bg = [1.0 -1.0 0.0];
Ag = [1.0 -c2 1.0];
[H,W] = freqz(Bg,Ag,128);
Ph = 10*log10((var.*abs(H).*abs(H)+1)/max(var.*abs(H).*abs(H)));
plot(fscale,Ph);
title('Figure 17:  True Power Spectral Density for 1336 hz Tone + Noise');
xlabel('Frequency');
ylabel('Power (dB)');
function y = ywfiltbank(nsamp,nbands,nord,fs,df,x)
%
% Pro-Matlab function to construct Yule-Walker based filter bank and
% estimate the PSD of the input using the filter bank.
%
% Input parameters:  nsamp - Number of input samples
%
nbands - Number of spectral bands to build filters for
%
nord - Butterworth bandpass filter order = nord
%
fs - Sample rate of data
%
df - Frequency step
%
x - Vector of input samples of size 1:nsamp
%
% Function output:
y - Vector of PSD estimates, size 1:nbands
```

```
%
Bb = zeros(nbands,nord+1);
Ab = zeros(nbands,nord+1);  yf = zeros(nbands,nsamp);
y = zeros(1,nbands);
fndf = df/fs;
%
% Design a bank of filters, filter the data, square and accumulate:
%
F = [0 0 0 0 0 1]; %  Desired frequency breakpoints
M = [0 .5 1 1 .5 0];     %  Magnitude at frequency breakpoints
for i=2:nbands-2
fnorm=(i/nbands);
F(2) = max((fnorm-3*fndf),0.0);
F(3) = max((fnorm-fndf),0.0);
F(4) = min((fnorm+fndf),1.0);
F(5) = min((fnorm+3*fndf),1.0);
[Bb(i,:),Ab(i,:)] = yulewalk(nord,F,M);
yf(i,:) = filter(Bb(i,:),Ab(i,:),x);
end
y = sum(((yf./nsamp).^2)');  %  Integration process
end
function y = bfiltbank(nsamp,nbands,nord,fs,df,x)
%
% Pro-Matlab function to construct Butterworth based filter bank and
%  estimate the PSD of the input using the filter bank.
%
%  Input parameters:  nsamp - Number of input samples
%             nbands - Number of spectral bands to build filters for
%             nord - Butterworth bandpass filter order = 2*nord
%             fs - Sample rate of data
%             df - Frequency step
%             x - Vector of input samples of size 1:nsamp
%
%  Function output:   y - Vector of PSD estimates, size 1:nbands
%
Bb = zeros(nbands,2*nord+1);
Ab = zeros(nbands,2*nord+1);
yf = zeros(nbands,nsamp);       fndf = df/fs;
%
% Design a bank of filters, filter the data, square and accumulate:
%
for i=1:nbands
fnorm=(i/nbands);
```

```
passband = [max((fnorm-fndf),0.0) min((fnorm+fndf),1.0)];
[Bb(i,:),Ab(i,:)] = butter(nord,passband);
yf(i,:) = filter(Bb(i,:),Ab(i,:),x);
end
y = sum((yf.^2)');  %  Integration process
end
```

Listing 2.

```
/*

          Goertzel real DFT cell for eight real frequencies

*/

asmblock $goerdft {} (in;re0,re1,re2,re3,re4,re5,re6,re7,out)

   symbol real8 = 14;                 //Loop count * 2
   symbol length = 226;               //Number of samples for
                                        spectral construction
   variable zero = 0.0;
   variable one = 1.0;
   variable scale1 = 1.0/226.0;       //Scale factor for input
   variable integer count = 0;        //Outer loop variable
   variable harm[8] = "dftcoef.dat";  //File of coefficient data
   variable dftdat[16];               //Second order delay line data
   variable new = 0.0;                //New computation - initialize
                                        data areas

   variable temp;                     //Scaled input temporary variable
//
duration 356;
//
begin
//
   ldb   #harm                        //Set base register to coefficient
                                        store
   ldf   #dftdat                      //Set frame register to DFT delay
                                        line data
   lda   new                          //Initialize?
   sta   out                          //Not ready strobe to output
   jgt   frame                        //If not initialize continue
                                        computation
   lda   one                          //Otherwise intialize
   sta   new                          //New transform setup
   lda   zero                         //Zero accumulator
   ldd   #1                           //Set decrement register
   ldl   #real8+1                     //Load loop to zero 16 locations
zloop: sta   [F+L]                    //Zero DFT delay variables
   djne  zloop
   lda   #length                      //Loop for 9766 ks/s operation (226)
   sta   count                        //226 samples for all harmonics
```

```
    jmp   again                      //All done - ready for 1st sample
                                       next time
  frame: ldd   #2                    //Decrement register for double
                                       decrements
    ldl   #real8                     //DFT loop for all harmonics
    ldx   in                         //x(n)
    mpy   scale1                     //Scale x(n) by 1/226 to avoid
                                       overflow
    lda   L                          //Get L register (14)
    asr                              //Divide by 2 = 7 to initialize
                                       base
    add   B                          //Add address of coefficient store
    ldb   A                          //Start at rear and move forward
    stmh  temp                       //x(n)/226
loop:  ldx   [B]                     //2Cos(2Pik/N)
    mpy   [F+L+1]                    //2Cos(2Pik/N)*y(n-1)
    lda   temp                       //Take new scaled data sample
    sub   [F+L]                      //x(n)-y(n-2)
    sty   [F+L]                      //Move data y(n-1)->y(n-2)
    add   MH                         //x(n)-y(n-2)+2Cos(2Pik/N)
    sta   [F+L+1]                    //New y(n-1)
    lda   B                          //New coefficient address
    sub   #1
    ldb   A
    djne  loop                       //Calculate for all harmonics
    lda   count                      //Countdown to zero
    sub   #1
    sta   count
    jgt   again                      //Keep going until all data is
                                       done
    ldl   #real8                     //Reinitialize to coefficient
                                       store end
    lda   L
    asr
    add   B
    ldb   A
eloop: lda   [B]                     //Begin energy calculations
    asr                              //Cos(2Pik/N)
    ldx   A                          //Cos(2Pik/N)->x register
    mpy   [F+L+1]                    //y(n)*Cos(2Pik/N)
    nop
    nop
    lda   MH
    sub   [F+L]                              //y(n)*Cos(2Pik/N)-y(n-1)
    sta   [F+L+1]                            //->y(n)
    ldx   A
    mpy   A                                  //y(n)^2
    lda   B                                  //Decrement coefficient pointer
    sub   #1
    ldb   A
    stmh  [F+L+1]                            //y(n)^2->y(n)
    djne  eloop                              //Do for all eight frequencies
    lda   [F+1]                              //Store output in each case
    sta   re0
    lda   [F+3]
    sta   re1
    lda   [F+5]
    sta   re2
    lda   [F+7]
    sta   re3
    lda   [F+9]
    sta   re4
    lda   [F+11]
    sta   re5
    lda   [F+13]
    sta   re6
    lda   [F+15]
    sta   re7
    lda   zero                               //Set up for re-initialization
    sta   new                                //Transform complete
    sta   out                                //Ready strobe to output again: nop
end
```

Adaptive Closed Loop Velocity Control for the D.C. Motor

by Darren Castle, Northern Illinois University

The focus of this application is the steady state velocity control of both variable and constant flux D.C. motors, entailing an on-line adaptive control method that offsets the effects of both nonlinearity due to brush friction and load torque.

In order to compensate for these effects utilizing analog components two additional controllers are necessary. In order to offset the effects of the brush friction, minor loop velocity feedback can be included within the closed loop velocity system. In the main loop, the tachometer signal is fed back and compared with the input to yield the error signal. In the minor loop, the tachometer signal is passed through a phase lead network and fed back to the preamp. The idea is to create a dominant pole zero pair to correct the varying time constant pole and linearize the steady state constant.[1] In order to offset the effects of load torque, a transducer can be utilized to provide a load signal as an additional input to the system. The load information in conjunction with a controller minimizes the effect of the load upon the output velocity response.[2]

An alternative to these methods is model reference adaptive control (MRAC), which provides both a linear steady state response and minimizes the load effect without the use of a torque transducer. The purpose is to force the closed loop velocity system to follow an ideal reference model consisting of a steady state constant and a time constant pole. Initially, the controller sends the command input signal to the motor and to the reference model. The outputs of the motor and the reference model are fed into the adaptive cell which minimizes the mean square error function(MSE).[3] This action is performed by first passing the motor tachometer signal through a first order adaptive filter and then subtracting this result from the reference model signal. This error is passed to Widrow's LMS algorithm to update the adaptive filter gain. In this manner the difference between the adaptive filter output and the reference model will iterate to zero. In order to force the steady state output of the motor to match the reference model, the updated adaptive filter gain must be copied to an amp that modifies the command pulse as it passes on to the motor and into the reference model. In this manner adjustments are made to the output command signal to compensate for motor nonlinearites and load torque.

Although the control scheme appears complex upon review, the signal flow diagram in Figure 1 shows how easily the SPROClab development system implements the MRAC system. The pulse cell, PL1, provides the input command signal switching ten seconds high and low for a sampling frequency of 153 Hz. The difference amplifiers, DIFFAMP1 and DIFFAMP2, with the voltage reference cells, VR1 and VR2, provide compensation for the offset of the ADC and the DAC.The transfer function cell, TF1, represents the reference model containing the LaPlace domain coefficients of the ideal closed loop velocity system in the file, *mod1.tff*. During SPROCbuild, the LaPlace transfer function is automatically converted to the z domain equivalent. The adaptive cell, DL1, was created to minimize the MSE function providing adaptive filtering, error creation, and LMS updating of the adaptive filter gain. The output of DL1 is the filter gain coefficient and is passed into a multiplier, MULT1, that modifies the input command signal. The SPROClab system has no problem in handling this feedback of the adaptive gain element.

Real time-testing involves the control of the feedback modular servo system featuring a 24-volt variable magnetic flux D.C. motor that can be configured to run in either armature control resembling a shunt wound motor or for field control resembling a series motor. For this purpose, the motor is configured for armature control, where the effects of brush friction are more noticeable than field control. The reference model in the controller is representative of the steady state and transient parameters established by the minor loop feedback with the phase lead network. The steady state of the model produces .86667 volts per input volt to the system. With the tachometer conversion rate of 367.28 RPM per volt, a one volt input will produce approximately 318 RPM. A comparison of the uncompensated system and the compensated system for input command signals up to 2.0 volts versus the output tachometer signal is shown in Figure 2. The uncompensated system varies as much as 30 RPM from the ideal steady state, while the compensated system varies at the most 0.5 RPM. A comparison of the uncompensated system and the compensated system undergoing varying load for an input of 1.0 volt is shown in Figure 3, depicting output voltage versus load. The load torque is provided by a magnetic brake with a ten point scale with each point equal to 20 Nmm of torque. The uncompensated curve deviates upward of 35 RPM from the unloaded position while the compensated curve shows no deviation at all. This does not mean that the load curve can not deviate from the unloaded position. It must be remembered that the adaptive filter is steady state compensation only. A load must remain constant long enough for the adaptive filter to compensate. This length of time depends upon the time it takes for the motor transients to clear, 0.250 seconds for this system. To achieve greater control, the adaptive filter size must be increased and trained with a random white noise signal. As the steady state controller only demands 0.1 of a single general signal processor, the SPROClab system allows room for transient improvement.

