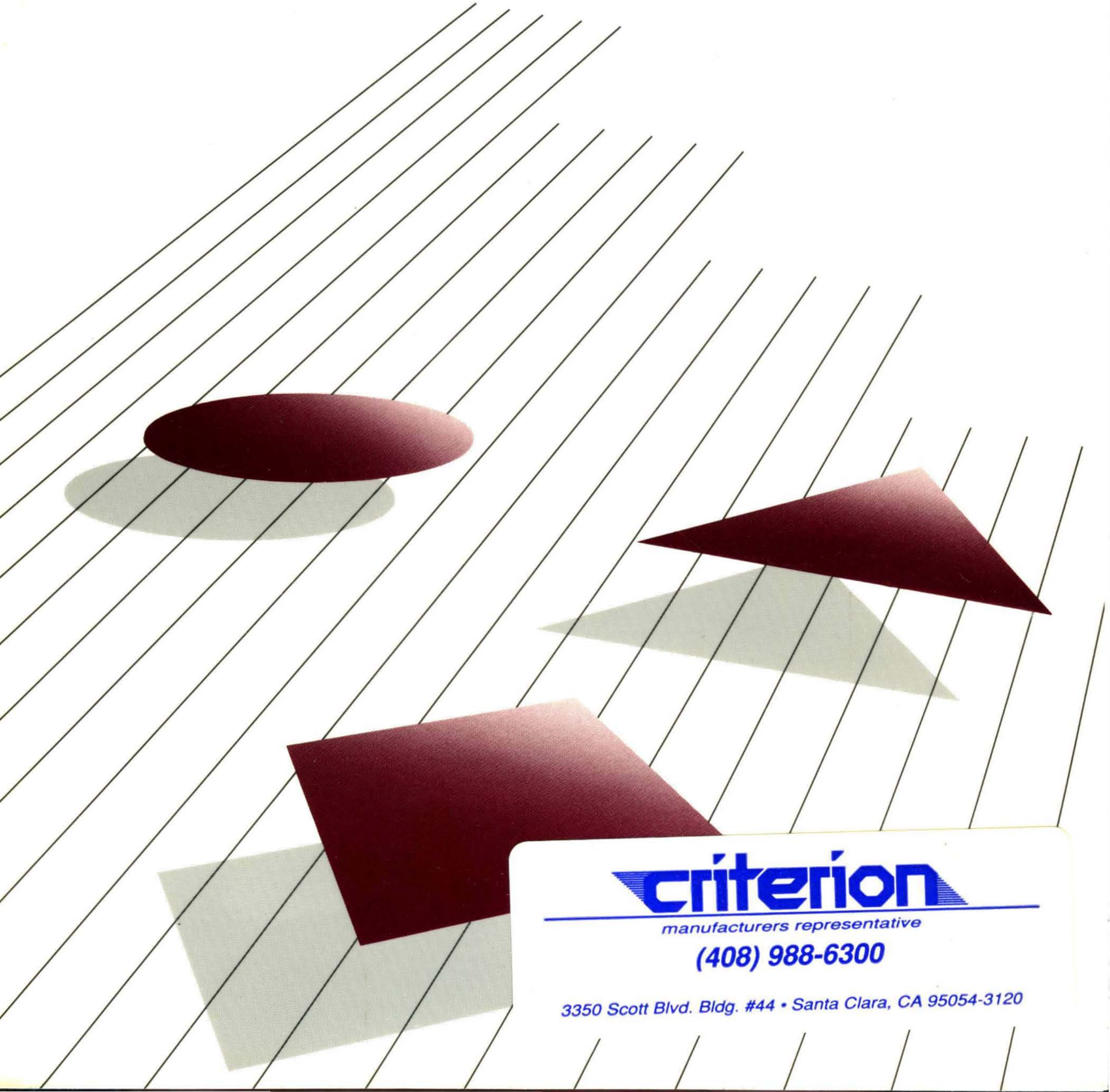
WAFERSCALE INTEGRATION, INC.

Programmable System™ Devices

PSD

Design and Applications Handbook

1990



WAFERSCALE INTEGRATION, INC.

Programmable System™ Devices (PSD) Design and Applications Handbook

1990

47280 Kato Road, Fremont, California 94538
415-656-5400 Facsimile: 415-657-5916 Telex: 289255

Printed in U.S.A.

WAFERSCALE INTEGRATION, INC.

Section Index

General Information

For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.

WAFERSCALE INTEGRATION, INC.

Table of Contents

General Information

PSD Product Specifications

PSD Development Systems

PSD Applications

PSD Applications (Cont.)

Article Reprint

Package Information

Sales Representatives and Distributors

Introduction to Programmable System™ Devices (PSD)

Programmable System Devices, or PSDs, are user-configurable system level building blocks on-a-chip enabling quick implementation of application specific controllers and peripherals.

WSI PSDs are ideal for designers who require fast time-to-market, low risk, greater system integration and lower power consumption. PSDs enable designers to configure their microcontroller/peripheral to meet exact design requirements. WSI's PSDs are unique in that they are the only VLSI devices available today that provides a *user-configurable off-the-shelf solution at the system level.*

The user-configurability of PSDs enables them to be used in many different applications, including:

□ Computers (Workstations and PCs) — Fixed Disk Control, Modem, Imaging, Laser Printer Control

□ Telecommunications — Modem, Cellular Phone, Digital PBX, Digital Speech, FAX, Digital Signal Processing

□ Industrial — Robotics, Power Line Access, Power Line Monitor

□ Medical Instrumentation — Hearing Aids, Monitoring Equipment, Diagnostic Tools

□ Military — Missile Guidance, Radar, Sonar, Secure Communications, RF Modems

PSDs are available in a variety of space saving surface mount and through-hole package configurations for commercial, industrial, and military applications. WSI offers windowed package options for prototyping and low cost OTP (one-time programmable) packages for high volume applications. PSDs utilize WSI's proprietary split-gate CMOS EPROM technology for low power consumption.

There are currently four PSD family devices in production. These include the PAC1000, MAP168, PSD301, and SAM448.

□ The PAC1000 is a user-configurable microcontroller. It may be used as a stand-alone microcontroller or as a peripheral to microprocessors. It is ideal for embedded control applications, including graphics, local area network, and disk drive control in both military and commercial applications.

□ The MAP168 is a user-configurable peripheral. It is used in DSP applications including modems, motor control and medical instrumentation. The MAP168 is ideal for DSP based applications where fast time-to-market, small form factor and low power consumption are essential. When combined together in an 8- or 16-bit system, virtually any DSP chip (TMS320 series, etc.) and the MAP168 work together to create a very powerful 2-piece chip-set. This combination provides essentially all of the required control and peripheral element of a DSP system.

□ The PSD301 is a user-configurable peripheral for microcontroller applications including disk drives, low cost modems, and mobile phones. The PSD301 is ideal for microcontroller based applications where fast time-to-market, small form factor and low power consumption are essential. When combined together in an 8- or 16-bit system, virtually any microcontroller (8051, 8096, 16000, etc.) and the PSD301 work together to create a very powerful 2-piece chip-set. This implementation provides the required control and peripheral element of a microcontroller based system peripheral with no external "glue" logic required.

□ The SAM448 is a user-configurable sequencer for state machine and bus interface applications. Its flexible I/O and architecture make it ideal for use in interfacing to both existing bus architectures (AT, VME, MCA-bus), and evolving bus standards (EISA, NuBUS).

Application specific features can be easily programmed into the PSD EPROM array for quick design implementation. Unlike the current generation of programmable gate arrays, which require the use of unpredictable, and often time unavailable routing resources, all PSD logic is fully connected internally. This means that all timing is predictable ahead of design implementation, and routing is assured. This greatly simplifies and reduces the design implementation and simulation process, and provides designers with a significantly more reliable, lower risk path to market. WSI PSDs also eliminate the NRE, turn-around-time, and risks associated with gate arrays and other ASIC solutions.

As product life cycles continue to shrink, designers can win the race from idea to marketable product with WSI PSDs. PSDs are quickly configured and programmed by the designer by using low cost, easy-to-use WSI PC-based development tools. The user-friendly menu-driven software includes high level design entry, simulation and programming packages for rapid system development.

WSI supports its PSD product family with an applications hotline and bulletin board, as well as highly trained, technical Field Applications Engineers. As standard products, WSI PSDs are available from WSI's franchised world-wide distribution network.

WAFERSCALE INTEGRATION, INC.

Company Profile

Introduction

WaferScale Integration, Inc. (WSI) designs and produces the world's broadest and fastest families of CMOS PROMs, RPROMs, EPROMs, and Programmable System™ Devices (PSD). These product families target the needs of system designers who must reduce system development time and deliver market competitive products in continuously shorter periods of time. WSI's programmable VLSI products additionally enable higher system performance from smaller, more compact end products due to higher levels of system integration at the chip level.

WSI's mission is clear — to build a great company by serving its customers with a portfolio of high-performance programmable VLSI products that enable designers to achieve faster time to market with new, advanced electronic systems.

The company's patented self-aligned, split-gate EPROM technology forms the core of WSI's programmable products and delivers higher performance and greater density than competing "stacked gate" EPROM technologies. This core technology has enabled WSI to be first in the industry with numerous breakthroughs in speed, density, process and packaging. WSI has leveraged this technology into the broadest family of CMOS PROMs, RPROMs, and EPROMs available.

WSI's new "off the shelf" user-configurable PSDs provide system level building blocks on a single chip that enable quick implementation of application specific controllers and peripherals. They are the first to integrate high-performance EPROM, SRAM and logic and deliver a performance and integration breakthrough to the programmable products market. PSDs are user-configurable on a PC or compatible and can be tailored for use in a variety of system applications. As a result, WSI has established itself as a leading supplier of high-performance programmable VLSI solutions to a broad customer base that includes some of the world's largest and most technologically advanced electronics companies.

Founded in 1983, WSI is headquartered in a 66,000 square foot facility in Fremont, California and has more than 125 employees. Through a long-term equity, manufacturing and technology license agreement with Sharp Corporation of Japan, WSI produces its products in a world-class production facility that guarantees the highest quality at competitive costs.

Markets and Applications

WSI's high-performance non-volatile memory and PSD products are used by the world's leading suppliers of high-performance electronic systems in communications, data processing, military and industrial markets. Customer end products cover a broad spectrum and typically include cellular telephones, workstations, DSP computers, navigation controllers, T1 multiplexers, modems, image processors, missiles, LAN controllers, high density disk drives and the like.

Customer applications include image processing, digital signal processing, bus control, LAN data and file control, real time process control, graphics processing, hard disk control, flight simulators, DMA control, and others. WSI products are ideally suited for these applications where designers are faced with increasingly shorter product life cycles and must develop new, competitive high-performance products in short periods of time.

Products

Memory Products

EPROMs

WSI offers the broadest line of CMOS EPROM products available featuring architectures ranging from 8K x 8 to 128K x 8, plus several x16 products, with speeds ranging from 40 to 200 ns. Commercial, industrial and MIL-STD-883C/SMD products are available. A wide variety of package selections are available including plastic and hermetic, through-hole and surface mount types.

"L" Family

WSI's "L" family memory products are the industry's fastest, low power JEDEC pinout EPROMs and meet the requirements of many mainstream system applications. With speeds ranging from 90 to 200 ns and architectures from 8K x 8 to 128K x 8 including several x16 products, "L" family EPROMs are ideal for high-performance personal computers and workstations. Taking advantage of its split-gate EPROM technology, WSI uses a conservative 1.2 micron lithography to achieve world-class memory densities that traditionally require lower yielding sub-micron technologies.

"F" Family

The "F" family is WSI's fastest line of EPROMs, featuring speeds ranging from 40 to 110 ns and architectures from 8K x 8 to 32K x 8, plus several x16 products. The high speed and word width options of the "F" family EPROMs make them attractive for use in high-end engineering and scientific workstations, data communications and other high-performance applications.

RPROMs

RPROMs provide bipolar PROM pin-out with matching speed and CMOS low power operation. The RPROM (Re-Programmable Read Only Memory) product series includes architectures ranging from 2K x 8 to 32K x 8 with speeds ranging from 25 to 70 ns.

Commercial, industrial and MIL-STD-883C/SMD configurations are available in a variety of hermetic and plastic package styles.

Programmable System™ Devices (PSDs)

WSI's family of Programmable System Devices (PSDs) represent a new class of programmable VLSI products, achieving unparalleled levels of performance, configurability and integration. Offering a significantly higher level of integration over programmable logic, PSDs are the first programmable VLSI products to integrate high-speed EPROM, SRAM and logic on a single chip thereby providing complete system solutions to the design engineer. PSDs are off-the-shelf system building elements that can be quickly configured and programmed for a variety of system applications thus enabling system designers to shorten system development time.

The PSD is a new solution for system designers who build high-end systems around embedded controllers and advanced microprocessors. These new systems require faster, more highly integrated and lower cost VLSI solutions as well as rapid design cycles. WSI's new PSD family meets this demanding set of needs.

The initial members of WSI's PSD family includes:

- ❑ The PAC1000 User-Configurable Microcontroller
- ❑ The MAP168 User-Configurable Peripheral with Memory
- ❑ The PSD301 User-Configurable Peripheral with Memory
- ❑ The SAM448 User-Configurable Microsequencer

Products (Cont.)

Design Tools and Support

WSI's development tools minimize the time required for designers to program PSDs for use in a variety of system applications. PSDs are supported with complete easy-to-use program development, simulation and programming software, the PC hosted MagicPro™ Memory and PSD Programmer, a dial-in applications bulletin board and WSI's team of factory and field applications engineers. As a result, WSI customers achieve their goal of shorter system development time and reach new markets sooner.

Custom Circuits

To serve the needs of its customers with unique requirements, WSI offers its custom circuit capability using its cell based library of EPROM, static RAM and logic functions. Standard products described in this catalog can usually be modified on a custom basis to serve particular requirements. New customer defined custom products that incorporate high-performance non-volatile memory, SRAM and logic can be produced that deliver significant speed or system integration advantages. Contact your local WSI sales office for additional information.

Manufacturing

A key ingredient for success in leading-edge semiconductors is a world-class fabrication facility that ensures high volume capacity and prompt delivery of highly reliable and high yielding VLSI circuits. To this end, WSI has licensed its proprietary CMOS EPROM and logic process technology to Sharp Corporation of Osaka, Japan. This long term alliance ensures high quality, high-volume production, competitive costs and fast delivery. The Sharp facility in Fukuyama, Japan employs the most advanced sub-micron VLSI integrated circuit manufacturing equipment available including ion implantation, reactive ion etch, and wafer stepper lithographic systems.

Quality and Reliability

WSI is deeply committed to product excellence. This begins with proper management attitude and direction and with this focus, the Quality and Reliability Program is able to operate efficiently. As a result, product quality becomes part of each employee's responsibility.

Quality and Reliability begin with the proper product and process designs and is supported by material and process controls. Examples are products manufactured on an epitaxial silicon layer to reduce latch-up sensitivity, all pins are designed to withstand >2,000 volts ESDS, numerous ground taps are used which increases product noise immunity, metal traces are designed to carry a current density of $>2.0 \times 10^5$ amps/cm^2, top passivation extends over into the scribe lane to seal the die edges, data retention is performed 100% on re-programmable products (T_A = +225°C, T = 72 hours), automated die attach and bonding is used extensively, wafers are fabricated in a Class 10 clean room, raw materials, chemicals and gases are inspected before use, and statistical controls are used to keep the process on course.

Product and process introductions or changes are routinely evaluated for worthiness. Life tests are conducted at higher than typical stress levels (T_A = +150°C, V_{CC} = +6.5V) and even at these stress levels, WSI products have demonstrated low failure rates (see the Quality and Reliability section in the WSI 1990 databook).

WSI is active in Military programs and its Quality and Reliability System supports Compliant Non-Jan products. WSI also supports DESC's (Defense Electronics Supply Center) Standardized Military Drawings (SMD) program. As of October, 1989, WSI has eighteen products on SMDs with additional products pending. Several additional products not on SMDs are available per MIL-STD-883C. See Section 7 (Military Products) in the WSI 1990 databook.

Sales Network

WSI's international sales network includes regional sales managers, field applications engineers, manufacturers representatives and many of the leading component distributors in the United States, Europe and Asia. See Section 7.

United States

Direct sales and field applications engineering offices in Boston, Chicago, Huntsville, Philadelphia, Los Angeles areas and Fremont, CA; more than 25 manufacturers' representatives for major national accounts; national distributors including Schweber Electronics, Time Electronics and Wyle Laboratories; and regional distributors.

International

Distributors in West Germany, England, France, Italy, Sweden, Finland, Denmark, Norway, Spain, Belgium, Luxembourg, the Netherlands, and Israel. Distributors for the Asia/Pacific Rim region in Japan, Korea, Taiwan, Hong Kong and Australia.

WAFERSCALE INTEGRATION, INC.

Ordering Information

High-Performance CMOS Products

WS57C----- -35 D I B

Basic Part Number

Manufacturing Process:

(Blank) = WSI Standard Manufacturing Flow
B = MIL-STD-883C Manufacturing Flow

Operating Temperature Range:

(Blank) = Commercial: 0° to +70°C
V_{CC}: +5V ± 5%

I = Industrial: −40° to +85°C
V_{CC}: +5V ± 10%

M = Military: −55° to +125°C
V_{CC}: +5V ± 10%

Package:

		Package:	Window
A	=	PPGA Plastic Pin Grid Array	No
B	=	0.900″ Size Brazed Ceramic DIP	No
C	=	CLLCC Ceramic Leadless Chip Carrier	Yes*
D	=	0.600″ CERDIP	Yes
F	=	Ceramic Flatpack	Yes*
G	=	CPGA Ceramic Pin Grid Array	No
H	=	Ceramic Flatpack	No*
J	=	Plastic Leaded Chip Carrier	No*
K	=	0.300″ Thin CERDIP	No
L	=	CLDCC Ceramic Leaded Chip Carrier	Yes*
N	=	CLDCC Ceramic Leaded Chip Carrier	No*
P	=	0.600″ Plastic DIP	No
Q	=	Plastic Quad Flatpack	No*
R	=	Ceramic Side Brazed	Yes
S	=	0.300″ Thin Plastic DIP	No
T	=	0.300″ Thin CERDIP	Yes
W	=	Waffle Packed Dice	—
X	=	Ceramic Pin Grid Array	Yes
Y	=	0.600″ CERDIP	No
Z	=	CLLCC	No

Speed:

−35 = 35 ns
−55 = 55 ns
−70 = 70 ns
Etc.

*Surface Mount

Section Index

PSD Product Specifications

For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

MAP168/PSD301 Introduction

User-Configurable Peripheral with Memory

Overview

In 1988 WSI introduced a new concept in programmable VLSI: the Programmable System™ Device (PSD). The PSD is defined as a family of *User-configurable system level building blocks on-a-chip enabling quick implementation of application specific controllers and peripherals.* The first generation PSD series includes the MAP168, a User-Configurable Peripheral with Memory; the SAM448, a User-Configurable Microsequencer; and the PAC1000, a User-Configurable Microcontroller.

The MAP168 is a high-performance, user-configurable peripheral with memory. It is used in DSP applications including modems, motor control and medical instrumentation. The MAP168 is ideal for DSP based applications where fast time-to-market, small form factor and low power consumption are essential. When combined together in an 8- or 16-bit system, virtually any DSP chip (TMS320 series, etc.) and the MAP168 work together to create a very powerful 2-piece chip-set. This implementation provides the core of the required control and peripheral elements of a DSP system.

The MAP168 contains three elements normally associated with discrete solutions to system memory requirements. It incorporates EPROM and SRAM plus a Programmable Address Decoder (PAD), all on the same die. The MAP168 is ideal for the systems designer who wishes to reduce the board space of his final design. By using the MAP168 in a system, five or six EPROM, SRAM and decode logic chips may be reduced into a single 44 pin PLDCC, CLDCC or PGA package.

The second generation PSD301 is a user-configurable peripheral for microcontroller applications including disk drives, low cost modems, and mobile phones. The PSD301 is ideal for microcontroller based applications where fast time-to-market, small form factor and low power consumption are essential. When combined together in an 8- or 16-bit system, virtually any microcontroller (8051, 8096, 16000, etc.) and the PSD301 work together to create a very powerful 2-piece chip-set. Together, this implementation provides all the required control and peripheral elements of a microcontroller based system peripheral with no external "glue" logic required.

Architecture

The MAP168 and PSD301 products incorporate the flexibility of using discrete memory addressing and decoding. With the support of WSI's user friendly PSD software called MAPLE, designers may configure their MAP168/PSD301 subsystems for 8 or 16 bit data paths. If the host system uses an 8051 microcontroller, the MAP168/PSD301 can be programmed with an eight bit data path. A sixteen bit data path can be programmed for microcontrollers like Intel's 80196. The depth of the memory organization will be modified accordingly to accept the different data path widths. The low cost MAPLE software package will handle the data path width adjustment automatically. The user can select either 16K bytes of EPROM and 4K bytes of SRAM or 8K words of EPROM and 2K words of SRAM. The flexibility of the MAP168/PSD301 products enables two devices to be cascaded in width. It is possible to double the memory size of a sixteen bit system by using two MAP168 products in parallel but programmed in a byte-wide configuration. For example, with two MAP168 devices, 16K words of EPROM and 4K words of SRAM may be organized as upper and lower data bytes of a 16 bit word. Alternately, two MAP168 chips may expand the system memory vertically as two word organized memory devices. A block diagram of the MAP168 is shown in Figure 1.

An important feature of the MAP168/PSD301 products is their ability to incorporate the memory address decoding on-chip. One

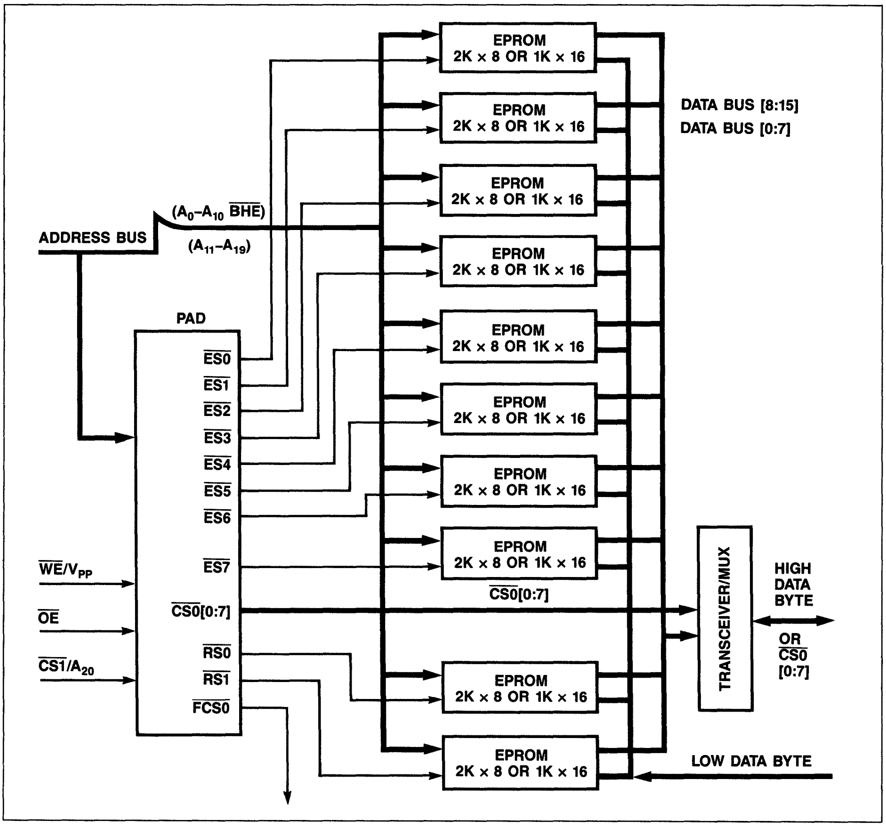
Architecture (Cont.)

MAP168 memory peripheral can reside with other MAP168 devices in the same memory addressing scheme, with the on-chip decoder allocating the memory blocks to different non-conflicting segments of the entire memory area. The decoding function is achieved by an on-chip feature called a Programmable Address Decoder (PAD), which is similar to a single fuse array programmable logic device supporting one product term (AND gate) per output in the MAP168 and four product terms per output in the PSD301.

In the MAP168, eighteen standard chip select outputs from the PAD are available with one fast chip select output generally used to select other external high speed memory devices. The chip select lines may be subdivided into $\overline{\text{ES0}}$–$\overline{\text{ES7}}$, active low internal EPROM chip selects, and two internal RAM chip selects $\overline{\text{RS0}}$ and $\overline{\text{RS1}}$. In byte-wide applications, eight chip select outputs drive external pins $\overline{\text{CS0}}$–$\overline{\text{CS7}}$. These can be used as external chip selects for other MAP168 devices or system memory. These outputs are not available for word-wide MAP168 configurations because the $\overline{\text{CS0}}$–$\overline{\text{CS7}}$ output pins carry the higher order data byte. Only $\overline{\text{FCS0}}$ is available for external chip selection.

Figure 1 shows the organization of the EPROM and SRAM in relation to the PAD, for the MAP168 device.

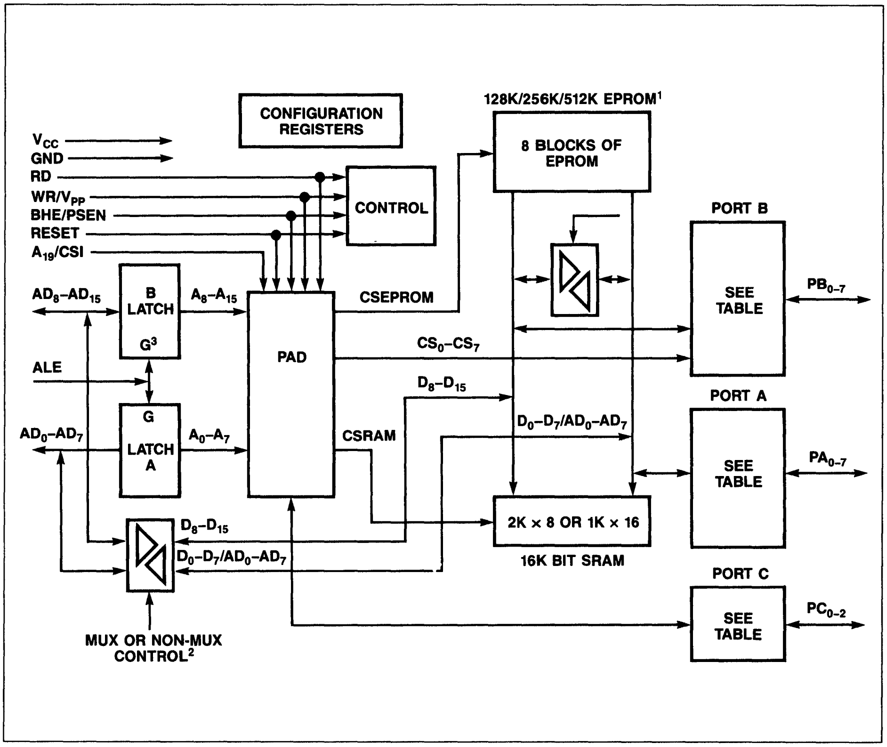
Figure 1. MAP168 Memory Architecture



Important Features:

- 40 ns EPROM/SRAM Access Time.
- Byte or Word Operation, Mappable into 1M Word or 2M Byte Address Space
- 22 ns Chip-Select 8 Outputs, 17 ns Fast Chip Select Output.
- 128K EPROM Bits, 32K SRAM Bits, On-Chip Programmable Decoder, Security Bit.

Figure 2. PSD301 Family Architecture



	By 8 Configuration		By 16 Configuration		
	Port A	Port B	Port A	Port B	Port C
Non-MUX Address Data^5	D_0–D_7	CS_0–CS_7 or PB_0–PB_7	D_0–D_7^{4}	D_8–D_{15}	CS_8–CS_{10}^{6} A_{16}–A_{18}
MUX Address Data	A_0–A_7^{4} PA_0–PA_7 AD_0–AD_7	CS_0–CS_7^{4} PB_0–PB_7	A_0–A_7^{4} PA_0–PA_7 AD_0–AD_7	CS_0–CS_7 PB_0–PB_7	

NOTES:
1. Three MAP300 EPROM densities.
2. Internal signal can be set during programming.
3. Latch B can be set to be transparent (not dependent on ALE).
4. Each I/O pin can be individually set to perform one of the two functions.
5. The non-MUX configuration is compatible to MAP168 pinout.
6. Port C is independent of any configuration and can be chip select out or address in.

Software Support

The object code generated for the support microprocessor/microcontroller is generated by an assembler. This code, when generated as an Intel MCS file, may be easily programmed into the EPROM section of the MAP168/PSD301 device because the MAPLE software has been designed to accept this standard format.

The programmable address decoder is used to define the mapping of the various EPROM and SRAM memory blocks. This mapping is achieved by the designer in the MAPLE environment. The software provides a safeguard that prevents the designer from inadvertently overlapping the address selection. After selecting the memory block assignments, the MAP168/PSD301 device may be programmed by the WSI MagicPro™ memory and PSD programmer.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

MAP168

User-Configurable Peripheral with Memory

Features

- ❑ First-generation Programmable System Device (PSD)
  - User-Configurable Peripheral with Memory
  - 16Kx8 EPROM
  - 4Kx8 SRAM
  - Programmable address decoder
- ❑ Byte or Word Memory Configurations
  - 16Kx8 or 8Kx16 EPROM
  - 4Kx8 or 2Kx16 SRAM
  - 2Mbyte or 1 Mword address range
- ❑ High-Speed Operation
  - 40-nsec memory access
  - 17-nsec fast chip select output
- ❑ External Chip Select Outputs
  - 8 external chip selects
  - 1 fast chip-select output
- ❑ Programmable Security
  - Protects memory map
  - Protects program code
- ❑ Programming Support Tools
  - PSD integrated software environment
  - PC-XT/AT/PS2 platform support
  - MAPLE location entry Software
  - MAPPRO device programming Software
  - MagicPro device programmer (PC-XT, AT)
- ❑ Military and Commercial Specifications
  - 44-pin Ceramic Leaded Chip Carrier package
  - 44-pin Plastic Leaded Chip Carrier package
  - 44-pad Ceramic Leadless Chip Carrier package
  - 44-pin Ceramic Pin Grid Array package

General Description

In 1988 WSI introduced a new concept in programmable VLSI, Programmable System Devices (PSD). The PSD family consists of user-configurable system-level building blocks on-a-chip, enabling quick implementation of application-specific controllers and peripherals. The first generation PSD series includes the MAP168 User-Configurable Peripheral with Memory; the SAM448, a User-Configurable Microsequencer; and the PAC1000, a User-Configurable Microcontroller.