REFERENCES:

[1] Modular Servo System MS150, Feedback Instruments Ltd., Crowborough, Sussex, England.

[2] Kuo, Benjamin C.; "Automatic Control Systems", 5th Ed., Prentice Hall Inc., 1987.

[3] Widrow, B.; Stearns, S.; "Adaptive Signal Processing", Prentice Hall Inc., 1985.

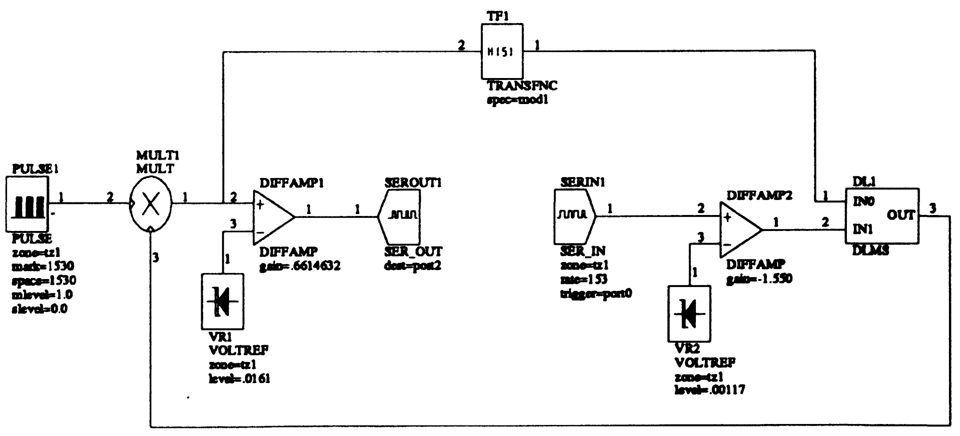


Figure 1

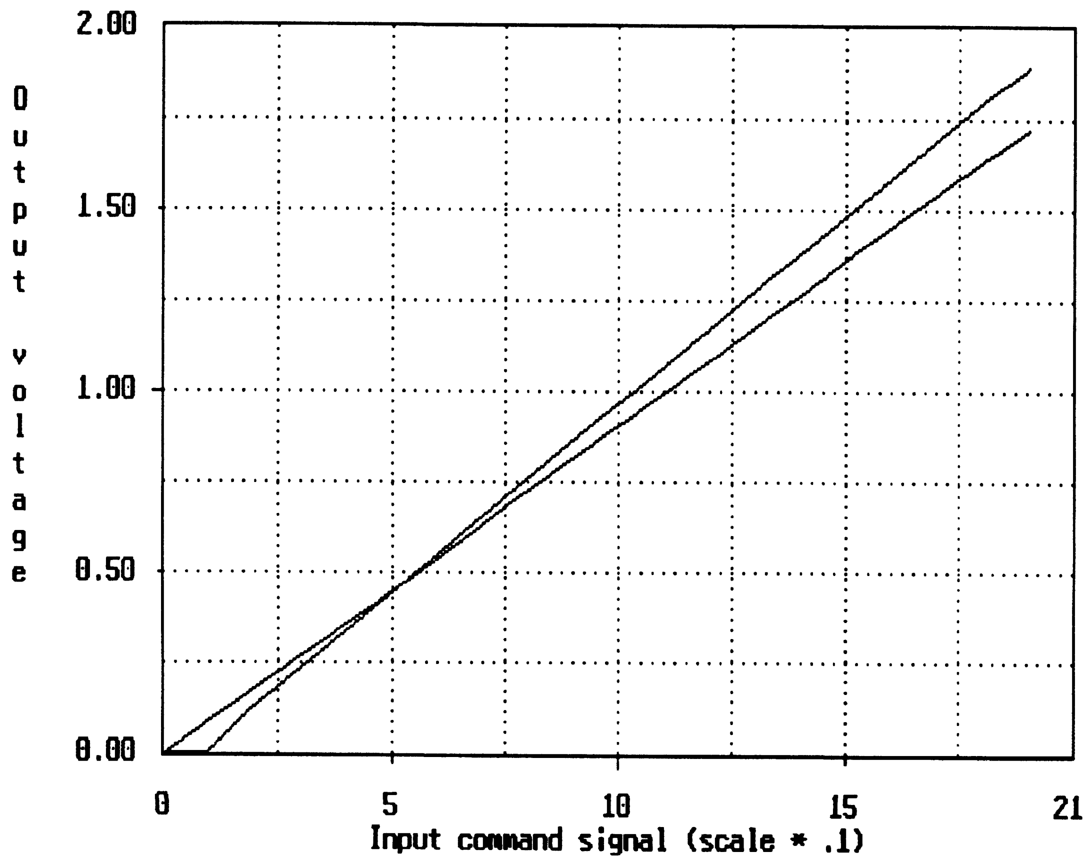


Figure 2. Linearization of Steady State, Compensated and Uncompensated

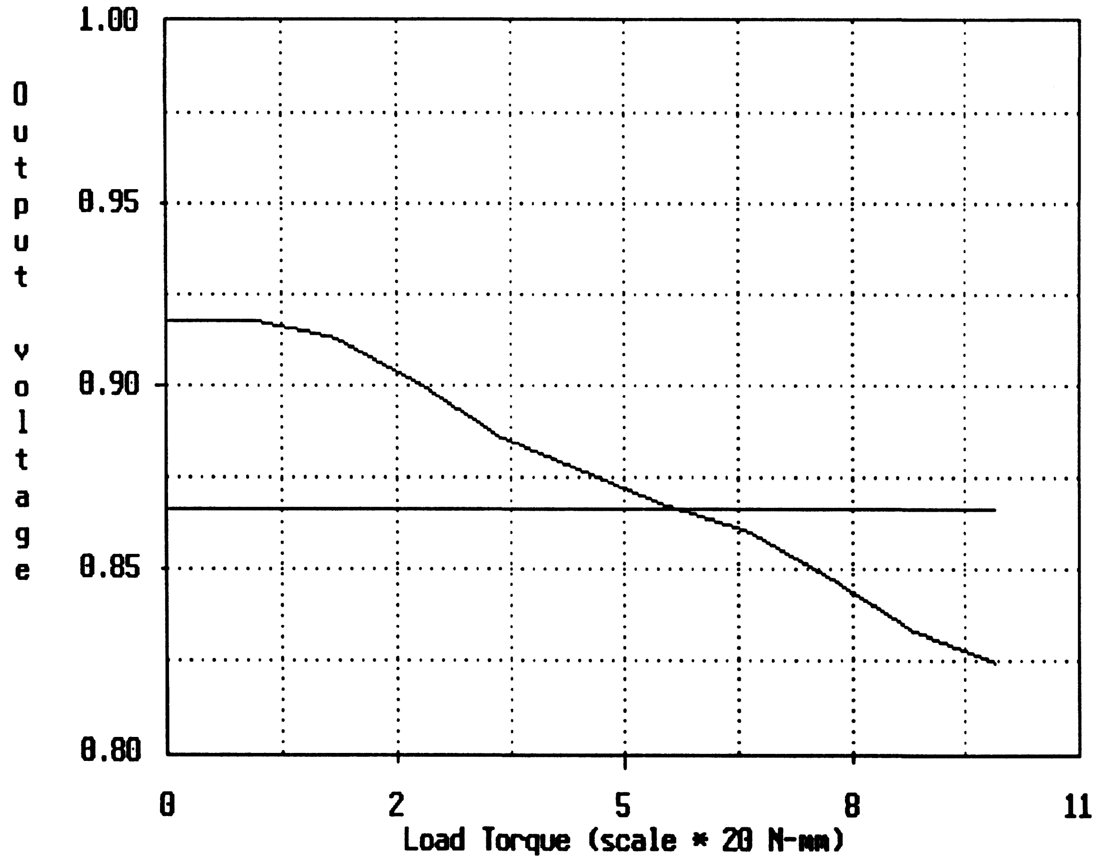


Figure 3. Minimization of Load Effect for 1.8 volt Input

Listing 1. DLMS.SDL

```
asmblock dlms {%mu} (in0,in1;out)
duration 18;
variable fixed en;
variable fixed wn=1.0;
variable fixed mu=%mu;
begin
        LDX in1
        MPY wn
        NOP
        NOP
        LDA in0
        SUB MH
        STA en
        MPY en
        NOP
        NOP
        LDX MH
        MPY mu
        NOP
        NOP
        LDA wn
        ADD MH
        STA wn
        STA out
end
```

Hands-Free Telephone Convener

by Dan Greenwood

The development of faster and more efficient signal processors has led to more advanced telecommunication systems. Likewise adaptive filtering techniques have enhanced telecommunication systems by performing such functions as line and acoustic echo cancellation. This application note will present a SPROC system which may be used in parallel with an existing telephone. The convener allows the user to implement a hands-free telephone system once a call is initialized.

A central problem associated with hands-free telephones is the inherent instability due to the closed loop path caused by acoustic coupling between the system speaker and microphone. A traditional solution to this problem is the use of analog echo suppressors. The echo suppressors are voiced activated switches which open the closed loop path to prevent instability. One problem with echo suppression is that the systems are limited to be half-duplex. Therefore, both individuals may not speak simultaneously. A second disadvantage to echo suppression is the noticeable click due to the opening and closing of the switches which usually cannot respond quickly enough to catch the beginnings of speech utterances.

Adaptive filters may be used, however, to implement echo cancellers which allow full-duplex communication since the echo path is never opened. In a conventional hands-free system there are two forms of echo to cancel. Acoustic echo is caused by the coupling of the speaker output with the system microphone. Acoustic echo consists of two portions: short and long echo. Short echo is due to the direct coupling between the speaker and microphone due to reflections from near surfaces such as a table top. Long echo is due to reverberations of the speaker output from the room's walls and objects which are not close to the system. Typically, long echoes may have a delay in the order of hundreds of milliseconds. The second form of echo is line echo which is induced by mismatched impedances between the two connected users. Both forms of echo should to be addressed to insure the system stability and provide over all improved sound quality.

Among the functions incorporated into the system will be adaptive line and acoustic echo cancellers, speech detectors, and a closed loop gain controller. The closed loop gain controller will perform such functions as the management software attenuators in the closed loop path and the update eligibility for each of the echo cancellers. The two speech detectors will provide the needed information for the closed loop controller to make a decision as to the current operational mode of the system, i.e. transmit, receive, double talk, and idle. Figure 1 shows a block diagram of the complete hands-free convener system. The SPROC's four GSP's may be used in parallel to implement the variety of functions in the telephone convener system. That is, while certain GSPs are performing echo cancellation the remaining processors can implement such tasks as speech detection, power estimates, and software attenuation calculations. The SPROC's data flow architecture is ideal for this type of process since no processing power is required for such random processes as interrupts. The processors can be used almost entirely for pure signal processing operations. The application note will include real-time test results, new SPROC cell developments, and implementation notes.