The MAP168 is the first of WSI's Programmable System Devices (PSD) product line. The device integrates high performance, user-configurable blocks of EPROM, SRAM, and logic in a single circuit. The major functional blocks include a Programmable Address Decoder (PAD), 16K bytes of high speed EPROM, and 4K bytes of high speed SRAM. A block diagram is given in Figure 1.

The MAP168 device is a complete memory subsystem that can be mapped anywhere in a 2M-byte address space of a microprocessor or microcontroller system. The EPROM and SRAM memory blocks can be user-configured in either byte-wide or word-wide organizations. The MAP168 device significantly reduces the board space and power necessary to implement memory subsystems, increases system performance, and provides for secure data or program storage.

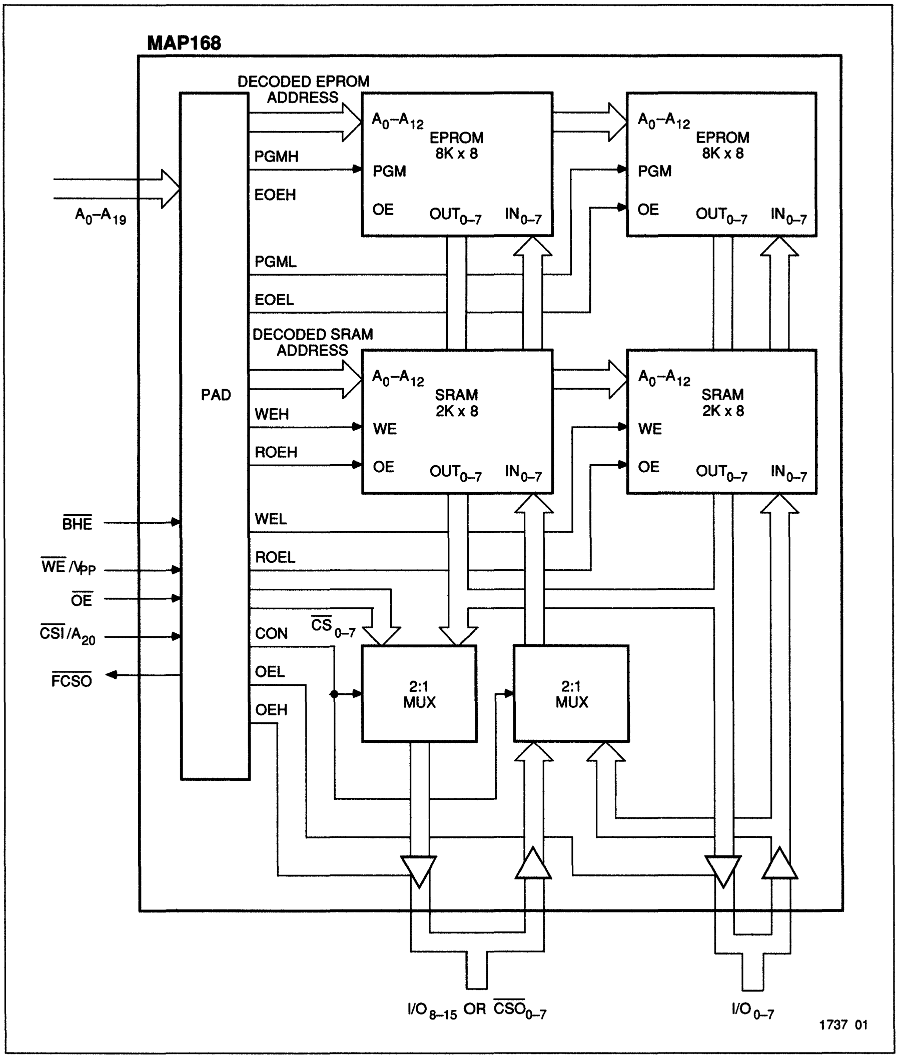
The device's high level of integration and flexibility make it ideal for high-speed microprocessors, microcontrollers, and Digital Signal Processors like the TMS320XX family. The EPROM can be configured either as 16Kx8 or 8Kx16. The SRAM can be configured either as 4Kx8 or 2Kx16. Individual memory blocks of 2Kx8 or 1Kx16 can be selectively mapped anywhere in the address space. Since the Chip Select Input ($\overline{CSI}$) can be programmed as A20, the highest-order address bit, the device's address range can extend from 1M byte with $\overline{CSI}$ to 2M byte without $\overline{CSI}$.

For 16-bit microprocessors capable of byte operations, the MAP168 device provides a Byte High Enable input for accessing bytes on any address boundary.

Pinout is compatible with the JEDEC WS27C257 256K high-speed EPROM. This pinout provides for memory expansion with future WSI EPROM and PSD products.

The device's PAD and EPROM memory are

Figure 1.
Block Diagram



General Description (Con't)

programmed using the same WSI MagicPro programmer used to program other WSI devices. Two software packages, MAPLE Location Entry and MAPPRO Device Programming Software are available in the menu-driven WISPER software environment on an IBM® PC XT/AT or 100% compatible platform.

For additional information on the MAP168 device, refer to *Application Note No. 002, Introduction to the MAP168 User-Configurable Peripheral with Memory.* For additional information on development and programming software for the MAP168 device, refer to the *MAP168 User-Configurable Peripheral with Memory Software User's Manual.*

Functional Description

The user-configurable architecture of the MAP168 consists of an EPROM memory block, an SRAM memory block, and a fast Programmable Address Decoder (PAD) that can be configured to select 2K-byte memory blocks anywhere in a 2M-byte address range. The device can be programmed to operate with memory configured either in a byte or word organization (bytes can be addressed in word mode). A programmable security bit prevents access to the PAD address-decode configuration table.

Table 1. Pin Description

Signal	I/O	Description
A_{0-19}	I	*Address Lines.* For access to EPROM or SRAM.
$\overline{\text{FCSO}}$	O	*Fast Chip-Select Output (active low).* Used by the Programmable Address Decoder (PAD).
$\overline{\text{BHE}}$	I	*Byte High Enable (active low).* Selects the high-order byte when writing to SRAM.
$\overline{\text{WE}}/V_{pp}$	I	*Write Enable (active low) or Programming Voltage.* In normal mode, this pin causes data on the I/O pins to be written into SRAM. In programming mode, the pin supplies the programming voltage, V_{pp}.
$\overline{\text{OE}}$	I	*Output Enable (active low).* Enable the I/O pins to drive the external bus.
$\overline{\text{CSI}}/A_{20}$	I	*Chip Select Input (active low) or High-Order Address.* This pin can be programmed as the bus-access chip select or as an additional high-order address bit (A_{20}).
I/O_{0-7}	I/O	*Low-Order Byte of EPROM or SRAM.*
$I/O_{8-15,}$ $\overline{\text{CSO}}_{0-7}$	I/O	*High-Order Byte or Chip-Select Outputs.* In word mode, these pins serve as the high-order byte ($I/O_{8-15,}$) of EPROM or SRAM. In byte mode, the bits serve as Chip-Select Out signals ($\overline{\text{CSO}}_{0-7}$) for the Programmable Address Decoder (PAD).

Programmable Address Decoder

The MAP168 device has a minimum of 20 address inputs A_0–A_{19} allowing the EPROM and SRAM memory blocks to reside anywhere in a 1M-byte address space. If the $\overline{\text{CSI}}/A_{20}$ input is user-configured as an address line, the maximum addressable space increases to 2M bytes, as shown in the Configurations table.

The 16K bytes of EPROM and 4K bytes of SRAM, can be configured into eight independent 2K-byte blocks and two 2K-byte blocks respectively, as shown in the Memory Architecture figure. The PAD is a user-configurable address decoder that compares input addresses to the 2K-byte address range selected for each of the eight EPROM blocks and two SRAM blocks. When the input address A_0–A_{20} is detected to be within one of the EPROM or SRAM address ranges, the PAD enables an internal chip select (ES_0–ES_7 or RS_0–RS_1) to the selected block. If no block is selected, both the EPROM and SRAM memories remain in a power-down mode and the outputs are disabled allowing other devices to drive the data bus. The SRAM retains its data in the power-down mode. The 2K-byte address ranges for any of the eight EPROM or two SRAM blocks may not overlap.

The PAD can also be user-configured to generate up to eight external chip selects, $\overline{CS}_0$–$\overline{CS}_7$. These outputs can be used to decode the input address lines A_0–A_{20} and to select other devices in the system. The outputs $\overline{CS}_0$–$\overline{CS}_7$ are available on the eight higher-order I/O_8–I/O_{15} lines but only when the MAP168 device is configured in the byte mode; the lines are not available as chip-select outputs when the device is configured in the word mode.

The $\overline{\text{CSI}}/A_{20}$ input is user-configurable as the most-significant address line or as an active-low chip enable. Its function is programmed as part of the PAD programming cycle.

The PAD also provides $\overline{\text{FSCO}}$, a single, fast chip-select output configurable by the user for any address. It can overlap with any of the internal EPROM, SRAM or external $\overline{\text{CSO}}$ addresses.

Memory Subsystem

EPROM Memory

The memory configuration of the MAP168 device includes 128K bits of WSI's patented high-speed, split-gate, UV-erasable EPROM. The EPROM is configured in byte mode as 16Kx8 and in word mode as 8Kx16. The memory is organized as eight 2Kx8 or 1Kx16 blocks, as shown in the Block Diagram figure. Each block has a separate and independent address range that cannot overlap. Each block is individually selected by one of the ES_0–ES_7 internal chip selects generated by the PAD when an input address is detected within its designated address range, as shown in the Memory Architecture figure. If not selected, each block of EPROM remains in a power-down mode.

For programming, the EPROM memory requires the $\overline{\text{WE}}/V_{PP}$ input to maintain the programming voltage V_{PP}.

SRAM Memory

The device also includes 32K bits of high-speed SRAM. The SRAM is configured in byte mode as 4Kx8 and in word mode as 2Kx16. The memory is organized as two 2Kx8 or one 2Kx16 block(s), each with a separate and independent address range that cannot overlap. Each SRAM block is individually selected by one of the RS_0–RS_1, shown in the Memory Architecture figure, when an input address is detected by the PAD within its designated address range. When not selected, each of the SRAM memory blocks remains in a power down mode but does retain all data stored.

Data can be written into the SRAM only when the $\overline{\text{WE}}/V_{PP}$ input is active low.

Memory Subsystem EPROM Memory (Con't)

Byte/Word Mode

The PAD can be programmed to configure the MAP168 device for either a byte or word memory architecture. This allows the device to be used conveniently with either 8-bit or 16-bit microcontrollers, microprocessors or digital signal processor (DSP) systems. See the Configurations table.

In byte mode, the EPROM is organized as 16Kx8 and the SRAM as 4Kx8. The outputs of both are tied to the eight low-order input/output lines I/O$_0$–I/O$_7$ and enabled onto the output bus when the $\overline{OE}$ input is low.

Only when configured in byte mode are the eight external chip selects provided by the PAD available on the eight high-order input/output lines I/O$_8$–I/O$_{15}$ and enabled onto the output bus when the $\overline{OE}$ input is low.

In word mode, the EPROM is organized as 8Kx16 and the SRAM as 2Kx16. The outputs of both are tied to the 16 input/output lines I/O$_0$–I/O$_{15}$ and enabled onto the bus when $\overline{OE}$ is low.

In word mode, the $\overline{BHE}$ input along with address input A0 allows the eight bits of any 16-bit word on an even or odd boundary to be selected as shown in the High-Low Byte Selection table. This is a useful feature for 16-bit processors that are not restricted to reading or writing memory only on even-word address boundaries.

Mode Selection

The device's operational mode is controlled by three inputs, $\overline{CSI}$, $\overline{OE}$, and $\overline{WE}/V_{PP}$. There are ten separate modes of operation, all of which are shown the Mode Selection table.

Table 2. Configurations

	x8 Configuration		***x16 Configuration***	
	$\overline{CSI}$	A_{20}	$\overline{CSI}$	A_{20}
Address Space words	1M bytes	2M bytes	512K words	1M
Block Size words	2K bytes	2K bytes	1K words	1K
Addressable Blocks	512	1024	512	1024
EPROM Blocks	8	8	8	8
SRAM Blocks	2	2	2	2
Chip-Select Outputs	9	9	1	1
EPROM Configuration	16Kx8	16Kx8	8Kx16	8Kx16
SRAM Configuration	4Kx8	4Kx8	2Kx16	2Kx16
I/O Pins	8	8	16	16
Low-power Standby	yes	no	yes	no
Protected Mode	yes	yes	yes	yes
Byte Operations	yes	yes	yes	yes

Table 3. Mode Selection

Mode/Pin	$\overline{CSI}$	$\overline{OE}$	$\overline{WE}/V_{pp}$	Address	x16 (I/O_{0-15}) x8 (I/O_{0-7})	x16 ($\overline{FCSO}$) x8 $\overline{FCSO}$, $\overline{CSO}_{0-7}$
Read EPROM/SRAM	V_{IL}	V_{IL}	V_{IH}	EPROM/SRAM Selected	D_{OUT}	CS_{OUT}
Read External	V_{IL}	V_{IL}	V_{IH}	EPROM/SRAM Not Selected	High Z	CS_{OUT}
Output Disable	X	V_{IH}	X	X	High Z	CS_{OUT}
Stand-By	V_{IH}	X	X	X	High Z	CS_{OUT}
Write SRAM	V_{IL}	X	V_{IL}	SRAM Selected	D_{IN}	CS_{OUT}
Write External	V_{IL}	X	V_{IL}	No SRAM Selected	X	CS_{OUT}
Program EPROM	V_{IL}	V_{IH}	V_{PP}	EPROM Program Address	D_{IN}	D_{IN}
Program Verify EPROM	V_{IL}	V_{IL}	V_{IH}	EPROM Program Address	D_{OUT}	CS_{OUT}
Program PAD	V_{IL}	V_{IH}	V_{PP}	PAD Program Address	D_{IN}	D_{IN}
Program Verify PAD	V_{IL}	V_{IL}	V_{IH}	PAD Program Address	D_{OUT}	CS_{OUT}

Table 4. High/Low Byte Selection

x16 Configuration Only

$\overline{BHE}$ (Pin 1)	A_0	Write Operation	Read Operation
0	0	Whole word	Whole word
0	1	Upper byte from/to odd address	Upper byte = Data Out Lower byte = 'FF'
1	0	Lower byte from/to even address	Whole word
1	1	None	Upper byte = Data Out Lower byte = 'FF'

$\overline{WR}$ and $\overline{BHE}$ are used for SRAM functions

Table 5. Product Selection Guide

Parameter	MAP168-40	MAP168-45	MAP168-55	Units
Address Access Time (max)	40	45	55	ns
Chip-Select Access Time (max)	40	45	55	ns
Output Enable Time (max)	18	21	23	ns
Chip-Select Output Time	22	25	27	ns
Fast Chip-Select Output Time (max)	17	20	22	ns

Table 6. DC Characteristics

Parameter	Symbol	Test Conditions	Min	Max	Units
Output Low Voltage	V_{OL}	I_{OL}=8 mA		0.5	V
Output High Voltage	V_{OH}	I_{OH}=–2 mA	2.4		V
CMOS Standby Current	I_{SB1}	notes 1, 3			
—Commercial				20	mA
—Military				30	mA
TTL Standby Current	I_{SB2}	notes 2, 3			
—Commercial				30	mA
—Military				40	mA
CMOS Active Current No Blocks Selected	I_{CC} 1A	notes 1, 4			
—Commercial				20	mA
—Military				30	mA
CMOS Active Current EPROM Block Selected	I_{CC} 1B	notes 1, 4			
—Commercial				35	mA
—Military				45	mA
CMOS Active Current SRAM Block Selected	I_{CC} 1C	notes 1, 4			
—Commercial				55	mA
—Military				65	mA
TTL Active Current No Blocks Selected	I_{CC} 2A	notes 2, 4			
—Commercial				30	mA
—Military				40	mA
TTL Active Current EPROM Block Selected	I_{CC} 2B	notes 2, 4			
—Commercial				40	mA
—Military				50	mA
TTL Active Current SRAM Block Selected	I_{CC} 2C	notes 2, 4			
—Commercial				65	mA
—Military				75	mA
Input Load Current	I_{LI}	V_{IN}=5.5V or GND	–10	10	µA
Output Leakage Current	I_{LO}	V_{OUT}=5.5V or GND	–10	10	µA

Notes:

1. CMOS inputs: GND ± 0.3V or VCC ± 0.3V.
2. TTL inputs: $V_{IL} \leq 0.8V$, $V_{IH} \geq 2.0V$.
3. Add 1.5 mA/MHz for AC power component.
4. Add 3.5 mA/MHz for AC power component.