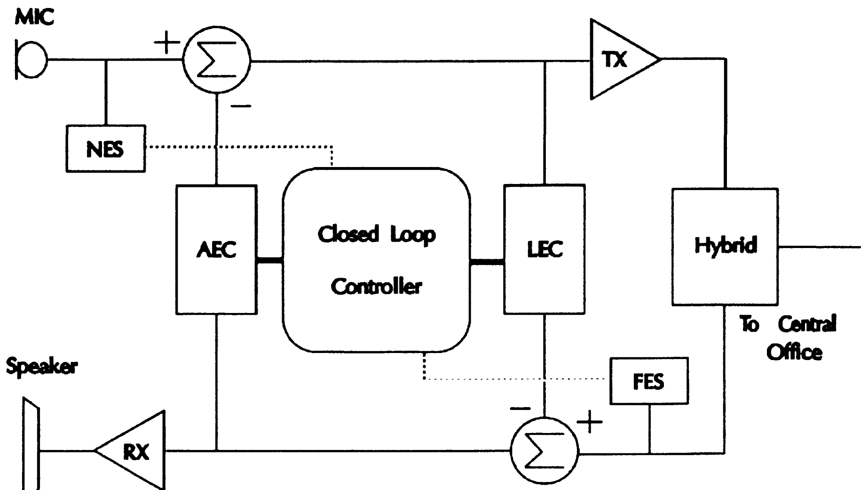


Figure 1. Closed Loop Convener System

Application of an Adaptive Equalizer Using the SPROC Development System

by Steve Voepel and Sen Kuo, Northern Illinois University

Abstract

This application demonstrates the ease of real-time development with the SPROClab development tools by way of a common adaptive equalizer which is necessary in most communication systems. The type of equalizer to be used is known a decision feedback system. A block diagram is given in Figure 1.

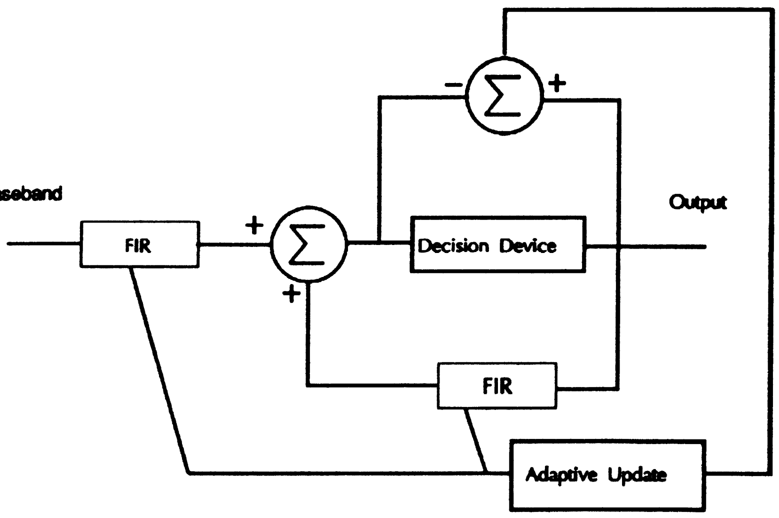


Figure 1. Block Diagram of Adaptive Channel Equalizer

A typical channel response, which simulates a degraded communications channel in the form of a low pass filter, can be expressed as:

$$C(z) = 0.3 + 0.9z^{-1} + 0.3z^{-2}$$

Figure 2 shows the entire adaptive equalizer system. By using the SPROC development system, the degraded channel and corresponding adaptive inverse filter, which will be named DFE, can be realized by choosing existing cells in the SPROC system and developing one new cell. The channel response will be implemented using the FIR cell and an ASCII file containing the coefficients given above.

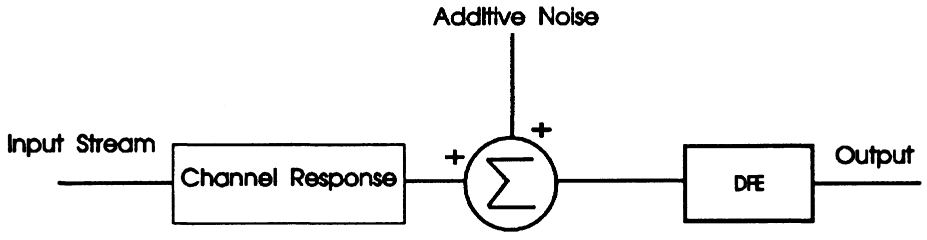


Figure 2. Channel Equalizer System

The adaptive cell DFE will be trained using white noise from a noise generator through one of the two serial input ports of the SPROC. The final system output can be viewed either using the SPROC's probe output port (8-bit resolution) or one of the two serial output ports with 16-bit resolution.

The SPROCdrive interface will allow the channel model to be varied during testing. This advantage will allow us to view a real-time adaptive response of the channel equalizer. With the ever increasing demand for fast development time for products, the SPROClab development system can provide a realistic and flexible means to achieve this goal.

Spectral Analysis with Applications on the SPROC-1400 Family of Signal Processors

Part 2 of 2

by Scott Andrews

Abstract

This second part of a two part application note will review the theory of the Goertzel method and other nonparametric spectral estimation techniques in use today.

Topics will include:

- Brief review of statistical estimation theory
- The PSD and the Wiener-Khinchin relations
- The periodogram spectral estimator
- Blackman-Tukey approach to spectral estimation

Section 5
Technical Support

Training Courses

STAR Semiconductor offers comprehensive technical training for SPROC signal processor users. Course include a detailed description of the SPROC architecture, a review of digital signal processing theory, an in-depth study of SPROClab hardware/software tools, and "hands-on" applications development workshops utilizing the entire suite of SPROClab design tools.

Courses run for two days, from 9 AM to 5PM each day. Course notes, binders, and lunches are provided by STAR Semiconductor. A substantial amount of class time is spent performing design exercises using the SPROClab development system (two students per system). Development systems are also available to students following classes for optional individual work in consultation with the instructors.

Prerequisites

Students are assumed to have a background in signal processing design and familiarity with PCs and MS-DOS.

SPROClab Training Course Outline

I. STAR Semiconductor

- profile and mission

II. SPROC Technology

- architecture
- software/hardware optimization

III. Theory of Digital Signal Processing

IV. SPROClab Development System

- hardware
- software

V. SPROClab Applications Examples

-tutorials

VI. Customer Specific Applications

- system requirements
- the SPROC approach

VII. Hardware Design Considerations

- functional description
- timing requirements

Training Seminars

In addition to 2-day training courses, STAR Semiconductor offers half-day training seminars at various locations around the United States. Contact your nearest STAR Semiconductor sales office for information regarding times and locations.

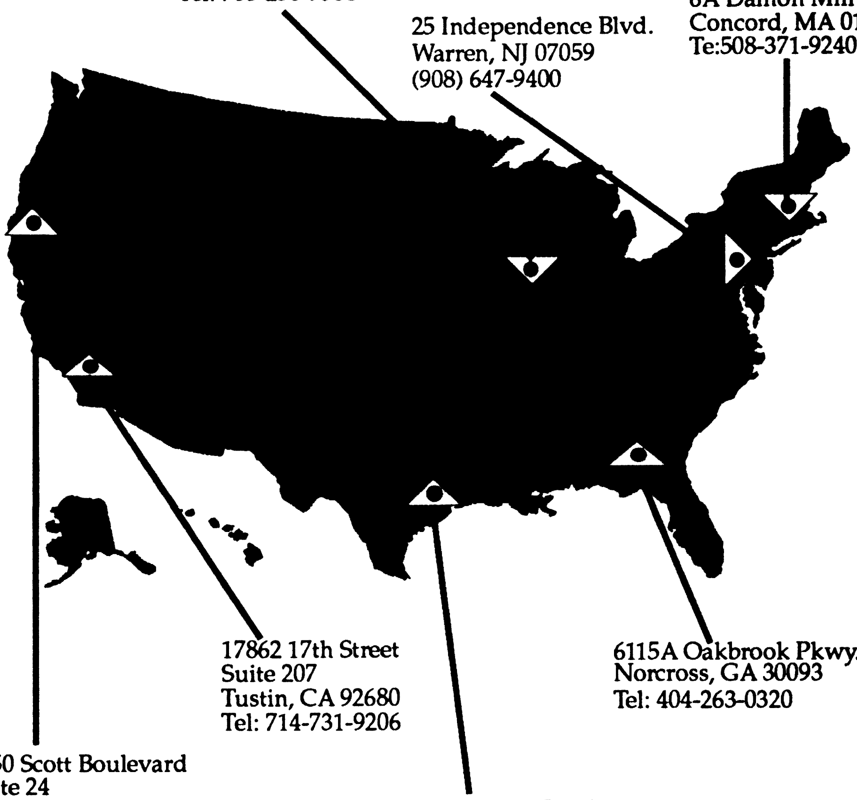
Field Technical Specialists

STAR Field Technical Specialists (FTSs) are available at the locations shown on the following page. Additional technical support is provided by headquarters applications who can be reached at

(908) 647-9400

In addition, a world-wide network of STAR Semiconductor representatives and distributors also provide technical support.

FTS Locations



1009 Hawthorne Drive
Itasca, IL 60014
Tel: 708-250-9586

25 Independence Blvd.
Warren, NJ 07059
(908) 647-9400

6A Damon Mill Sq.
Concord, MA 01742
Te:508-371-9240

17862 17th Street
Suite 207
Tustin, CA 92680
Tel: 714-731-9206

6115A Oakbrook Pkwy.