Table 7. AC Characteristics

Parameter	Symbol	MAP168-40		MAP168-45		MAP168-55		Units
		Min	Max	Min	Max	Min	Max	
Read Cycle Time	t_{RC}	40		45		55		ns
Address to Output Delay	t_{ACC}		40		45		55	ns
$\overline{\text{CSI}}$ to Output Delay	t_{CE}		40		45		55	ns
$\overline{\text{OE}}$ to Output Delay	t_{OE}		18		21		23	ns
Output Disable to Output Float	t_{OEF}		15		18		20	ns
Chip Disable to Output Float	t_{CSF}		15		18		20	ns
Address to Output Hold	t_{OH}	10		10		10		ns
Address to $\overline{\text{CSO}}_{0-7}$ True	t_{CSO}		22		25		27	ns
Address to $\overline{\text{FCSO}}$ True	t_{FCSO}		17		20		22	ns
SRAM Write Cycle Time	t_{WC}	40		45		55		ns
Chip Enable to Write End	t_{CSW}	40		45		55		ns
Address Setup Time	t_{AS}	0		0		0		ns
Address Hold Time	t_{AH}	0		0		0		ns
Address Valid to Write End	t_{AW}	40		45		55		ns
SRAM Write Enable Pulse Width	t_{PWE}	25		30		35		ns
Data Setup Time	t_{DS}	20		20		30		ns
Data Hold Time	t_{DH}	0		0		0		ns
Write Enable to Data Float	t_{WEF}		18		21		23	ns
Write Disable to Data Low Z	t_{WELZ}	3		3		3		ns
$\overline{\text{BHE}}$ Setup Time	t_{BHES}	0		0		0		ns
$\overline{\text{BHE}}$ Hold Time	t_{BHEH}	10		10		10		ns

Table 8. Data Retention Characteristics

Parameter	Symbol	Test Conditions	Min	Max	Units
Minimum V_{CC} for Data Retention	V_{DR}	V_{CC}=2.0V,	2.0		V
Current in Data Retention Mode	I_{CCDR}	$\overline{\text{CSI}} \geq V_{CC}–0.2V$,		1	mA
Chip Deselect to Data Retention	t_{CSDR}	$V_{IN} \geq V_{CC}–0.2V$	0		ns
Recovery Time from Data Retention	t_{RDR}	or $V_{IN} \leq 0.2V$	t_{RC}		ns

Absolute Maximum Ratings

Storage Temperature	–65°C to +150°C
Voltage to any pin with respect to GND	–0.6V to +7V
V_{PP} with respect to GND	–0.6 V to +14.0V
ESD Protection	>2000V

Stresses above those listed here may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect device reliability.

Table 9. Operating Range

Range	*Temperature*	*V_{CC}*
Commercial	0° to +70°C	+5V ± 5%
Military	–55° to +125°C	+5V ± 10%

Figure 3. Read Cycle Timing Diagram

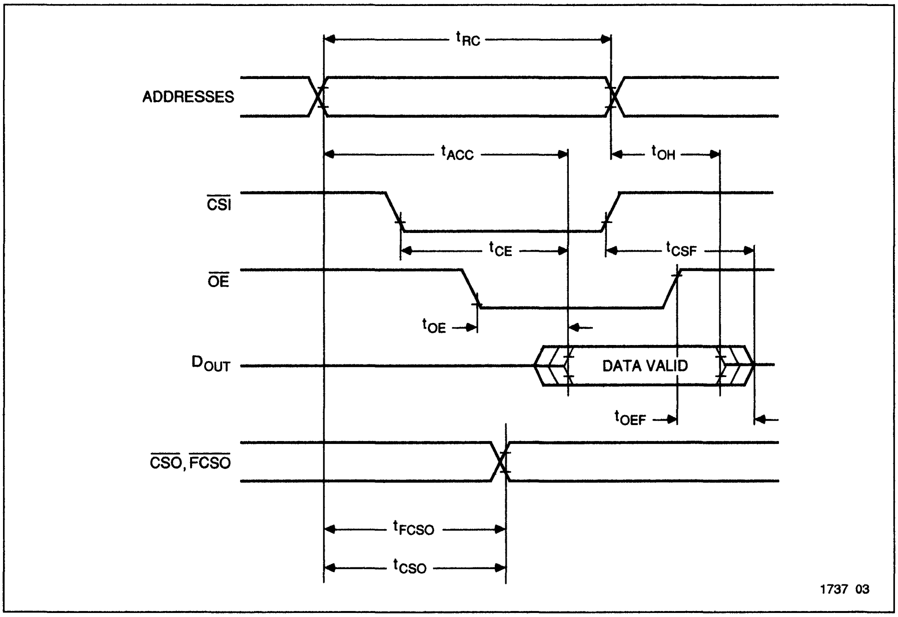
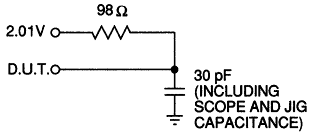


Figure 4. Test Load



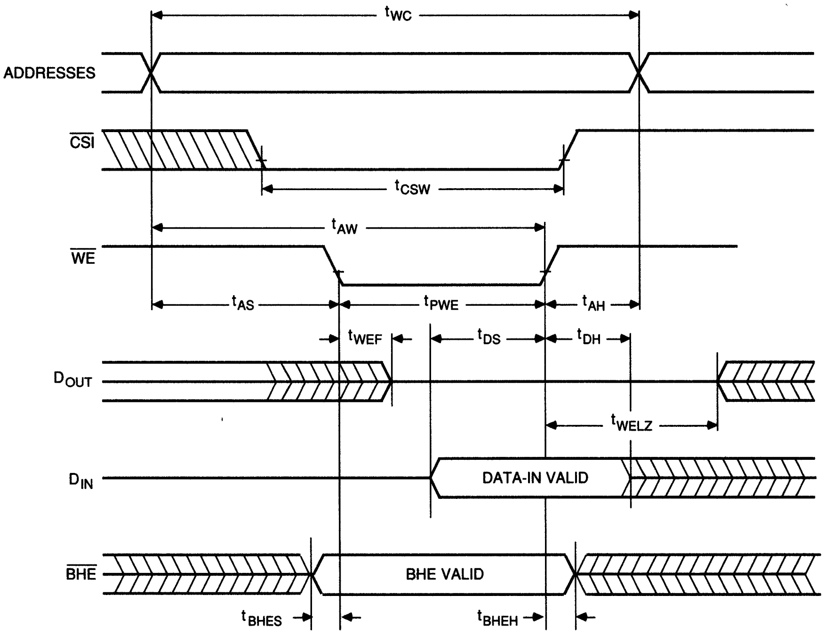
High-impedance test systems

1737 04

Table 10. Timing Levels

Level	Voltage
Input	0 and 3V
Reference	1.5V

Figure 5. Write Cycle Timing Diagram



1737 05

Figure 6. Memory Architecture

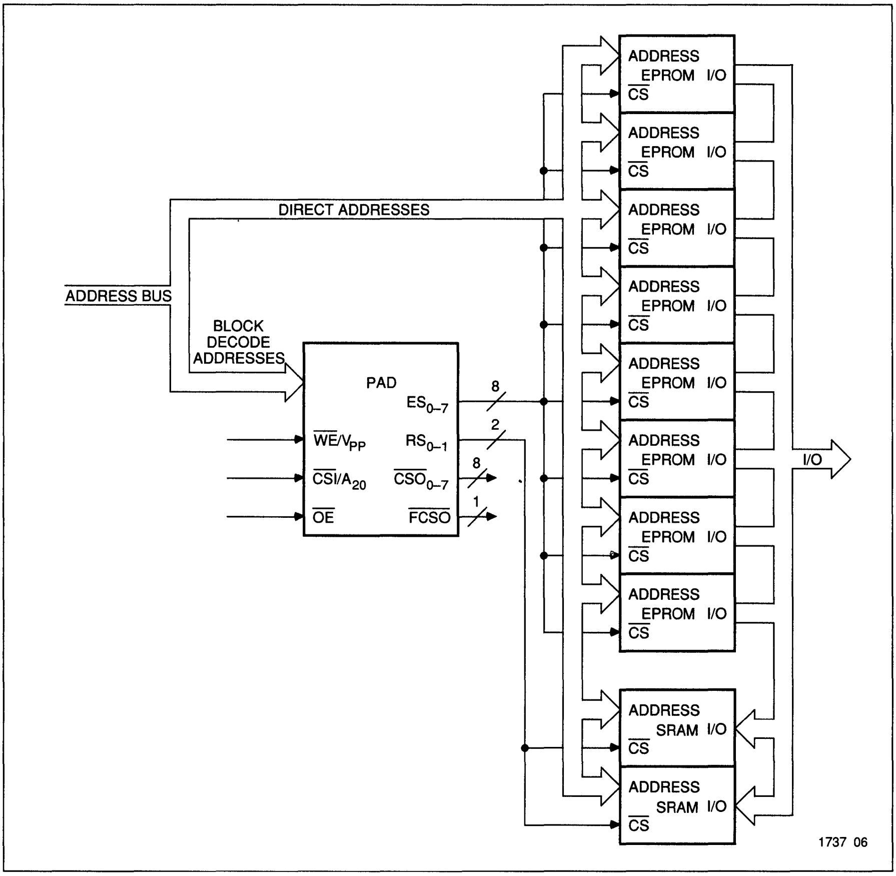


Table 11. MAP168 Pin Assignments

44-pin CLDCC Package
44-pin PLDCC Package
44-pad CLLCC Package

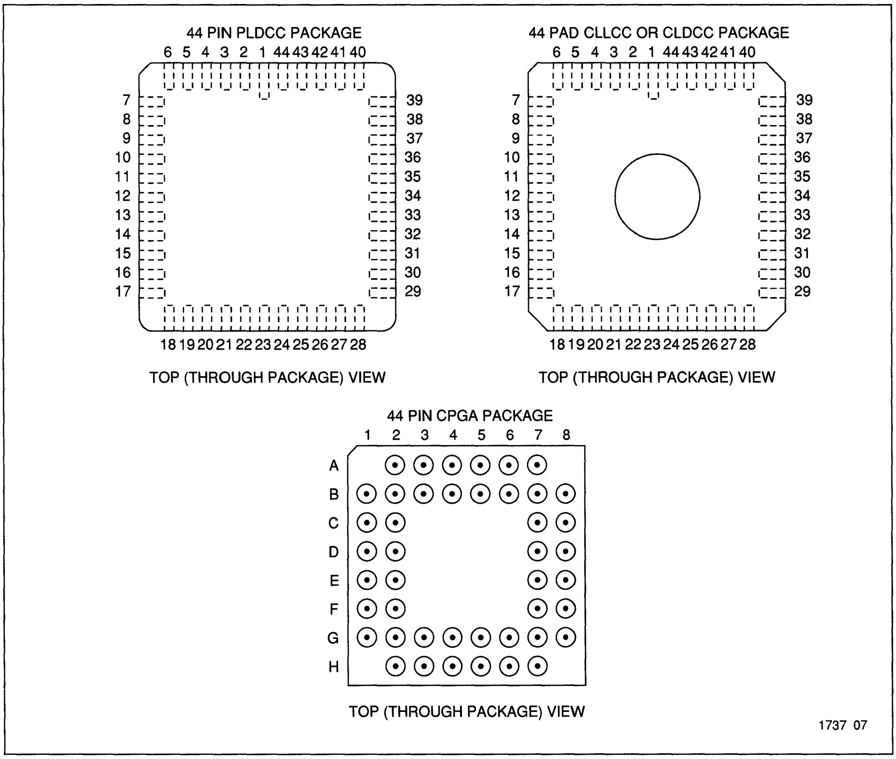
Pin No.	*x8*	*x16*
1	GND	$\overline{BHE}$
2	$\overline{WE}/V_{PP}$	$\overline{WE}/V_{PP}$
3	$\overline{CSI}/A_{20}$	$\overline{CSI}/A_{20}$
4	$\overline{CSO_7}$	I/O_{15}
5	$\overline{CSO_6}$	I/O_{14}
6	$\overline{CSO_5}$	I/O_{13}
7	$\overline{CSO_4}$	I/O_{12}
8	$\overline{CSO_3}$	I/O_{11}
9	$\overline{CSO_2}$	I/O_{10}
10	$\overline{CSO_1}$	I/O_9
11	$\overline{CSO_0}$	I/O_8
12	GND	GND
13	$\overline{FCSO}$	$\overline{FCSO}$
14	I/O_7	I/O_7
15	I/O_6	I/O_6
16	I/O_5	I/O_5
17	I/O_4	I/O_4
18	I/O_3	I/O_3
19	I/O_2	I/O_2
20	I/O_1	I/O_1
21	I/O_0	I/O_0
22	$\overline{OE}$	$\overline{OE}$
23	A_0	A_0
24	A_1	A_1
25	A_2	A_2
26	A_3	A_3
27	A_4	A_4
28	A_5	A_5
29	A_6	A_6
30	A_7	A_7
31	A_8	A_8
32	A_9	A_9
33	A_{10}	A_{10}
34	GND	GND
35	A_{11}	A_{11}
36	A_{12}	A_{12}
37	A_{13}	A_{13}
38	A_{14}	A_{14}
39	A_{15}	A_{15}
40	A_{16}	A_{16}
41	A_{17}	A_{17}
42	A_{18}	A_{18}
43	A_{19}	A_{19}
44	V_{CC}	V_{CC}

$\overline{WE}$ and $\overline{BHE}$ are for SRAM functions.

Table 12. MAP168 Pin Assignments

44-pin CPGA Package		
Pin No.	**x8**	**x16**
A_5	GND	$\overline{\text{BHE}}$
A_4	$\overline{\text{WE}}/V_{PP}$	$\overline{\text{WE}}/V_{PP}$
B_4	$\overline{\text{CSI}}/A_{20}$	$\overline{\text{CSI}}/A_{20}$
A_3	$\overline{\text{CSO}}_7$	I/O_{15}
B_3	$\overline{\text{CSO}}_6$	I/O_{14}
A_2	$\overline{\text{CSO}}_5$	I/O_{13}
B_2	$\overline{\text{CSO}}_4$	I/O_{12}
B_1	$\overline{\text{CSO}}_3$	I/O_{11}
C_2	$\overline{\text{CSO}}_2$	I/O_{10}
C_1	$\overline{\text{CSO}}_1$	I/O_9
D_2	$\overline{\text{CSO}}_0$	I/O_8
D_1	GND	GND
E_1	$\overline{\text{FCSO}}$	$\overline{\text{FCSO}}$
E_2	I/O_7	I/O_7
F_1	I/O_6	I/O_6
F_2	I/O_5	I/O_5
G_1	I/O_4	I/O_4
G_2	I/O_3	I/O_3
H_2	I/O_2	I/O_2
G_3	I/O_1	I/O_1
H_3	I/O_0	I/O_0
G_4	$\overline{\text{OE}}$	$\overline{\text{OE}}$
H_4	A_0	A_0
H_5	A_1	A_1
G_5	A_2	A_2
H_6	A_3	A_3
G_6	A_4	A_4
H_7	A_5	A_5
G_7	A_6	A_6
G_8	A_7	A_7
F_7	A_8	A_8
F_8	A_9	A_9
E_7	A_{10}	A_{10}
E_8	GND	GND
D_8	A_{11}	A_{11}
D_7	A_{12}	A_{12}
C_8	A_{13}	A_{13}
C_7	A_{14}	A_{14}
B_8	A_{15}	A_{15}
B_7	A_{16}	A_{16}
A_7	A_{17}	A_{17}
B_6	A_{18}	A_{18}
A_6	A_{19}	A_{19}
B_5	V_{CC}	V_{CC}

Figure 7. Pin Assignments Programming



Upon delivery from WSI or after each erasure (see Erasure section), the MAP168 device has all bits in the PAD and EPROM in the “one” or high state. Zeros are loaded through the procedure of programming.

Information for programming the device is available directly from WSI. Please contact your local sales representative.

Erasure

To clear all locations of their programmed contents, expose the device to an ultra-violet light source. A dosage of 15W-second/cm^2 is required. This dosage can be obtained with exposure to a wavelength of 2537Å and intensity of 1200µW/cm^2 for 15 to 20 minutes. The device should be about one inch from the source and all filters should be removed from the UV light source prior to erasure.

The MAP168 device and similar devices will erase with light sources having wavelengths shorter than 4000Å. Although erasure times will be much longer than with UV sources at 2537Å, the exposure to fluorescent light and sunlight will eventually erase the device; for maximum system reliability, these sources should be avoided. If used in such an environment, the package windows should be covered by an opaque label or substance.

System Development Tools

MAP168 System Development Tools are a complete set of PC-based development tools. Installed on an IBM PC or compatible computer, these tools provide an integrated, easy-to-use software and hardware environment to support MAP168 device development. The tools run on an IBM-XT, AT, or compatible computer running MS-DOS version 3.1 or later. The system must be equipped with 640K bytes of RAM and a hard disk.

System Development Tools (Con't)

Hardware

The MAP168 System Programming Hardware consists of:

- ❑ WS6000 MagicPro Memory and PSD Programmer
- ❑ WS6003 44-pin LCC Package Adaptor (for 44-pin CLLCC, CLDCC, and PLDCC packages)
- ❑ WS6011 44-pin CPGA Package Adaptor

The MagicPro Programmer is the common hardware platform for programming all WSI programmable products. It consists of the IBM-PC plug-in Programmer Board and the Remote Socket Adaptor Unit.

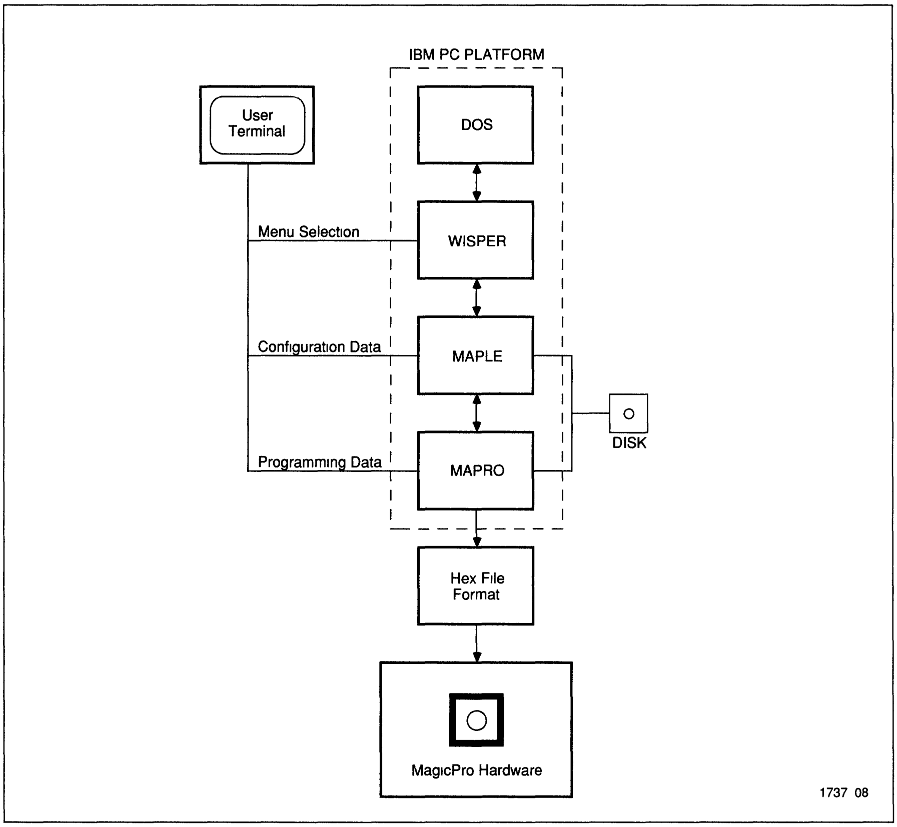
Software

The MAP168 System Development Software consists of the following:

- ❑ WISPER Software—PSD Software Environment
- ❑ MAPLE Software—MAP168 Location Editor
- ❑ MAPPRO Software—Device Programming Software

The configuration of the MAP168 device is entered using MAPLE software. MAPRO software configures MAP168 devices by using the MagicPro programmer and the socket adaptor. The programmed MAP168 is then ready to be used. The development cycle is depicted in Figure 8.

Figure 8. MAP168 Development Cycle



System Development Tools (Con't)

Support

WSI provides a complete set of quality support services to registered System Development Tools owners. These support services include the following:

❑ 12-month Software Updates.

❑ Hotline to WSI Application Experts—For direct design assistance.

❑ 24-Hour Electronic Bulletin Board—For design assistance via dial-up modem.

Training

WSI provides in-depth, hands-on workshops for the MAP168 device and System Development Tools. Workshop participants learn how to program their own high-performance, user-configurable mappable memory subsystems. Workshops are held at the WSI facility in Fremont, California.

Ordering Information

MAP168 Part Number	Speed (ns)	Package Type	Package Drawing	Operating Temperature	Manufacturing Procedure
MAP168-40C*	40	44-pad CLLCC	C3	Commercial	Standard
MAP168-40J*	40	44-pin PLDCC	J2	Commercial	Standard
MAP168-40L*	40	44-pin CLDCC	L4	Commercial	Standard
MAP168-45C	45	44-pad CLLCC	C3	Commercial	Standard
MAP168-45CM*	45	44-pad CLLCC	C3	Military	Standard
MAP168-45CMB*	45	44-pad CLLCC	C3	Military	MIL-STD-883C
MAP168-45J	45	44-pin PLDCC	J2	Commercial	Standard
MAP168-45L	45	44-pin CLDCC	L4	Commercial	Standard
MAP168-45LM*	45	44-pad CLDCC	L4	Military	Standard
MAP168-45LMB*	45	44-pad CLDCC	L4	Military	MIL-STD-883C
MAP168-45X	45	44-pin CPGA	X2	Commercial	Standard
MAP168-45XM*	45	44-pin CPGA	X2	Military	Standard
MAP168-45XMB*	45	44-pin CPGA	X2	Military	MIL-STD-883C
MAP168-55C	55	44-pad CLLCC	C3	Commercial	Standard
MAP168-55CM	55	44-pad CLLCC	C3	Military	Standard
MAP168-55CMB	55	44-pad CLLCC	C3	Military	MIL-STD-883C
MAP168-55J	55	44-pin PLDCC	J2	Commercial	Standard
MAP168-55L	55	44-pin CLDCC	L4	Commercial	Standard
MAP168-55LM	55	44-pin CLDCC	L4	Military	Standard
MAP168-55LMB	55	44-pin CLDCC	L4	Military	MIL-STD-883C
MAP168-55X	55	44-pin CPGA	X2	Commercial	Standard
MAP168-55XM	55	44-pin CPGA	X2	Military	Standard
MAP168-55XMB	55	44-pin CPGA	X2	Military	MIL-STD-883C

*These products are advanced information.

Ordering Information

System Development Tools

Part Number	*Contents*
MAP168-GOLD	WISPER Software MAPLE Software User's Manual WSI-SUPPORT WS6000 MagicPro Programmer
MAP168-SILVER	WISPER Software MAPLE Software User's Manual WSI-SUPPORT
WS6000	MagicPro Programmer IBM PC plug-in Adaptor Card Remote Socket Adaptor
WS6003	44-pin LCC Package Adaptor for 44-pin CLLCC, CLDCC, and PLDCC Packages. Used with the WS6000 MagicPro Programmer.
WS6011	44-pin CPGA Package Adaptor. Used with the WS6000 MagicPro Programmer.
WSI-SUPPORT	Support Services including: □ 12-month Software Update Service □ Hotline to WSI Application Experts □ 24-hour access to WSI Electronic Bulletin Board
WSI-TRAINING	Workshops at WSI, Fremont, CA. For details and scheduling, call PSD Marketing, (415) 656-5400.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

PSD301

User-Configurable Peripheral with Memory

Preliminary

Key Features

- ❑ Second Generation Programmable System Device
- ❑ User-Configurable Peripheral for Microcontroller Based Applications — Enables rapid design implementation and fast time to market
- ❑ Available in space saving surface mount and through-hole packages
- ❑ Windowed package option for prototyping
- ❑ Low cost OTP (one-time programmable) package for high volume applications
- ❑ CMOS for low power consumption
- ❑ User-Configurable to Interface with Any 8- or 16-Bit Microcontroller
  - — Programmable Address Decoder (PAD)
  - — Programmable Control Signals
  - — Programmable Polarity
  - — Built-In Address Latches
  - — Multiplexed or Non-Multiplexed Address/Data Buses
  - — Selectable 8- or 16-Bit Bus Width
  - — Power-Down
  - — Address Inputs Can Be Latched or Transparent
  - — Latched Low-Order Address Byte Available as Output
- ❑ Port Expansion/Reconstruction of Up to 16 I/O Lines
  - — Individually Configurable as Output or Input
- ❑ Highly Configurable, Many Operational Modes
- ❑ High-Density UV EPROM
  - — 256K Bits Configurable as 32K x 8 or as 16K x 16
  - — Divided Into Eight Equal Mappable Blocks
  - — EPROM Block Resolution of 4K Bytes or 2K Words
  - — EPROM: Up to 120 ns Access Time (Including PAD Decoding Time)
- ❑ Static RAM
  - — 16K Bits Configurable as 2K x 8 or as 1K x 16
  - — SRAM: Up to 120 ns Access Time (Including PAD Decoding Time)
- ❑ Addressable Range
  - — 1 MByte or 0.5 MWords
- ❑ Low Power TTL-Compatible CMOS Device

2

Applications

- ❑ Computers (Workstations and PCs) — Fixed Disk Control, Modem, Imaging, Laser Printer Control
- ❑ Telecommunications — Modem, Cellular Phone, Digital PBX, Digital Speech, FAX, Digital Signal Processing
- ❑ Industrial — Robotics, Power Line Access, Power Line Monitor
- ❑ Medical Instrumentation — Hearing Aids, Monitoring Equipment, Diagnostic Tools
- ❑ Military — Missile Guidance, Radar, Sonar, Secure Communications, RF Modems

Product Description

In 1988 WSI introduced a new concept in programmable VLSI, Programmable System Devices. The PSD family consists of user-configurable system-level building blocks on-a-chip, enabling quick implementation of application-specific controllers and peripherals. The first generation PSD series includes the MAP168, a User-Configurable Peripheral, which is ideal for DSP applications; the SAM448, a User-Configurable Microsequencer for control and interface applications, and the PAC1000, a User-Configurable Microcontroller.

The PSD301 is a second generation PSD. The PSD301 is ideal for microcontroller based applications where fast time-to-market, small form factor and low power consumption are essential. When combined together in an 8- or 16-bit system, virtually any microcontroller (8051, 8096, 16000, etc.) and the PSD301 work together to create a very powerful 2-piece chip-set. This implementation provides all the required control and peripheral elements of a microcontroller based system peripheral with no external "glue" logic required.

The PSD301 integrates high performance user-configurable blocks of EPROM, SRAM, and logic in a single circuit. The major functional blocks include a Programmable Address Decoder (PAD), 256K bits of high speed EPROM, 16K bits of high speed SRAM, input latches, and output ports. The PSD301 is ideal for applications requiring high performance, low power, and very small form factors. These include fixed disk control, modem, cellular telephone, instrumentation, computer peripherals, military and similar applications.

The PSD301 is an optimal solution for microcontrollers that need:

- ❑ I/O reconstruction (microcontrollers lose at least two I/O ports when accessing external resources).
- ❑ More EPROM and SRAM than the microcontroller's internal memory.
- ❑ Chip-select, control, or latched address lines that are otherwise implemented discretely.
- ❑ An interface to shared external resources.