Norcross, GA 30093
Tel: 404-263-0320

3350 Scott Boulevard
Suite 24
Santa Clara, CA 95054
Tel: 408-727-7707

12000 Ford Road
Suite 200
Dallas, TX 75234
Tel: 214-241-3505

SPROClab Evaluation Board
REFERENCE
MANUAL
Star—The Signal Processing Company

Technical Documentation

A SPROClab development system document set is delivered with every SPROClab development system. The document set includes the *SPROClab Development System User's Guide,* containing the following sections:

Section 1. Getting Familiar with the Development System

Section 2. Starting the Development System

Section 3. Entering Diagrams

Section 4. Defining Filters

Section 5. Defining Transfer Functions

Section 6. Converting a Design

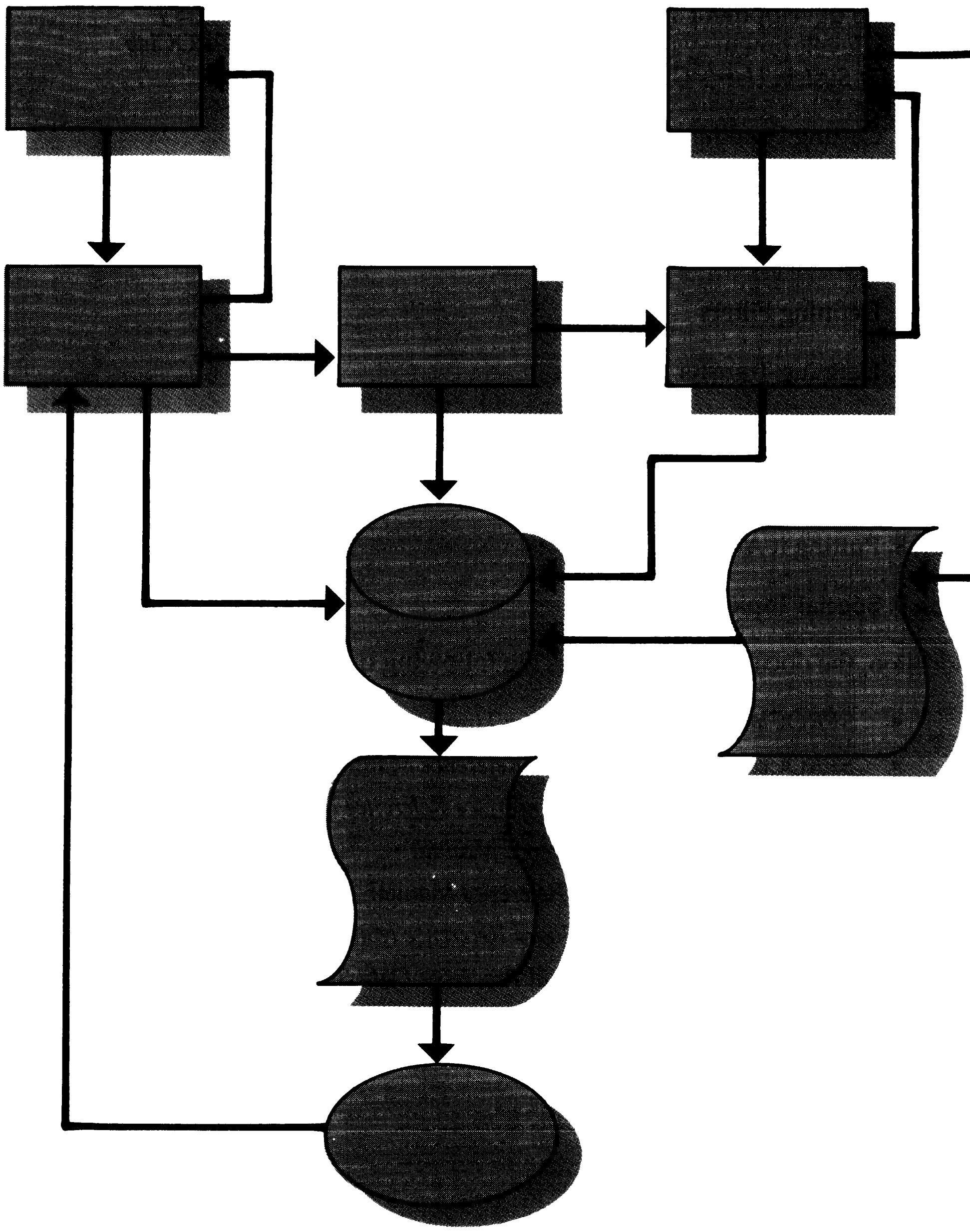
Section 7. Loading, Running, and Debugging a Design

Section 8. Putting it All Together

Section 9. Special Topics

In addition, the document set includes the following manuals:

- *SPROClab Development System Unpacking and Installation Guide*
- *SPROCcells Function Library Reference Manual*
- *SPROClink Microprocessor Interface Reference Manual*
- *SPROCdrive Interface Reference Manual*
- *SPROCbox Interface Unit Reference Manual*
- *SPROCboard Evaluation Board Reference Manual*
- *SPROC Programmable Signal Processor Data Sheet*
- *SPROC Description Language Reference Manual*



Section 6
Quality, Testing, Packaging/Handling

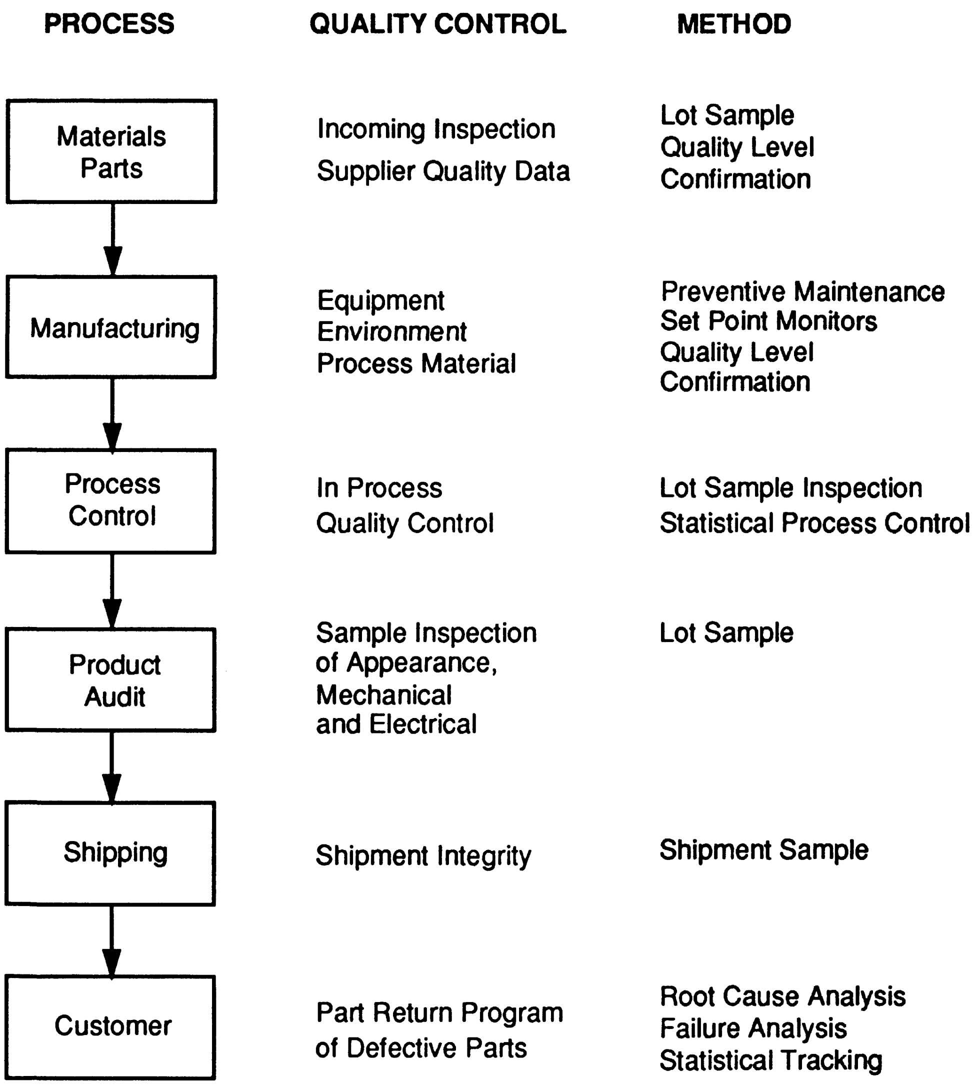
Quality and Reliability Assurance

STAR Semiconductor utilizes a state-of-the-art, CMOS VLSI process technology in the manufacture of its programmable signal processors. The quality and reliability of these circuits is assured through the strict adherence to a rigid set of quality and reliability conformance specifications.

Quality Control of Manufacturing

All silicon processes have been developed to meet specific quality and reliability goals. During the development process, test vehicles are used to verify that the targeted reliability goals are achieved. Electromigration, gate oxide integrity, and process defect inspection are examples of the tests that are conducted during the process development phase.

Once a process has been developed, sample parts are then subjected to a series of reliability tests to again confirm that the targeted goals have been achieved. These tests include life test, thermal stress, moisture resistance, and mechanical stress. Further testing of electromigration and oxide integrity is also done. Details of how these tests are performed are included in the *Reliability Methods* portion of this section.



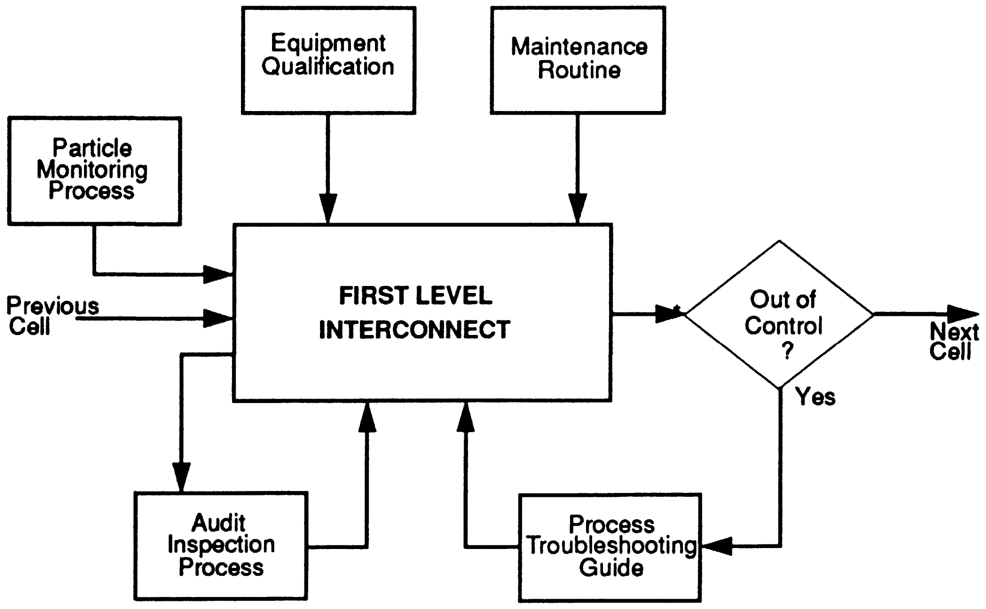
PROCESS	QUALITY CONTROL	METHOD
Materials Parts	Incoming Inspection Supplier Quality Data	Lot Sample Quality Level Confirmation
Manufacturing	Equipment Environment Process Material	Preventive Maintenance Set Point Monitors Quality Level Confirmation
Process Control	In Process Quality Control	Lot Sample Inspection Statistical Process Control
Product Audit	Sample Inspection of Appearance, Mechanical and Electrical	Lot Sample
Shipping	Shipment Integrity	Shipment Sample
Customer	Part Return Program of Defective Parts	Root Cause Analysis Failure Analysis Statistical Tracking

Simplified Process Flow Chart and Manufacturing Quality Controls

Manufacturing Process

Silicon Fabrication

Within the silicon fabrication area, process activities are subdivided into work cells. Each of the work cells has defined quality control activities which include equipment qualification, preventive maintenance, particle monitors, quality inspections, and statistical process control.



Example of a Workcell and Its Defined Quality-control Activities

Statistical process control procedures have been implemented throughout manufacturing to immediately identify potential process problems. Correction of the root cause of these problems is part of continuous process improvement.

Package Assembly

The package assembly operation also utilizes the work-cell quality-control concept. Statistical process control is the predominant tool used. When a process is "out of control", a documented procedure is followed to return the process to control. When a set-up is changed, specified procedures again ensure that the machine is production worthy and in control prior to releasing it to run production parts.

"Just-In-Time" Convergence has furnished an opportunity to provide fast feed-back of quality information. Each cell has a minimum of inventory so that quality problems are detected quickly and corrected.

Each work cell also has documented procedures on the operation of the equipment. Operators are trained and certified to these procedures. Maintenance procedures are also developed and documented for each of the cells and play a key role in producing high-quality products.

Electrical Test

STAR Semiconductor utilizes two types of testing: parametric testing for process/device characterization, and functional testing for IC verification. The former tests the way an IC is fabricated and is design independent while the latter tests characteristics about the actual design itself, i.e., part functionality. STAR Semiconductor ICs are subjected to both kinds of testing to make sure that the design is robust and yielding within the capabilities of the fabrication process, and that the fabrication process is working properly.

Parametric Test

To measure the process/device characteristics, structures such as individual transistors, contact strings, or diffusion resistance are used. These structures are located in the scribe lines on the wafer. Several wafers from each lot are sampled. If the wafers do not meet the device specifications, the entire lot is rejected.

Functional Test

To measure characteristics about the design itself (IC verification), full functional testing is performed. There are four major parts to this test: open, short, and I/O pin testing; functional testing; I/O leakage current testing; and supply current testing. Functional testing is usually performed at full device operational frequencies with the voltage levels adjusted to compensate for maximum temperature sensitivities.

Electrical testing is performed at both the wafer level and the final packaged part level. Wafer testing is used to assure overall device reliability while package testing is designed to detect packaging-induced failures.

In addition, a screen at the wafer test level checks for wafers that do not meet the "ship limit". If a wafer has fewer than 25% of the standard number of good die per wafer, it is scrapped. This procedure prevents potentially marginal die from being shipped to the customer and the field.

Before a wafer is shipped, a visual inspection is performed. This screening test is designed to detect gross visual defects that would affect metal or passivation quality.

Package testing consists of the same functional test as that performed at the wafer level. In addition, lead planarity and alignment testing is performed on surface-mount devices. Visual inspection is also performed to check for bent leads or other gross defects.

Work cell controllers in the test area perform real time statistical process control functions. Process, parametric, and functional test data are also collected and forwarded to a global database. The database allows engineers to cross correlate fabrication data and test results in an integrated environment. The results of the analysis are used to identify root causes and initiate process improvements.

Each tester is linked to a central process control computer that analyzes the yield results of every wafer and package as it is tested. The system detects statistically significant yield variations (that are often too subtle to be recognized by people) and reports them to the operator along with a knowledge-based rule for action. The operator uses this information to either restore control of the test process, or dispose of the deviant material. Thus, the system helps manage the test process to achieve the highest yields possible.

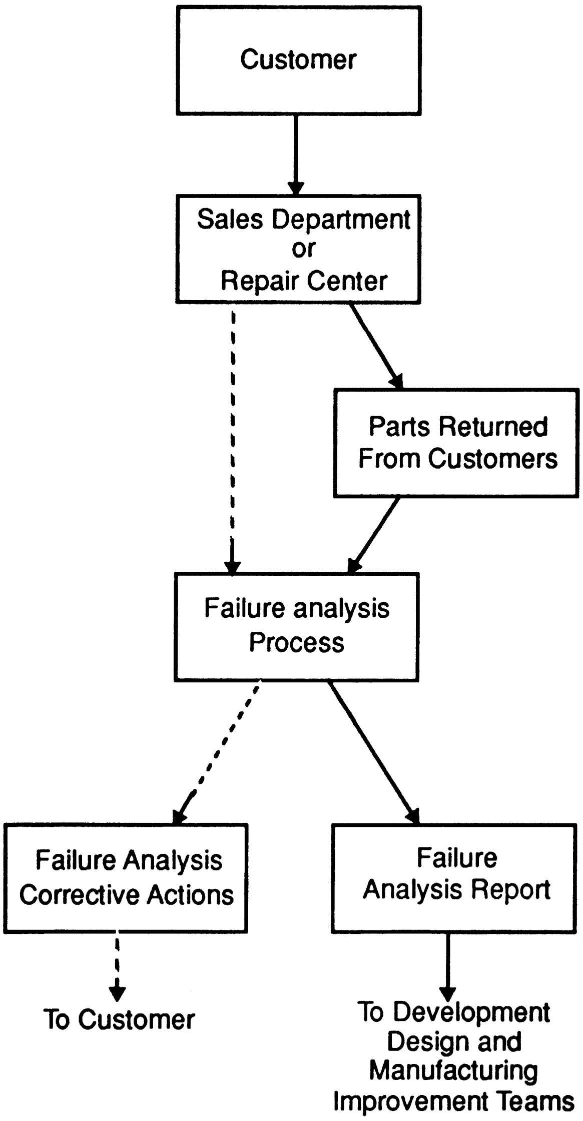
The system also summarizes the daily out-of-control events and produces reports that help focus the daily management activity of the engineering teams. This procedure means a rapid, focused response to statistically significant process deviations. The tools of this portion of the system include: cross-correlation matrices, wafer map, and trend analysis.

Shipping

The packing and shipping area also utilizes quality-control processes. All materials that ; used in shipping are subjected to testing to ensure the products will arrive at the custome location with no damage. Before these materials are utilized in the shipping process, the are evaluated for their ability to prevent ESD and handling damage. Periodic tests are conducted to verify that the shipping materials and process continue to meet the quality objective. All shipping documents are automatically generated from the customer's ord All information needed to correctly pack and ship the material is included on these documents. During the packing process, the parts being shipped are verified against the documents to ensure the correct part is being sent to the customer. Parts are shipped to arrive at the customer's site on the requested date of delivery.

Part Return Process

STAR Semiconductor customers are included in the quality and reliability improvement process. Parts which fail for any reason are returned to STAR Semiconductor for analysi This analysis, along with information received from yield improvement analysis and on-going monitors, is used to improve the development, design, and manufacturing processes.


Customer
Sales Department or Repair Center
Parts Returned From Customers
Failure analysis Process
Failure Analysis Corrective Actions
Failure Analysis Report
To Customer
To Development Design and Manufacturing Improvement Teams

Part Return Process Flow Diagram

Reliability Methods

Definition of Reliability

Reliability is continued performance to specification over time, that is quality sustained over an arbitrary period.

Field failure collection and analysis does not provide sufficiently rapid feedback to effectively initiate process reliability improvements. Long-term life results, however, can be related to field performance using the Arrhenius model. This model treats mechanisms which are accelerated by temperature, and which obey a rate equation whose key parameter is activation energy, Ea. Ea may be different for different failure mechanisms. Activation energy can be determined experimentally for any given process; the literature reports values for common failure mechanisms, such as gate oxide breakdown. If an analysis of a STAR CMOS device identifies a failure mechanism, then standard values for associated activation energies are assumed in the calculations. The Arrhenius model is used to derate Long-Term Life test data from 160 degrees Centigrade junction to lower temperatures. The failure rates are expressed in FITS (FITS= Failures/10^9 device hours).

Process yield and early-life reliability are related, since both depend on low levels of processing defects. It has been observed historically that as process yields are improved, process reliability also improves. A fair comparison of failure rates will demonstrate that a newer design often replaces many circuit components for a combined substantial improvement in overall system reliability.

The following tables describe what tests are performed for process and product qualification.

TEST	PURPOSE	CONDITONS	SAMPLE SIZE/ FAILURE LIMITS	REFERENCE
Reliability				
Long Term Life	To demonstrate the quality or reliability of devices subjected to specified conditions throughout an extended time period	1000 hours, 150°C, dynamic operation	129/1	MIL-STD-883C Method 1005.7 Test Condition A
Moisture Resistance	To evaluate the resistance of devices to deterioration caused by high humidity, heat, and electrical biases.	1000 hours, 85°C, 85% humidity, static bias	105/2	MIL-STD-883C Method 1004.7
Pressure Pot (Autoclave)	To determine the ability of plastic encapsulated devices to withstand accelerated humidity, pressure, and temperature	240 hours, 120°C, 100% relative humidity, 2 atmospheres absolute pressure, no bias	50/10	
Thermal Shock	To determine the resistance of devices exposed to extreme temperature changes	200 cycles, -55°C to +125°C, room pressure, no bias	116/0	MIL-STD-883C Method 1011.7 Test Condition B
Quality				
Solder Process Resistance	To determine the ability of devices to withstand a typical wave solder operation	1-5 cycles wave solder exposure	32/0	
External Visual	To determine the level of workmanship, damage, and external defects of the devices	3-30x visual inspection		MIL-STD-883C Method 2009.8
Electrical Test	To determine how well the devices meet the product specifications throughout a rated temperature range	-Functional: design specific -Parametric: standard shell -Temperature: design specific		MIL-STD-883C Method 3014.1

The tests listed in this table are a sample of those performed.

TEST	PURPOSE	CONDITIONS	SAMPLE SIZE/ FAILURE LIMITS	REFERENCE
ESD	To provide specific test procedures to assure that part performance will not be degraded, to an unacceptable level, by exposure to a succession of electrical discharges below a minimum voltage	- Room temperature - +/- 1500V to I/O pin - Functional test	2/0 at each point	MIL-STD-883C Method 3015.7
Latch-up	To provide specific test procedures and acceptance criteria to assure that CMOS devices are immune to latch-up	-Supply voltage +/- 4V - I/O current +/- 125mA <or> Pad voltage of 9.5V minimum for a triple-level-metal CMOS process	2/0 at each point	EIA JEDEC Publication 12
Part Qualification	To provide specific test procedures to determine potential failure mechanisms caused by die-package interactions. (These tests are not required for every device.)	- Various: includes gross leak, solderability, thermal shock, ESD, structural analysis, autoclave, and others		MIL-STD-883C Various Methods
Margin Check	To provide specific test procedures to determine potential failure mechanisms caused by design-process interactions	- Process skew lot split at gate critical dimensions - Functional test		

Quality and Reliability Monitors

Outgoing Quality Audit

As a final quality check of products, random samples are selected from inventory and subjected to visual and mechanical tests. These tests verify that the manufacturing process is in control .

Reliability Monitoring

To ensure that the manufacturing process is reliable, monitor procedures are conducted. The monitor is a memory device which has been fabricated in the qualified and controlled production process. Each month sample memory device parts are subjected to the tests shown below.

TEST	CONDITIONS	DURATION	REFERENCE
Life	150°C, 7 volts	1000 hrs	MIL-STD-883C Method 1005.7
Thermal Shock	-55°C to +125°C	1000 cycles	MIL-STD-883C Method 1004.7
HAST (Highly Accelerated Stress Test)	125°C, 2 atm	168 hrs	

If a failure should occur during any of these tests, the part is failure analyzed to determine the root cause. These data are then used to improve the production process to prevent potential reliability failures.