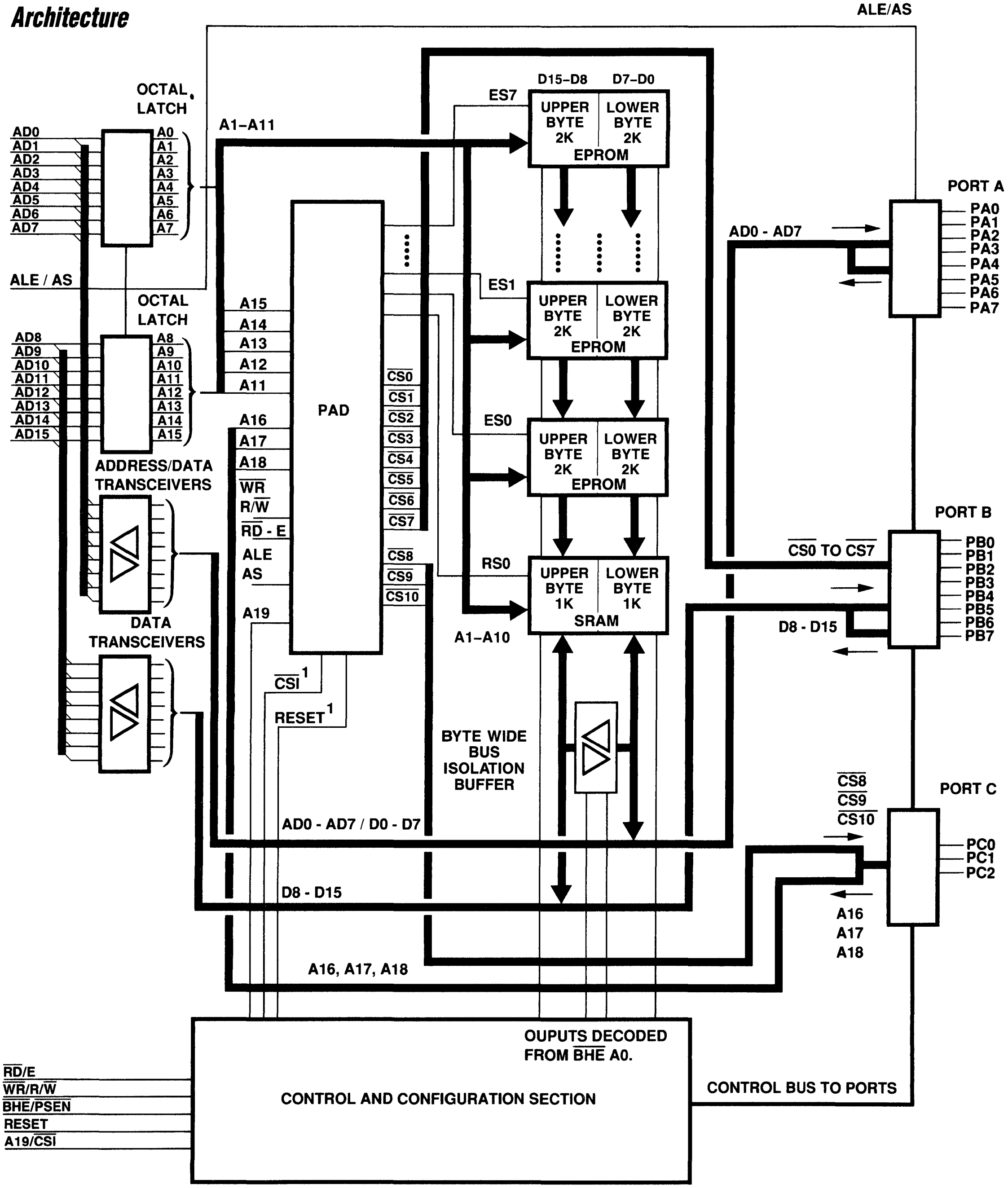
The PSD301 (shown in Figure 1) can efficiently interface with, and enhance, any 8- or 16-bit microcontroller system. No other solution provides microcontrollers with port expansion, latched addresses, a programmable address decoder (PAD), an interface to shared resources, 256 kbit EPROM, and 16 kbit SRAM on a single chip. The PSD301 does not require glue logic for interfacing to any 8- or 16-bit microcontrollers.

The 8051 microcontroller family can take full advantage of the PSD301's separate program and address spaces. Users of the 68HCXX family of microcontrollers can change the functionality of the control signals and directly connect the R/$\overline{W}$ and E signals. Users of 16-bit microcontrollers (including the 80186, 8096, 80196, 16XXX) can use the PSD301 in a 16-bit configuration. Address and data buses can be configured to be separated or multiplexed, whichever is required by the host processor.

The flexibility of the PSD301 I/O ports permit interfacing to shared resources. The user can assign the following functions to these ports: standard I/O pins, chip select outputs from the PAD, latched address or multiplexed low-order address/data byte. This enables users to design add-on systems such as disk drives, modems, etc., that easily interface to the host bus (e.g., IBM PC, SCSI).

The PSD301's on-chip programmable address decoder (PAD) enables the user to map the I/O ports, eight segments of EPROM (as 4K x 8, or as 2K x 16), SRAM (as 2K x 8 or as 1K x 16), and chip select outputs anywhere in the address space of the microcontroller. The PAD can implement up to 4 sum-of-product expressions based on address inputs and control signals. This further facilitates the interface to microcontrollers with different boot-up locations and I/O address mappings, e.g., the 8051 and 8096 microcontrollers have the boot-up addresses in the lower half of their memory maps; the 80186 and 68HCXX use high memory boot-up addresses.

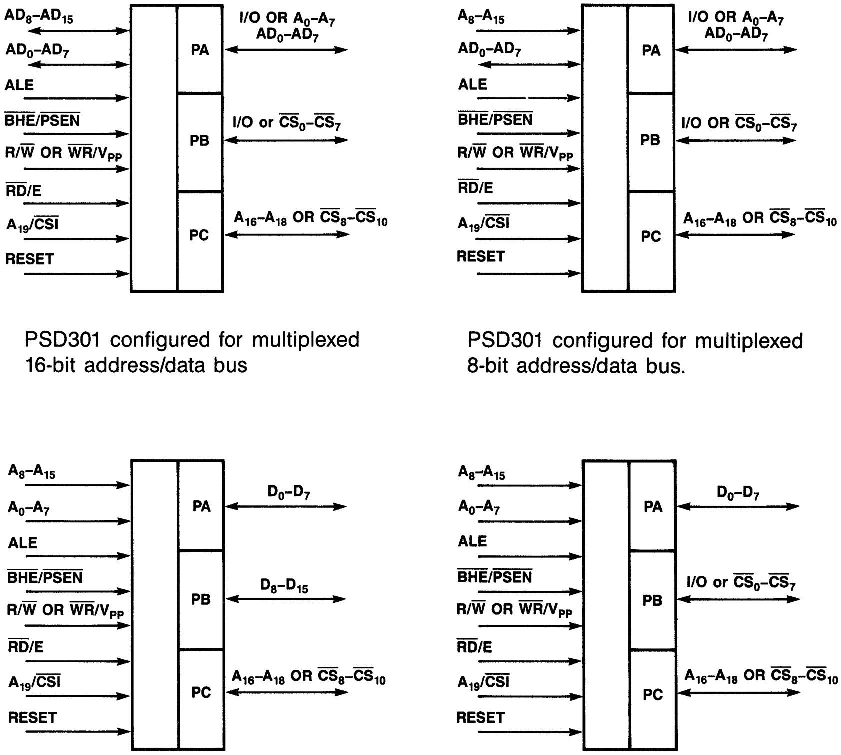
Figure 1. PSD301 Architecture



NOTES:
1. RESET and CSI are not available as programmable options in the PAD. An active RESET ensures that the PAD deselects all of its outputs, and a high level on CSI ensures that the PAD is in power-down mode.
2. Details of the PAD as a programmable array decoder are given in Figure 3.

Figure 2. PSD301 Port Configurations

Figure 2 shows the PSD301's I/O port configurations.


AD_8–AD_{15}
AD_0–AD_7
ALE
$\overline{BHE}/PSEN$
$R/\overline{W}$ OR $\overline{WR}/V_{PP}$
$\overline{RD}/E$
$A_{19}/\overline{CSI}$
RESET
PA
PB
PC
I/O OR A_0–A_7 AD_0–AD_7
I/O or $\overline{CS}_0$–$\overline{CS}_7$
A_{16}–A_{18} OR $\overline{CS}_8$–$\overline{CS}_{10}$
A_8–A_{15}
AD_0–AD_7
ALE
$\overline{BHE}/PSEN$
$R/\overline{W}$ OR $\overline{WR}/V_{PP}$
$\overline{RD}/E$
$A_{19}/\overline{CSI}$
RESET
PA
PB
PC
I/O OR A_0–A_7 AD_0–AD_7
I/O OR $\overline{CS}_0$–$\overline{CS}_7$
A_{16}–A_{18} OR $\overline{CS}_8$–$\overline{CS}_{10}$
A_8–A_{15}
A_0–A_7
ALE
$\overline{BHE}/PSEN$
$R/\overline{W}$ OR $\overline{WR}/V_{PP}$
$\overline{RD}/E$
$A_{19}/\overline{CSI}$
RESET
PA
PB
PC
D_0–D_7
D_8–D_{15}
A_{16}–A_{18} OR $\overline{CS}_8$–$\overline{CS}_{10}$
A_8–A_{15}
A_0–A_7
ALE
$\overline{BHE}/PSEN$
$R/\overline{W}$ OR $\overline{WR}/V_{PP}$
$\overline{RD}/E$
$A_{19}/\overline{CSI}$
RESET
PA
PB
PC
D_0–D_7
I/O or $\overline{CS}_0$–$\overline{CS}_7$
A_{16}–A_{18} OR $\overline{CS}_8$–$\overline{CS}_{10}$

PSD301 configured for multiplexed 16-bit address/data bus

PSD301 configured for multiplexed 8-bit address/data bus.

PSD301 configured for non-multiplexed 16-bit address/data bus.

PSD301 configured for non-multiplexed 8-bit address/data bus.

Legend:

AD_0–AD_7 = addresses A_0–A_7 multiplexed with data lines D_0–D_7.

AD_8–AD_{15} = addresses A_8–A_{15} multiplexed with data lines D_8–D_{15}.

Table 1. PSD301 Pin Descriptions

Name	Type	Description
$\overline{\text{BHE}}$/$\overline{\text{PSEN}}$	I	When the data bus width is 8 bits (CDATA = 0), this pin is $\overline{\text{PSEN}}$. In this mode, $\overline{\text{PSEN}}$ is the active low EPROM read pulse. The SRAM and I/O ports read signal is generated when $\overline{\text{RD}}$ is low (CRRWR = 0), or when E and R/$\overline{\text{W}}$ are high (CRRWR = 1). If the host processor is a member of the 8031 family, $\overline{\text{PSEN}}$ must be connected to the corresponding host pin. In other 8-bit host processors that do not have a special EPROM-only read strobe, $\overline{\text{PSEN}}$ should be tied to V_{CC}. In this case, $\overline{\text{RD}}$ or E and R/$\overline{\text{W}}$ provide the read strobe for the SRAM, I/O ports, and EPROM. When the data bus width is configured as 16 (CDATA = 1), this pin is $\overline{\text{BHE}}$. When $\overline{\text{BHE}}$ is low, a high-order byte is read from, or written into the PSD301, depending on the operation being read or write, respectively. In programming mode, this pin is pulsed between V_{PP} and 0.
$\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$/V_{PP}	I	In the operating mode, this pin's function is $\overline{\text{WR}}$ (CRRWR = 0) or R/$\overline{\text{W}}$ (CRRWR = 1). When configured as $\overline{\text{WR}}$, a write operation is executed during an active low pulse. When configured as R/$\overline{\text{W}}$, with R/$\overline{\text{W}}$ = 1 and E = 1, a read operation is executed; if R/$\overline{\text{W}}$ = 0 and E = 1, a write operation is executed. In programming mode, this pin must be tied to V_{PP} voltage.
$\overline{\text{RD}}$/E	I	When configured as $\overline{\text{RD}}$ (CRRWR = 0), this pin provides an active low $\overline{\text{RD}}$ strobe. When configured as E (CRRWR = 1), this pin becomes an active high pulse, which, together with R/$\overline{\text{W}}$ defines the cycle type. Then, if R/$\overline{\text{W}}$ = 1 and E = 1, a read operation is executed. If R/$\overline{\text{W}}$ = 0 and E = 1, a write operation is executed.
$\overline{\text{CSI}}$/A19	I	This pin has two configurations. When it is $\overline{\text{CSI}}$ (CA19/$\overline{\text{CSI}}$ = 0) and the pin is asserted high, the device is deselected and powered down. (See Tables 12 and 13 for the chip state during power-down mode.) If the pin is asserted low, the chip is in normal operational mode. When it is A19, (CA19/$\overline{\text{CSI}}$ = 1), this pin can be used as an additional input to the PAD. In this mode, there is no power-down capability.
RESET	I	This user-programmable pin can be configured to reset on high level (CRESET = 1) or on low level (CRESET = 0). It should remain active for at least 100 ns. See Tables 10 and 11 for the chip state after reset.
ALE or AS	I	In the multiplexed modes, the ALE pin functions as an Address Latch Enable or as an Address strobe and can be configured as an active high or active low signal. The ALE or AS trailing edge latches lines AD15/A15–AD0/A0, A16–A19, and $\overline{\text{BHE}}$, depending on the PSD301 configuration. See Table 8. In the non-multiplexed modes, it can be used as a general-purpose PAD input signal.

Legend: The I/O column abbreviations are: I = input; I/O = input/output; P = power.

NOTE: 3. All the configuration bits mentioned in Table 1 appear in parentheses and are explained in the Configuration Register section.

Table 1. PSD301 Pin Descriptions (Cont.)

Name	Type	Description
PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0	I/O	PA7–PA0 is an 8-bit port that can be configured to track AD7/A7–AD0/A0 from the input (CPAF2 = 1). Otherwise (CPAF2 = 0), each bit can be configured separately as an I/O or lower-order latched address line. When configured as an I/O (CPAF1 = 0), the direction of the pin is defined by its direction bit, which resides in the direction register. If a pin is an I/O output, its data bit (which resides in the data register) comes out. When it is configured as a low-order address line (CPAF1 =1), A7–A0 can be made the corresponding output through this port (e.g., PA6 can be configured to be the A6 address line). Each port bit can be a CMOS output (CPACOD = 0) or an open drain output (CPACOD = 1). When the chip is in non-multiplexed mode (CADDRDAT = 0), the port becomes the data bus lines (D0–D7). See Figure 4.
PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0	I/O	PB7–PB0 is an 8-bit port for which each bit can be configured as an I/O (CPBF = 1) or chip-select output (CPBF = 0). Each port bit can be a CMOS output (CPBCOD = 0) or an open drain output (CPBCOD = 1). When configured as an I/O, the direction of the pin is defined by its direction bit, which resides in the direction register. If a pin is an I/O output, its data (which resides in the data register) comes out. When configured as a chip-select output, $\overline{\text{CS0}}$–$\overline{\text{CS3}}$ are a function of up to four product terms of the inputs to the PAD; $\overline{\text{CS4}}$–$\overline{\text{CS7}}$ then are each a function of up to two product terms. When the chip is in non-multiplexed mode (CADDRAT = 0) and the data bus width is 16 (CDATA = 1), the port becomes the most significant byte of the data bus (D8–D15). See Figure 6.
PC0 PC1 PC2	I/O	This is a 3-bit port for which each bit is configurable as a PAD input or output. When configured as an input (CPCF = 0), the bits can be latched with ALE (CADDHLT = 1) or be transparent inputs to the PAD (CADDHLT = 0). When a pin is configured as an output (CPCF = 1), it is a function of one product term of all PAD inputs. See Figure 7.
AD0/A0 AD1/A1 AD2/A2 AD3/A3 AD4/A4 AD5/A5 AD6/A6 AD7/A7	I/O	In multiplexed mode, these pins are the multiplexed low-order address/data byte. After ALE latches the addresses, these pins input or output data, depending on the settings of the $\overline{\text{RD}}$/E, $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$, and $\overline{\text{BHE}}$/$\overline{\text{PSEN}}$ pins. In non-multiplexed mode, these pins are the low-order address input byte.
AD8/A8 AD9/A9 AD10/A10 AD11/A11 AD12/A12 AD13/A13 AD14/A14 AD15/A15	I/O	In 16-bit multiplexed mode, these pins are the multiplexed high-order address/data byte. After ALE latches the addresses, these pins input or output data, depending on the settings of the $\overline{\text{RD}}$/E, $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$, and $\overline{\text{BHE}}$/$\overline{\text{PSEN}}$ pins. In all other modes, these pins are the high-order address input byte.
GND	P	V_{SS} (ground) pin.
V_{CC}	P	Supply voltage input.

Operating Modes

The PSD301's four operating modes allow it to interface directly to 8- and 16-bit microcontrollers and microprocessors with multiplexed and non-multiplexed address/data buses. These operating modes are:

- ❑ Multiplexed 8-bit address/data bus
- ❑ Multiplexed 16-bit address/data bus
- ❑ Non-multiplexed address/data, 8-bit data bus
- ❑ Non-multiplexed 16-bit address/data bus

Multiplexed 8-Bit Address/Data Bus

This mode is used to interface to microcontrollers with an 8-bit data bus and a 16-bit or larger address bus. The low-order address/data bus (AD0/A0–AD7/A7) is bi-directional and permits the latching of the address when the ALE signal is active. On the same pins, the data is read from or written to the device; this depends on the state of the $\overline{\text{RD}}$/E, $\overline{\text{BHE}}$/$\overline{\text{PSEN}}$, and $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$ pins. The high-order address/data bus (AD8/A8–AD15/A15) contains the high-order address bus byte. Ports A and B can be configured as in Table 2.