CMOS User Precautions

Handling

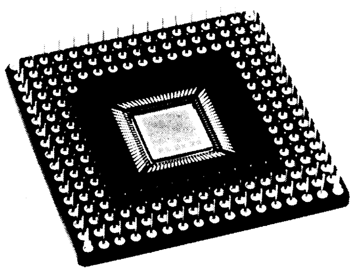
CMOS devices are sensitive to static electricity. Parts are shipped in static tubes and containers. Parts should be left in these tubes and containers until installed on printed circuit boards. Assembly equipment and work stations need to be designed to eliminate any static potential from coming into contact with CMOS parts. Operators need to be static safe before handling CMOS devices.

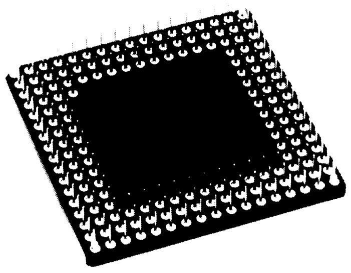
Packaging Information

STAR Semiconductor's line of SPROC-1000 series programmable signal processors are available in pin grid array (PGA), quad flat pack (QFP), and leaded chip carrier (LCC) packages.

Pin Grid Array Types

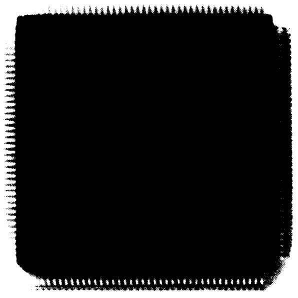
The PGA types are available in 132-pin ceramic packages and in 96-, and 132-pin plastic packages. The ceramic package has a co-fired ceramic design that permits use in applications where high electrical and high temperature performance are important considerations. The packages imbedded heat slug (option) assures greater thermal range without the need for external heat sinking.


(100-mil pitch)
CPGA
Ceramic Pin Grid Array


PPGA
Plastic Pin Grid Array
(100 mil pitch)

Quad Flat Pack Types

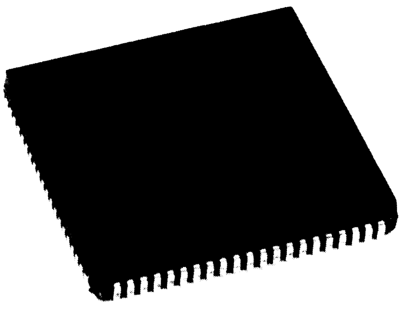
The QFP types are available in 100-, 144-, and 160-lead plastic packages. The QFP type is an EIAJ standard, high-density, high-lead-count package with gull-wing lead formation on all four sides. The gull-wing leads with their high pitch permit greater lead compliance and ease of lead inspection in surface-mount commercial applications. The low-stress molding compound of the package also assures greater thermal range without the need for external heat sinking.

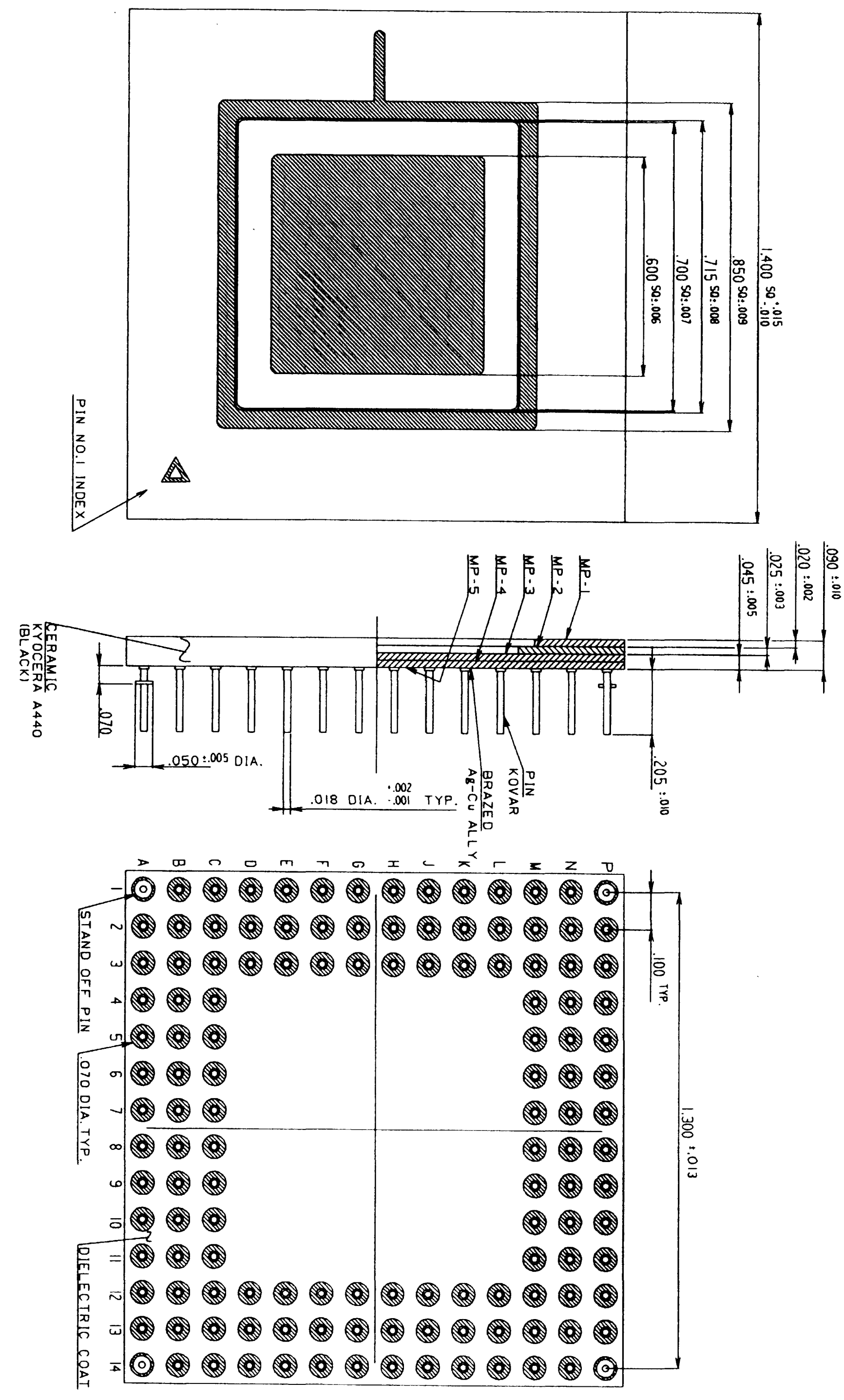

PQFP
Plastic Quad Flat Pack
(EIAJ standard, 31.5/25.6 mil pitch)

Leaded Chip Carrier Types

The LCC types are available in 84-lead plastic packages. The LCC type is a JEDEC standard, J-leaded package with a 0.050 pitch intended for surface-mount commercial applications. In such applications, this package can be used with most pick-and-place SMT robots on conventional boards with reflow soldering, and can also be mounted readily on both sides of the board for increased density.

The package is also available with an internal heat spreader to assure greater thermal range without the need for external heat sinking or fixtures.


PLCC
Plastic Leaded Chip Carrier
(JEDEC Standard, 50mil pitch)



1.400 SQ +.015 -.010
.850 SQ ±.009
.715 SQ ±.008
.700 SQ ±.007
.600 SQ ±.006
PIN NO.1 INDEX
.090 ±.010
.020 ±.002
.025 ±.003
.045 ±.005
MP-1
MP-2
MP-3
MP-4
MP-5
CERAMIC KYOCERA A440 (BLACK)
.070
.050 ±.005 DIA.
.018 DIA. +.002 -.001 TYP.
PIN KOVAR
BRAZED Ag-Cu ALLY
.205 ±.010
A B C D E F G H J K L M N P
1 2 3 4 5 6 7 8 9 10 11 12 13 14
.100 TYP.
1.300 ±.013
STAND OFF PIN
.070 DIA. TYP.
DIELECTRIC COAT

Ceramic Pin Grid Array Materials and Specifications

Standard:

- 99% Alumina co-fired ceramic
- Spot gold
- Low-stress die coat
- Gold-plated pin finish

Options (at additional cost):

- Imbedded heat slug

Shipping Containers

Depending upon quantity and lead count, all STAR Semiconductor packages are shipped in either trays or ESD-protected bags and foam.

Trays and marking are included in all price quotes. All trays are ESD-protected and desiccant packed.

Sockets and Insertion Tools

Below is a partial list of manufacturers known to offer sockets for STAR Semiconductor PGA package types:

AMP, Inc.	3M Textool	Precicontact, Inc.
Harrisburg, PA 17105	Austin, TX	835 Wheeler Way
(717) 564-0100	(800) 328-7731	Langhorne, PA 19047
		(215) 757-1202

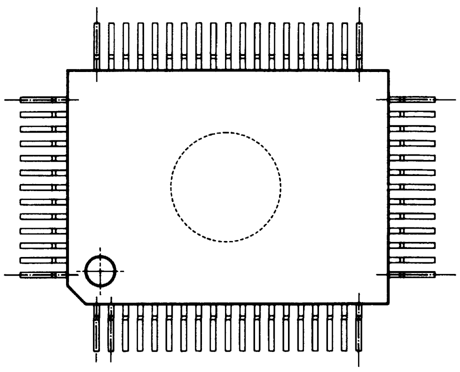
PGA insertion and extraction tools are offered by:

Micro Electronics Systems Corp.
New Milford, CT
(203) 350-5004

PGA insertion tool part # P/N 166
PGA extraction tool part # P/N 266

This listing does not imply an endorsement by STAR Semiconductor. Each user must evaluate the particular socket/tool type.

Plastic Quad Flat Pack



Materials and Specifications

Standard:

- Copper alloy or alloy 42 lead frame
- Spot silver
- Low-stress molding compound
- Solder-plated lead finish

Electrical Characterization:

Lead Count	Inductance (nH) shortest-longest	Capacitance (pF) shortest-longest	Resistance shortest-longest
100	5.3 - 10.8	0.5 - 0.8	210-260
160	15.1 - 20.5	0.6 - 0.8	270-280

Notes:

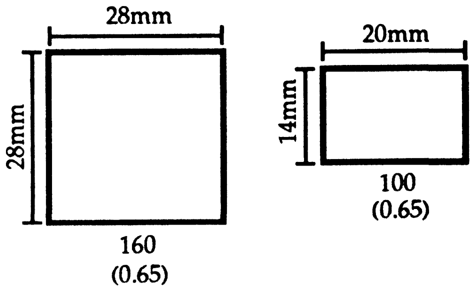
A. The inductance value is the measured self-inductance of a single package lead without mutual coupling effects
B. The bondwires are included in the measured inductance values.
C. The bondwires are included in the measured resistance values.
D. The capacitance value is the measured mutual capacitance between two adjacent leads and includes bondwires and pads.
E. The capacitance measurement is for the die-free package and does not include bond wires.

EIAJ PQFP BODY SIZES

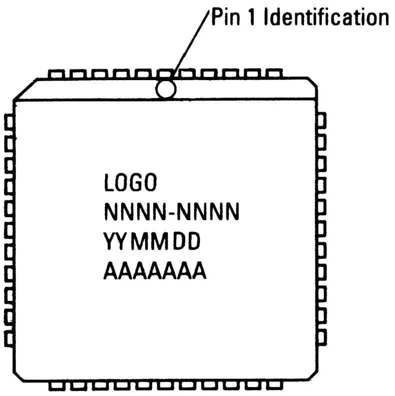
Shipping Trays

PIN COUNT	QTY
100	30, 15
160	30, 15

Trays and marking are included in all price quotes. All trays are EDS-protected and desiccant packed.