Multiplexed 16-Bit Address/Data Bus

This mode is used to interface to microcontrollers with a 16-bit data bus and a 16-bit or larger address bus. The low-order address/data bus (AD0/A0–AD7/A7) is bi-directional and permits the latching of the address when the ALE signal is active. On the same pins, the data is read from or written to the device; this depends on the state of the $\overline{\text{RD}}$/E, $\overline{\text{BHE}}$/$\overline{\text{PSEN}}$, and $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$ pins. The high-order address/data bus (AD8/A8–AD15/A15) is bi-directional and permits latching of the high-order address when the ALE signal is active on the same pins. The high-order data bus is read from or written to the device, depending on the state of the $\overline{\text{RD}}$/E, $\overline{\text{BHE}}$/$\overline{\text{PSEN}}$, and $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$ pins. Ports A and B can be configured as in Table 2.

Non-Multiplexed Address/Data, 8-Bit Data Bus

This mode is used to interface to non-multiplexed 8-bit microcontrollers with an 8-bit data bus and a 16-bit or larger address bus. The low-order address/data bus (AD0/A0–AD7/A7) is the low-order address input bus. The high-order address/data bus (AD8/A8–AD15/A15) is the high-order address bus byte. Port A is the low-order data bus. Port B can be configured as shown in Table 2.

Non-Multiplexed 16-Bit Address/Data Bus

This mode is used to interface to non-multiplexed 16-bit microcontrollers with a 16-bit data bus and a 16-bit or larger address bus. The low-order address/data bus (AD0/A0–AD7/A7) is the low-order address input bus. The high-order address/data bus (AD8/A8–AD15/A15) is the high-order address bus byte. Port A is the low-order data bus. Port B is the high-order data bus.

Table 2 summarizes the effect of the different operating modes on ports A, B, and the address/data pins. The configuration of Port C is independent of the four operating modes.

Table 2. PSD301 Bus and Port Configuration Options

	Multiplexed Address/Data	Non-Multiplexed Address/Data
8-Bit Data Bus		
Port A	I/O and/or low-order address lines or Low-order multiplexed address/data byte	D0–D7 data bus lines
Port B	I/O and/or $\overline{\text{CS0}}$–$\overline{\text{CS7}}$	I/O and/or $\overline{\text{CS0}}$–CS7
AD0/A0–AD7/A7	Low-order multiplexed address/data byte	Low-order address bus byte
AD8/A8–AD15/A15	High-order address bus byte	High-order address bus byte
16-Bit Data Bus		
Port A	I/O and/or low-order address lines or Low-order multiplexed address/data byte	Low-order data bus byte
Port B	I/O and/or $\overline{\text{CS0}}$–$\overline{\text{CS7}}$	High-order data bus byte
AD0/A0–AD7/A7	Low-order multiplexed address/data byte	Low-order address bus byte
AD8/A8–AD15/A15	High-order multiplexed address/data byte	High-order address bus byte

Programmable Address Decoder (PAD)

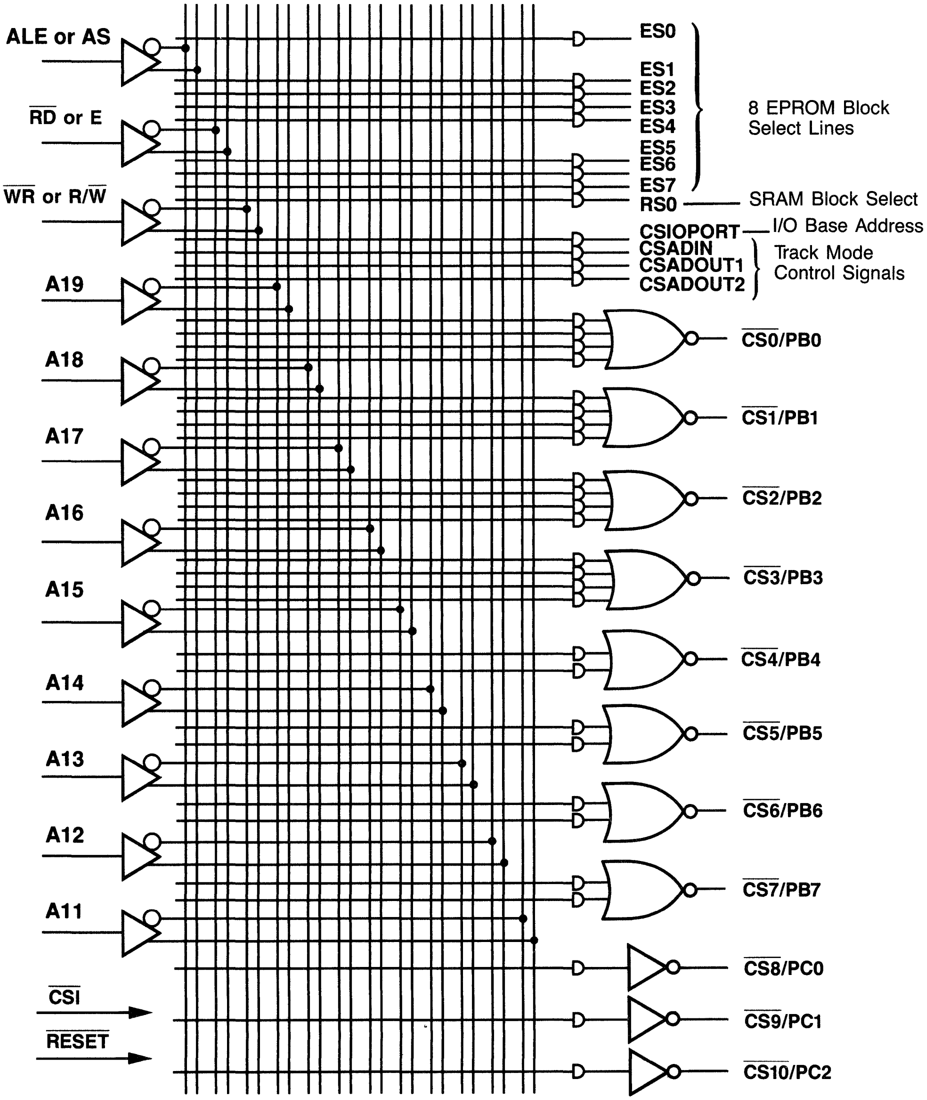
The PSD301's programmable address decoder (PAD) has 14 inputs and 24 outputs. All its I/O functions are listed in Table 3 and shown in Figure 3.

The PAD is used to select all chip internal parts and to generate external chip-selects (see Figure 3). Pins A11–A15, $\overline{\text{RD}}$/E, $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$, Reset, and ALE are fixed functions. A16–A19 can be address inputs or general purpose inputs to the PAD for implementing logic functions. Internal and external PAD select signals can override EPROM memory whose addresses overlap. This lets the user make more efficient use of the address space. For example, if the EPROM is not used completely for program storage, the unused EPROM address space can be allocated to I/O ports, SRAM, or other PAD select signals. Using WSI's MAPLE software, any input function to the PAD can be selected as active low, active high, or don't care.

Table 3. PSD301 I/O Functions

	Function
PAD Inputs	
$\overline{\text{CSI}}$ or A19	In $\overline{\text{CSI}}$ mode (when high), PAD deselects all of its outputs and enters a power-down mode (see Tables 12 and 13). In A19 mode, it is another input to the PAD.
A16–A18	These are general purpose inputs from Port C. See Figure 3, note 4.
A11–A15	These are address inputs.
$\overline{\text{RD}}$ or E	This is the read pulse or enable strobe input.
$\overline{\text{WR}}$ or R/$\overline{\text{W}}$	This is the write pulse or R/$\overline{\text{W}}$ select signal.
ALE	This is the ALE input to the chip.
RESET	This deselects all outputs from the PAD; it can not be used in product term equations. See Tables 10 and 11.
PAD Outputs	
$\overline{\text{CS0}}$–$\overline{\text{CS3}}$	These chip-select outputs can be routed through Port B. Each of them is a function of up to four product terms of the PAD inputs.
$\overline{\text{CS4}}$–$\overline{\text{CS7}}$	These chip-select outputs can be routed through Port B. Each of them is a function of up to two product terms of the PAD inputs.
$\overline{\text{CS8}}$–$\overline{\text{CS10}}$	These chip-select outputs can be routed through Port C. See Figure 3, note 4. Each of them is a function of one product term of the PAD inputs.
ES0–ES7	These are internal chip-selects to the 8 EPROM banks. Each bank can be located on any boundary that is a function of one product term of the PAD address inputs.
RS0	This is an internal chip-select to the SRAM. Its base address location is a function of one term of the PAD address inputs.
CSIOPORT	This internal chip-select selects the I/O ports. It can be placed on any boundary that is a function of one product term of the PAD inputs. See Tables 6 and 7.
CSADIN	This internal chip-select, when Port A is configured as a low-order address/data bus in the track mode (CPAF2 = 1), controls the input direction of Port A. CSADIN is gated externally to the PAD by the internal read signal. When CSADIN and a read operation are active, data presented on Port A flows out of AD0/A0–AD7/A7. This chip-select can be placed on any boundary that is a function of one product term of the PAD inputs. See Figure 5.
CSADOUT1	This internal chip-select, when Port A is configured as a low-order address/data bus in track mode (CPAF2 = 1), controls the output direction of Port A. CSADOUT1 is gated externally to the PAD by the ALE signal. When CSADOUT1 and the ALE signal are active, the address presented on AD0/A0–AD7/A7 flows out of Port A. This chip-select can be placed on any boundary that is a function of one product term of the PAD inputs. See Figure 5.
CSADOUT2	This internal chip-select, when Port A is configured as a low-order address/data bus in the track mode (CPAF2 = 1), controls the output direction of Port A. CSADOUT2 must include the write-cycle control signals as part of its product term. When CSADOUT2 is active, the data presented on AD0/A0–AD7/A7 flows out of Port A. This chip-select can be placed on any boundary that is a function of one product term of the PAD inputs. See Figure 5.

Figure 3. PSD301 PAD Description



NOTES:

4. $\overline{\text{CSI}}$ is a power-down signal. When high, the PAD is in stand-by mode and all its outputs become non-active. See Tables 12 and 13.
5. RESET deselects all PAD output signals. See Tables 10 and 11.
6. Maximum PAD latency is 35 ns.
7. A18, A17, and A16 are internally multiplexed with $\overline{\text{CS10}}$, $\overline{\text{CS9}}$, and $\overline{\text{CS8}}$, respectively. Either A18 or $\overline{\text{CS10}}$, A17 or $\overline{\text{CS9}}$, and A10 or $\overline{\text{CS8}}$ can be routed to the external pins of Port C.

Configuration Bits

The configuration bits shown in Table 4 are non-volatile cells that let the user set the device, I/O, and control functions to the proper operational mode. Table 5 lists all configuration bits. The configuration bits are programmed and verified during the programming phase. In operational mode, they are not accessible. To simplify implementing a specific mode, use the WSI's PSD301 MAPLE software to set the bits.

Table 4. PSD301 Non-Volatile Configuration Bits

Use This Bit	To
CDATA	Set the data bus width to 8 or 16 bits.
CADDRDAT	Set the address/data buses to multiplexed or non-multiplexed mode.
CRRWR	Set the $\overline{\text{RD}}$/E and $\overline{\text{WR}}$/V_{PP} or R/$\overline{\text{W}}$ pins to $\overline{\text{RD}}$ and $\overline{\text{WR}}$ pulse, or to E strobe and R/W status.
CA19/$\overline{\text{CSI}}$	Set A19/$\overline{\text{CSI}}$ to $\overline{\text{CSI}}$ (power-down) or A19 input.
CALE	Set the ALE polarity.
CPAF2	Set Port A either to track the low-order byte of the address/data multiplexed bus or to select the I/O or address option.[8]
CSECURITY	Set the security on or off.
CRESET	Set the RESET polarity.
$\overline{\text{COMB}}$/SEP	Set $\overline{\text{PSEN}}$ and $\overline{\text{RD}}$ for combined or separate address spaces (see Figures 8 and 9).
CPAF1	Configure each pin of Port A in multiplexed mode to be an I/O or address out.
CPACOD	Configure each pin of Port A as an open drain or active CMOS pull-up output.
CPBF	Configure each pin of Port B as an I/O or a chip-select output.
CPBCOD	Configure each pin of Port B as an open drain or active CMOS pull-up output.
CPCF	Configure each pin of Port C as an address input or a chip-select output.
CADDHLT	Configure pins A16–A19 to go through a latch or to have their latch transparent.

NOTE: 8. CPAF1 determines whether the output is I/O or address.

Table 5. PSD301 Configuration Bits

Configuration Bits	No. of Bits	Description

CDATA	1	8-bit or 16-bit data bus width
		CDATA = 0, 8-bit data bus
		CDATA = 1, 16-bit data bus

CADDRDAT	1	Address/data multiplexed or non-multiplexed (separate buses)
		CADDRDAT = 0, non-multiplexed address/data bus
		CADDRDAT = 1, multiplexed address/data bus

CRRWR	1	CRRWR = 0, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ active low strobes
		CRRWR = 1, R/$\overline{\text{W}}$ status and E active high pulse

CA19/$\overline{\text{CSI}}$	1	A19 or $\overline{\text{CSI}}$ CA19/$\overline{\text{CSI}}$ = 0, enable power-down mode CA19/$\overline{\text{CSI}}$ = 1, A19 input to PAD

CALE	1	Active high or active low
		CALE = 0, active high
		CALE = 1, active low

CRESET	1	Active high or active low CRESET = 0, active low reset signal CRESET = 1, active high reset signal
$\overline{\text{COMB}}$/SEP	1	Combined or separate memory space for EPROM and SRAM $\overline{\text{COMB}}$/SEP = 0, combined $\overline{\text{COMB}}$/SEP = 1, separate

CPAF1	8	Port A I/Os or A0–A7
		CPAF1 = 0, Port A pin = I/O
		CPAF1 = 1, Port A pin = Ai (0 ⩽ i ⩽ 7)

CPAF2	1	Port A AD0–AD7 (address/data multiplexed bus)
		CPAF2 = 0, address or I/O on Port A (according to CPAF1)
		CPAF2 = 1, address/data multiplexed on Port A (track mode)

CPBF	8	Port B I/Os or $\overline{\text{CS0}}$–$\overline{\text{CS7}}$
		CPBF = 0, Port B Pin = $\overline{\text{CSi}}$ (0 ⩽ i ⩽ 7)
		CPBF = 1, Port B Pin = I/O

CPCF	3	Port C A16–A18 or $\overline{\text{CS8}}$–$\overline{\text{CS10}}$
		CPCF = 0, Port C Pin = Ai (16 ⩽ i ⩽ 18)
		CPCF = 1, Port C Pin = $\overline{\text{CSi}}$ (8 ⩽ i ⩽ 10)

CPACOD	8	Port A CMOS or open-drain outputs
		CPACOD = 0, CMOS output
		CPACOD = 1, open-drain output

CPBCOD	8	Port B CMOS or open-drain outputs
		CPBCOD = 0, CMOS output
		CPBCOD = 1, open-drain output