Leaded Chip Carrier



PLCC Materials and Specifications

Standard:

- Stamped lead frame
- Olin 151 copper alloy
- Spot silver at wirebonds
- Solder-lead finish

Electrical Characterization:

Lead Count	Inductance (nH) shortest-longest	Capacitance (pF) shortest-longest	Resistance shortest-longest
84	7.6 - 11.8	1.4 - 1.5	93 - 100

Notes:

A. The range of values results from variations in trace length from the shortest to the longest leads.
B. The inductance value is the measured self-inductance of a single package lead without mutual coupling effects
C. The bondwires are included in the measured inductance values.
D. The bondwires are included in the measured resistance values.
E. The capacitance value is the measured mutual capacitance between two adjacent leads and includes bondwires and pads.

Operating and Handling Considerations

This section summarizes important operating recommendations and precautions which should be followed in the interest of maintaining the high standards of performance of solid state devices.

The ratings included in the data bulletins are based on the Absolute Maximum Rating System, which is defined by the following industry Standard (JEDEC) statement:

> Absolute-Maximum Ratings are limiting values of operating and environmental conditions applicable to any electron device of a specified type as defined by its published data, and should not be exceeded under the worst probable condition.
>
> The device manufacturer chooses these values to provide acceptable serviceability of the device, taking no responsibility for equipment variations, environmental variations, and the effects of changes in operating conditions due to variations in device characteristics.

The equipment manufacturer should design so that initially and throughout life no absolute-maximum value for the intended service is exceeded with any device under the worst probable operating conditions with respect to supply voltage variation, equipment component variation, equipment control adjustment, load variation, signal variation, environmental conditions, and variations in device characteristics.

It is recommended that equipment manufacturers consult STAR Semiconductor whenever device applications involve unusual electrical, mechanical, or environmental operating conditions.

General Considerations

The design flexibility provided by these devices makes possible their use in a broad range of applications and under many different operating conditions. When incorporating these devices in equipment, therefore, designers should anticipate the rare possibility of device failure and make certain that no safety hazard would result from such an occurrence.

The small size of most solid state products provides obvious advantages to the designers of electronic equipment. However, it should be recognized that these compact devices usually provide only relatively small insulation area between adjacent leads. When these devices are used in moist or contaminated atmospheres, therefore, supplemental protection must be provided to prevent the development of electrical conductive paths across the relatively small insulating surfaces.

Devices should not be connected into or disconnected from circuits with the power on because high transient voltages may cause permanent damage to the devices.

Testing Precautions

In common with many electronic components, solid-state devices should be operated and tested in circuits which have reasonable values of current limiting resistance, or other forms of effective current overload protection. Failure to observe these precautions can cause excessive internal heating of the device resulting in destruction and/or possible shattering of the enclosure.

Mounting

Integrated circuits are normally supplied with lead-tin plated leads to facilitate soldering into circuit boards. In those applications requiring welding of the device leads, rather than soldering, the devices may be obtained with gold or nickel plated Kovar* leads**. It should be recognized that this type of plating will not provide complete protection against lead corrosion in the presence of high humidity and mechanical stress.

*Trade Name: Westinghouse Corp.

**MIL-M-38510A, paragraph 3.5.6.1(a), lead material

The aluminum-foil-lined cardboard "sandwich pack" employed for static protection of the flat-pack also provides some additional protection against lead corrosion, and it is recommended that the devices be stored in this package until used.

When integrated circuits are welded onto printed circuit boards or equipment, the presence of moisture between the closely spaced terminals can result in conductive paths that may impair device performance in high-impedance applications. It is therefore recommended that conformal coatings or potting be provided as an added measure of protection against moisture penetration.

In any method of mounting integrated circuits which involves bending or forming of the device leads, it is extremely important that the lead be supported and clamped between the bend and the package seal, and that bending be done with care to avoid damage to lead plating. In no case should the radius of the bend be less than the diameter of the lead, or in the case of rectangular leads, less than the lead thickness. It is also extremely important that the ends of the bent leads be straight to assure proper insertion through the holes in the printed-circuit board.

Handling

All CMOS gate inputs have a gate protection network. All outputs have diode protection provided by inherent p-n junction diodes. These protective elements at input and output interfaces protect CMOS devices from gate-oxide failure in handling environments where static discharge is not excessive. In low-temperature, low-humidity environments, improper handling may result in device damage.

Unused Inputs

All unused input leads must be connected to either VSS or VDD, whichever is appropriate for the logic circuit involved. A floating input on a high-current type not only can result in faulty logic operation, but can cause the maximum allowable power dissipation to be exceeded and may result in damage to the device. Inputs to these types, which are mounted on printed-circuit boards that may temporarily become unterminated, should have a pull-up resistor to VSS or VDD. A useful range of values for such resistors is from 10 kilohms to 1 megohm.

Input Signals

Signals shall not be applied to the inputs while the device power supply is off unless the input current is limited to a steady state value of less than 10 milliamperes. Input currents of less than 10 milliamperes prevent device damage; however, proper operation may be impaired as a result of current flow through the structural diode junction.

Output Short Circuits

Shorting of outputs to VSS or VDD can damage many of the higher-output-current CMOS types. In general, these types can all be safely shorted for supplies up to 5 volts, but will be damaged (depending on type) at higher power-supply voltages. For cases in which a short-circuit load, such as the base of a p-n-p or an n-p-n transistor, is directly driven, the device output characteristics given in the published data should be consulted to determine the requirements for safe operation.

The SPROC signal processing solution. Performance without compromise.
Star Semiconductor
The SPROC™ Signal Processor Solution
STAR SEMICONDUCTOR
SPROC 1400-4
BUSINESS REPLY MAIL

Section 7
Supplementary Information

Catalogs and Brochures

STAR Semiconductor provides a series of introductory literature describing the company mission, SPROC technology, the SPROClab development environment and applications suggestions. This information can be obtained by contacting the STAR sales office in your area or by calling STAR Semiconductor at

(908) 647-9400

Domestic Sales Offices:

ALABAMA

Novus Group, Inc.
2905 Westcorp Blvd.
Suite 120
Huntsville, AL 35805
(205) 534-0044
FAX: (205)534-0186

CALIFORNIA

Norcomp
3350 Scott Blvd.
Suite 24
Santa Clara, CA 95054
(408) 727-7707
FAX: (408 986-1947

S.C. Cubed
17862 17th Str.
Suite 207
Tustin, CA 92680
(714) 731-9206
FAX: (714) 731-7801

S.C. Cubed
68 Long Court
Suite 2C
Thousand Oaks, CA 91360
(805) 496-7307
FAX: (805) 495-3601

S.C. Cubed
5060 Shoreham Pl.
Suite 200
San Diego, CA 92122
(619) 458-5808
FAX: (619) 458-5823

FLORIDA

Semtronics Assoc., Inc.
657 Maintland Ave.
Altamonte Springs, FL 32701
(407) 831-8233
FAX: (407)831-2844

Semtronics Assoc., Inc.
1467 S. Missouri Ave.
Clearwater, FL 34616
(813) 461-4675

Semtronics Assoc., Inc.
3741 NW 55th St.
Ft. Lauderdale, FL
33309
(305) 731-2484

GEORGIA

Novus Group, Inc.
6115-A Oakbrook Pkwy
Norcross, GA 30093
(404) 263-0320
FAX: (404) 263-8946

ILLINOIS

Beta Technology, Inc.
1009 Hawthorn Dr.
Itasca, IL 60143
(708) 250-9586
FAX: (708) 250-9592

MARYLAND

Micro Comp, Inc.
1421 S. Canton Ave.
Baltimore, MD 21227
(301) 644-5700
FAX: (301) 644-5707

MASSACHUSETTS

STAR Semiconductor Corp.
6A Damon Mill Square
Concord, MA 01742
(508) 371-9240

Mill Bern Assoc.
2 Mack Rd.
Woburn, MA 01801
(617) 932-3311
FAX: (617) 932-0511

NEW YORK

Parallax
734 Walt Whitman Rd.
Melville NY 11747
(516) 351-1000
FAX: (516) 351-1606

NORTH CAROLINA

Novus Group, Inc.
102L Commonwealth Ct
Cary, NC 27511
(919) 460-7771
FAX: (919) 460-5703

OHIO

Bear Marketing, Inc.
240 W. Elmwood, Ste. 1012
Dayton, OH 45459
(513) 436-2061
FAX: (513) 436-9137

Bear Marketing, Inc.
P.O. Box 427
3554 Brecksville Rd.
Richfield, OH 44286-0427
(216) 659-3131
FAX: (216) 659-4823

OKLAHOMA

Orion Assoc.
7966 E. 41st, Ste. 7E
Tulsa, OK 74145
(918) 665-3562
FAX : (918) 665-3585

OREGON

Thorson Company NW
9600 S.W. Oak St.
Suite 320
Portland, OR 97223-6586
(503) 293-9001
FAX: (503) 293-9007

PENNSYLVANIA

Bear Marketing, Inc.
4284 Route 8, Ste. 211
Allison Park, PA 15101
(412) 492-1150
FAX: (412) 492-1155

Delta Technical Sales, Inc
122 N. York Rd., Ste. 9
Hatboro, PA 19040
(215) 957-0600
FAX: (215) 957-0920

TEXAS

Orion Assoc.
12000 Ford Rd.
Suite 200
Dallas, TX 75234
(214) 241-3505
FAX: (214) 241-3503

Orion Assoc.
9430 Research Blvd.
Echelon IV, Ste. 400
Austin, TX 78759-6535
(512) 343-4532
FAX: (512) 343-4534

Orion Assoc.
4800 Sugar Grove Blvd.
Suite 290
Stafford, TX 77477
(713) 240-6767
FAX: (713) 240-1329

WASHINGTON

Thorson Company NW
12340 NE 8th St.
Bellevue, WA 98005
(206) 455-9180
FAX: (206) 455-9185

WISCONSIN

Beta Technology, Inc.
9401 W. Beloit Rd.
Suite 409
Milwaukee, WI 53227
(414) 543-6609
FAX: (414) 543-9288

CANADA

Electro Source, Inc.
230 Galaxy Blvd.
Rexdale, Ontario
M9W 5R8
(416) 675-4490
FAX: (416) 675-6871

Electro Source, Inc.
Suite 420
Pointe Claire, Quebec
H9R 4S2
(514) 630-7486
FAX: (514) 630-7421

Electro Source, Inc.
3665 Kingsway
Suite 300
Van Couver, B.C.
V5R 5W2
(604) 435-8066
FAX: (604) 435-8181

Electro Source, Inc.
340 March Rd.
Suite 503
Kanata, Ontario
K2K 2E4
(613) 592-3214
FAX: (613) 592-4256

International Sales Offices:

ENGLAND

Micro Call LTD
17 Thame Park Rd.
Thame, Oxfordshire
OX9 3XD
England
(011) 44-844-261919
FAX: (011) 44-844-261683

FRANCE

REPTRONICsa
1 bis, rue Marcel Paul
Batement A
Z.I. de la Bonde 91300
Massey, France
(011) 33-1-601 39300
FAX: (011) 33-1-601 39118

GERMANY

Metronik
Leonhardsweg 2
8025 Unterhaching
Germany
(011) 49-89 6110848
FAX: (011) 49-89 6112246
FAX: (011) 49-89 6116468

HONG KONG

EXCEL ASSOC., LTD.
Unit No. 2520-2525
Tower 1
Metroplaza, Hing Fong Rd
Kwai Fong, N.T.
(011) 852-418-0909
FAX: (011) 852-418 1600

IRELAND

Memec (Ireland) LTD.
Innovation Centre
Enterprise Hse.
Plassey Technological Park
Limerick, Ireland
(011) 353-61 330742-5
FAX: (011) 353-61 331888

ISRAEL

Aviv Electronics
5 Har'arad St.
P.O. Box 24190
Tel-Aviv 61241
Israel
(011) 972-3-544 7202
FAX: (011) 972-3-544 7650

Distributed by:

Marshall Industries locations throughout the United States and Canada.

Corporate Headquarters - 9320 Telstar Ave.
El Monte, CA 91731
(818) 307-6000

WESTERN REGION

Los Angeles	Denver	Salt Lake City
26637 W. Agoura Rd. Calabasas, CA 91302 (818) 878-7000	12351 North Grant Thornton, CO 80241 (303) 451-8383	2355 S. 1070 West, Ste. D Salt Lake City, UT 84119 (801) 973-2288

SOUTHWEST REGION

Irvine	San Diego	Phoenix
One Morgan Irvine, CA 92718 (714) 458-5301	10105 Carroll Canyon Rd. San Diego, CA 92131 (619) 578-9600	9831 S. 51st St. Ste. C107-109 Phoenix, AZ 85044 (602) 496-0290

NORTHWEST REGION

Sacramento	Seattle	San Francisco
3039 Kilgore Ave., #140 Rancho Cordova, CA 95670 (916) 635-9700	11715 N. Creek Pkwy. S. Ste. 112 Bothell, WA 98011 (206) 486-5747	336 Los Coches St. Milpitas, CA 95035 (408) 942-4600
Portland		
9705 S.W. Gemini Dr. Beaverton, OR 97005 (503) 644-5050		

CENTRAL REGION

Dayton	Cleveland	Pittsburgh
3520 Park Center Dr. Dayton, OH 45414 (513) 898-4480	30700 Bainbridge Rd., Unit A Solon, OH 44139 (216) 248-1788	401 Parkway View Dr. Pittsburgh, PA 15205 (412) 788-0441

Kansas City

10413 W. 84th Terrace
Pine Ridge Business Park
Lenexa, KS 66214
(913) 492-3121

St. Louis

3377 Bollenberg Dr.
Bridgeton, MO 63044
(314) 291-4650

MIDWEST REGION

Minneapolis

3955 Annapolis Lane
Plymouth, MN 55447
(612) 559-2211

Chicago

50 E. Commerce Dr., Unit 1
Schaumburg, IL 60173
(708) 490-0155

Milwaukee

Crossroads Corporate Ctr. 1
20900 Swenson Dr.,
Ste. 150
Waukesha, WI 53186
(414) 797-8400

Indianapolis

6990 Corporate Dr.
Indianapolis, IN 46278
(317) 297-0483

Michigan

31067 Schoolcraft
Livonia, MI 48150
(313) 525-5820

TEXAS REGION

Dallas

2045 Chenault St.
Carrollton, TX 75006
(214) 233-5200

Houston

7250 Langtry
Houston, TX 77040
(713) 895-9200

Austin

8504 Cross Park Dr.
Austin, TX 78754
(512) 837-1991

FLORIDA REGION

Orlando

380 S. Northlake Blvd.
Ste. 1024
Altamonte Springs, FL 32701
(407) 767- 8585

Ft. Lauderdale

2700 W. Cypress Creek Rd.
Ste. D114
Ft. Lauderdale, FL 33309
(305) 977-4880

Tampa

2840 Scherer Dr., Ste. 410
St. Petersburg, FL 33716
(813) 573-1399

EASTERN REGION

Boston

33 Upton Dr.
Wilmington, MA 01887
(508) 658-0810

Connecticut

20 Sterling Dr.
Barnes Industrial Park, N.
P.O. Box 200
Wallingford, CT 06492-0200
(203) 265-3822

Binghamton

100 Marshall Dr.
Endicott, NY 13760
(607) 785-2345

Rochester	Toronto	Montreal
1250 Scottsville Rd. Rochester, NY 14624 (716) 235-7620	4 Paget Rd., Units 10 & 11 Building 1112 Brampton, Ontario L67 5G3 (416) 458-8046	148 Brunswick Blvd. Pointe Claire, Quebec H9R 5P9 (514) 694-8142

MID-ATLANTIC REGION

Long Island	Philadelphia	Maryland
95 Oser Ave. Hauppage, LI, NY 11788 (516) 273-2053	158 Gaither Dr. Mt. Laurel, NJ 08054 (609) 234-9100	2221 Broadbirch Dr. Silver Spring, MD 20904 (301) 622-1118
North New Jersey		
101 Fairfield Rd. Fairfield, NJ 07006 (201) 882-0320		

SOUTHEASTERN REGION

Raleigh	Atlanta	Huntsville
5224 Greens Dairy Rd. Raleigh, NC 27604 (919) 878-9882	5300 Oakbrook Parkway, Ste. 140 Norcross, GA 30093-9990 (404) 923-5750	3313 Memorial Parkway S. Huntsville, AL 35801 (205) 881-9235

SOFTWARE LICENSE AGREEMENT

BEFORE OPENING THIS PACKAGE, you should carefully read all the terms and conditions of the following software license agreement. Opening this package constitutes your acceptance of these terms and conditions. If you do not agree to these terms and conditions, you should promptly return the unopened package to STAR Semiconductor Corp. ("STAR"), and your money will be refunded.

License Grant

In consideration of payment of the applicable license fee, STAR hereby grants to you a non-exclusive, non-transferable license (a) to use the enclosed software provided by STAR in machine-readable, object-code form and the related documentation (the "Software") on a single CPU and solely for your own internal use and (b) to use the programmation files generated by the Software only on DSP chips supplied by STAR or its authorized representatives or dealers. You may not reverse engineer, decompile, disassemble or modify, or make any copies of, the Software. The rights and license granted hereunder are restricted solely and exclusively to you and may not in any way, directly or indirectly, be licensed, assigned, sublicensed, leased or otherwise transferred by you without the prior written consent of STAR. STAR may terminate this license by written notice to you in the event you fail to comply with the terms of this Agreement.

Ownership

Ownership of the Software, and of any copyright, patent, trade secret or other intellectual and proprietary rights therein, are and remain in the future solely and exclusively in STAR and/or its licensors.

Limited Warranty

STAR warrants that the diskette(s) or other media on which the Software is furnished to be free from defects in material and workmanship under normal use for a period of ninety (90) days from date of shipment to you. Subject to the foregoing, the Software is licensed "AS IS" without any warranty or representation, and without any support or right to any corrections, bug fixes, maintenance, modifications, enhancements, improvements or extensions, now or in the future. STAR does not warrant or represent that the Software will meet your requirements or that the operation of the Software will be uninterrupted or error free. Should the Software prove defective, STAR has no obligation or responsibility whatsoever. Nevertheless, in the event that STAR does provide assistance to any user of the Software, all of the limitations with respect to warranties and remedies shall apply to any assistance so rendered. **The warranty in the first sentence of this paragraph is in lieu of all other warranties, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.**

Limitation of Liability

In no event shall STAR be liable to you or anyone else for any liability, loss or damage including, without limitation, indirect, incidental, special, punitive or consequential damages of any kind, or loss of use or other economic loss, even if STAR has been advised of the possibility of such damages. You agree that you will bear the entire risk of using the software and that you will indemnify, defend and hold STAR harmless against any claims arising out of your use of the software. Notwithstanding the foregoing, in the event STAR is determined to be liable for damages with respect to the software, in no event shall the amount of damages exceed the amount of the license fee paid therefor.

STAR Terms and Conditions

Your licensing of the Software from STAR, and your purchases of any products from STAR, shall be governed by STAR's standard terms and conditions.

General

This Agreement may not be modified or waived, in whole or in part, except in writing, executed by authorized representatives of both parties. This Agreement shall be governed by the internal, domestic laws of the State of New Jersey and shall inure to the benefit of STAR, its successors and assigns. This Agreement is the sole and exclusive statement of the Agreement between us which supersedes any proposal or prior agreement or understanding, oral or written, and any other communications relating to the subject matter of this Agreement.

STAR Semiconductor Corp.
25 Independence Boulevard
Warren, NJ 07059

Index

Keith Allsop	Keith Allsop	[illegible]	JIm Magos
[illegible]	Scott Andrews	[illegible]	Terry Montlick
[illegible]	Chafik Behidj	Chester M. Nowicki	Chester Nowicki
[illegible]	Ken Brizel	[illegible]	Jack O'Donnell
[illegible]	Jonathan Chandross	Paul J Pedevillano	Paul Pedevillano
Karen Champigny	Karen Champigny	[illegible]	Sam Phillips
[illegible]	Tony Curran	[illegible]	Dick Randlett
[illegible]	Michael D'Agostino	[illegible]	Jeff Robinson
[illegible]	Kathy Geene	[illegible]	Keith Rouse
[illegible]	Michael Gillis	Jack Salata	Jack Salata
[illegible]	Ivan Godard	David C Scott	David Scott
[illegible]	Dan Greenwood	STEPHEN G SCOTT	Steve Scott
[illegible]	Paul Jordan	[illegible]	Frank Sgammato
[illegible]	Hesh Khajezadeh	[illegible]	Steve Sheslow
[illegible]	Andy Krassowski	[illegible]	Terri Swartz
[illegible]	JIm Lake	Kevin Watts	Kevin Watts
[illegible]	Ed Lee	[illegible]	Bernie Witkosky

Retail $4.95

Star Semiconductor Corp.
25 Independence Boulevard
Warren, NJ 07059

908.647.9400

DB1000-1/December, 1991

Printed in the USA