CADDHLT	1	A16–A19 latched or latch transparent
		CADDHLT = 0, address latch transparent
		CADDHLT = 1, address latched (ALE dependent)

CSECURITY	1	Security on or off
		CSECURITY = 0, no security
		CSECURITY = 1, secured part (cannot be copied)

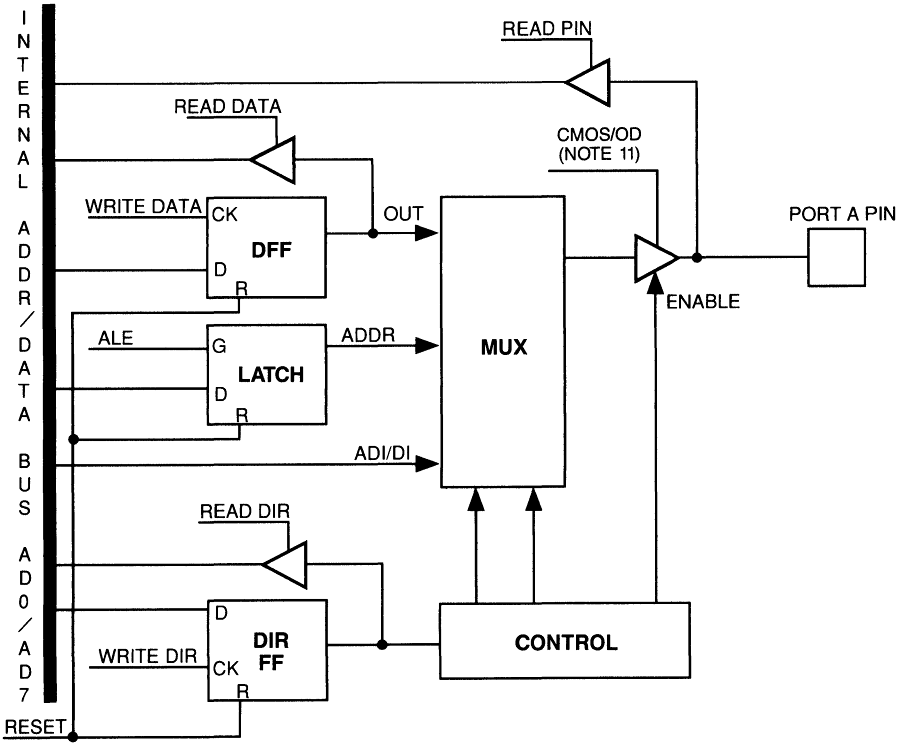
Total Number of Bits	45	

NOTES: 9. WSI's MAPLE software will guide the user to the proper configuration choice.
10 In an unprogrammed or erased part, all configuration bits are 0.

Port Functions

The PSD301 has three I/O ports (Ports A, B, and C) that are configurable at the bit level. This permits great flexibility and a high degree of customization for specific applications. The following is a description of each port. Figure 4 shows the pin structure of Port A.

Figure 4. Port A Pin Structure



NOTE: 11. CMOS/OD determines whether the output is open drain or CMOS.

Port A in Multiplexed Address/Data Mode

The default configuration of Port A is I/O. In this mode, every pin can be set as an input or output by writing into the respective pin's direction flip flop (DIR FF, in Figure 4). As an output, the pin level can be controlled by writing into the respective pin's data flip flop (DFF, in Figure 4). When DIR FF = 1, the pin is configured as an output. When DIR FF = 0, the pin is configured as an input. The controller can read the DIR FF bits by accessing the READ DIR register; it can read the DFF bits by accessing the READ DATA register. Port A pin levels can be read by accessing the READ PIN register. Individual pins can be configured as CMOS or open drain outputs. Open drain pins require external pull-up resistors. For addressing information, refer to Tables 6 and 7.

Alternatively, each bit of Port A can be configured as a low-order latched address bus bit. The address is provided by the port address latch, which latches the address on the trailing edge of ALE. PA0–PA7 can become A0–A7, respectively. This feature of the PSD301 lets the user generate low-order address bits to access external peripherals or memory that require several low-order address lines.

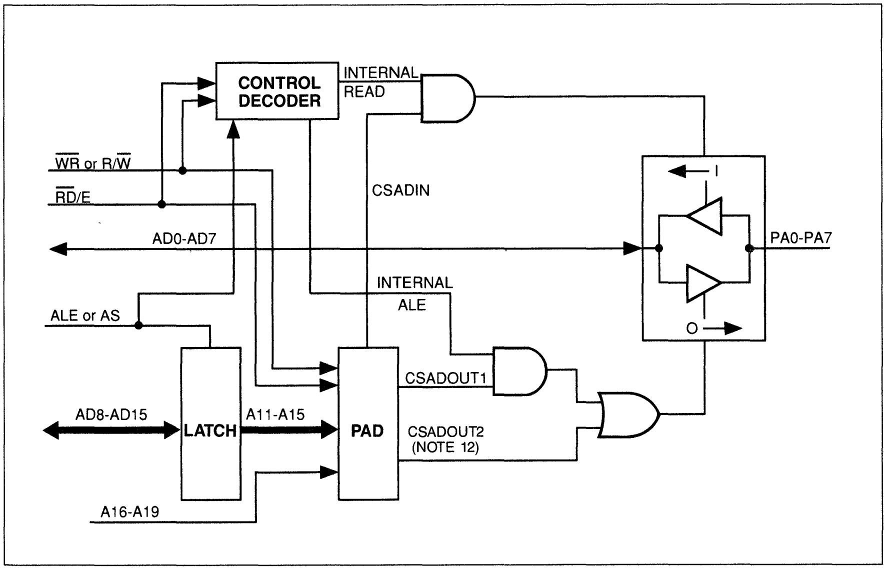
Port Functions (Cont.)

Another mode of Port A (CPAF2 = 1) sets the entire port to track the inputs AD0/A0–AD7/A7, depending on specific address ranges defined by the PAD's CSADIN, CSADOUT1, and CSADOUT2 signals. This feature lets the user interface the microcontroller to shared external resources without requiring external buffers and decoders. In this mode, the port is effectively a bi-directional buffer. The direction is controlled by using the input signals ALE, $\overline{\text{RD}}$/E, $\overline{\text{WR}}$/V$_{\text{PP}}$ or R/$\overline{\text{W}}$, and the internal PAD outputs CSADOUT1, CSADOUT2 and CSADIN (see Figure 5). When CSADOUT1 and ALE are true, the address on the input AD0/A7–AD7/A7 pins flows out through Port A. (Carefully check the generation of CSADOUT1, and ensure that it is stable during the ALE pulse; see Figures 22 and 23). When CSADOUT2 is active, a write operation is performed (see note to Figure 5). The data on the input AD0/A7–AD7/A7 pins flows out through Port A. When CSADIN and a read operation is performed (depending on the mode of the $\overline{\text{RD}}$/E and $\overline{\text{WR}}$/V$_{\text{PP}}$ or R/$\overline{\text{W}}$ pins), the data on Port A flows out through the AD0/A7–AD7/A7 pins. In this operational mode, Port A is tri-stated when none of the above-mentioned three conditions exist.

Port A in Non-Multiplexed Address/ Data Mode

In this mode, Port A becomes the low order data bus byte of the chip. When reading an internal PSD301 location, data is presented on Port A pins. When writing to an internal PSD301 location, data present on Port A pins is written to that location.

Figure 5. Port A Track Mode



NOTE: 12. The expression for CSADOUT2 must include the following write operation cycle signals:
For CRRWR = 0, CSADOUT2 must include $\overline{\text{WR}}$ = 0.
For CRRWR = 1, CSADOUT2 must include E = 1 and R/$\overline{\text{W}}$ = 0.

Port B in Multiplexed Address/Data and in 8-Bit Non-Multiplexed Modes

The default configuration of Port B is I/O. In this mode, every pin can be set as an input or output by writing into the respective pin's direction flip flop (DIR FF, in Figure 6). As an output, the pin level can be controlled by writing into the respective pin's data flip flop (DFF, in Figure 6). When DIR FF = 1, the pin is configured as an output. When DIR FF = 0, the pin is configured as an input. The controller can read the DIR FF bits by accessing the READ DIR register; it can read the DFF bits by accessing the READ DATA register. Port B pin levels can be read by accessing the READ PIN register.

Port Functions (Cont.)

Individual pins can be configured as CMOS or open drain outputs. Open drain pins require external pull-up resistors. For addressing information, refer to Tables 6 and 7.

Alternatively, each bit of Port B can be configured to provide a chip-select output signal from the PAD. PB0–PB7 can provide $\overline{CS0}$–$\overline{CS7}$, respectively. Each of the signals $\overline{CS0}$–$\overline{CS3}$ is comprised of four product terms. Thus, up to four ANDed expressions can be ORed while deriving any of these signals. Each of the signals $\overline{CS4}$–$\overline{CS7}$ is comprised of two product terms. Thus, up to two ANDed expressions can be ORed while deriving any of these signals.

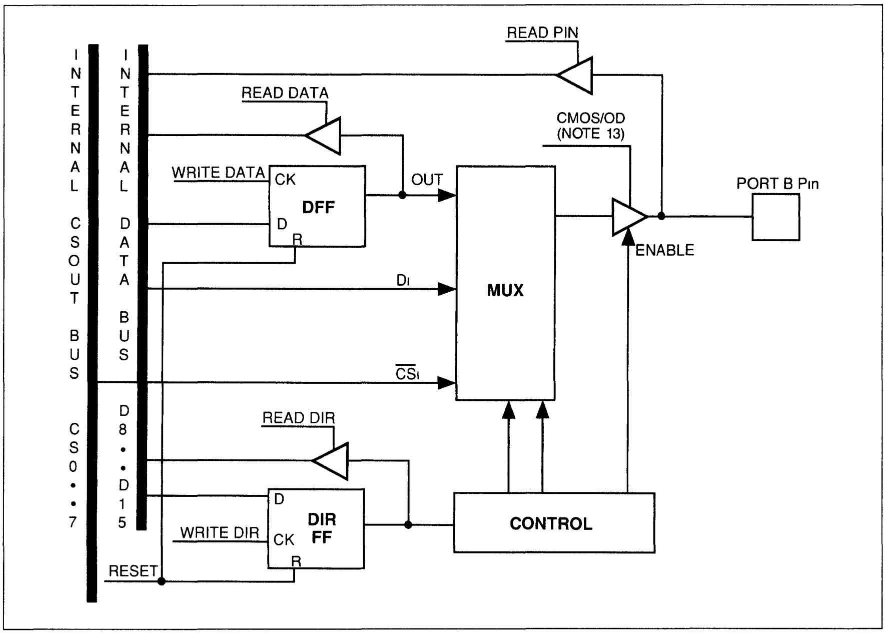
Port B in 16-Bit Non-Multiplexed Address/Data Mode

In this mode, Port B becomes the high-order data bus byte of the chip. When reading an internal PSD301 high-order data bus byte location, the data is presented on Port B pins. When writing to an internal PSD301 high-order data bus byte location, data present on Port B is written to that location. See Table 9.

Accessing the I/O Port Registers

Tables 6 and 7 show the offset values with the respect to the base address defined by the CSIOPORT. They let the user access the corresponding registers.

Figure 6. Port B Pin Structure



NOTE: 13 CMOS/OD determines whether the output is open drain or CMOS.

Table 6. I/O Port Addresses in an 8-bit Data Bus Mode

Register Name	Byte Size Access of the I/O Port Registers Offset from the CSIOPORT
Pin Register of Port A	+2 (accessible during read operation only)
Direction Register of Port A	+4
Data Register of Port A	+6
Pin Register of Port B	+3 (accessible during read operation only)
Direction Register of Port B	+5
Data Register of Port B	+7

Table 7. I/O Port Addresses in a 16-bit Data Bus Mode

Register Name	Word Size Access of the I/O Port Registers Offset from the CSIOPORT
Pin Register of Ports B and A	+2 (accessible during read operation only)
Direction Register of Ports B and A	+4
Data Register of Ports B and A	+6

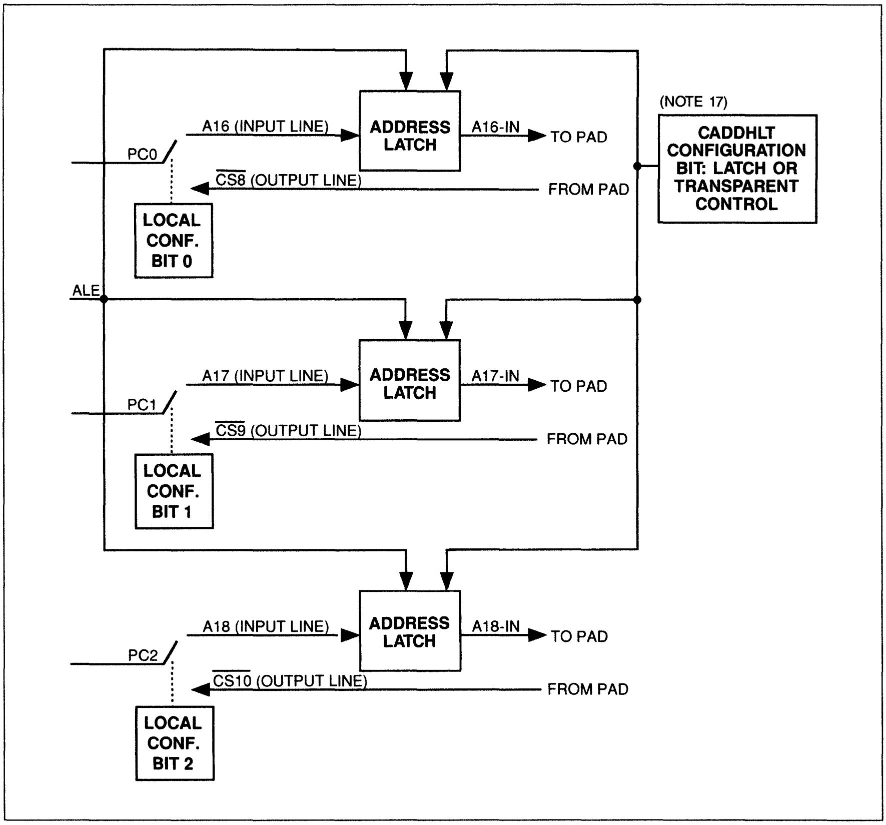
NOTES: 14. When the data bus width is 16, Port B registers can only be accessed if the $\overline{BHE}$ signal is low.
15. When accessing words, the high-order byte is connected to Port B, and the low-order byte is connected to Port A.
16. I/O Ports A and B are still byte-addressable, as shown in Table 6. For I/O Port B register access, $\overline{BHE}$ must be low.

Port C in All Modes

Each pin of Port C (shown in Figure 7) can be configured as an input or output from the PAD. As inputs, the pins are named A16–A18. Although the pins are given names of the high-order address bus, they can be used for any other logic inputs to the PAD. For example, A8–A10 can also be connected to those pins, reducing the boundaries of $\overline{CS0}$–$\overline{CS7}$ resolution to 256 bytes. Port C address latches can be programmed to latch the inputs by the trailing edge ALE or to be transparent.

Alternatively, PC0–PC2 can become $\overline{CS8}$–$\overline{CS10}$ outputs, respectively, providing the user with more external chip-select PAD outputs. Each of the signals $\overline{CS8}$–$\overline{CS10}$ is comprised of one product term.

Figure 7. Port C Structure



NOTE: 17. The CADDHLT configuration bit determines if A18–A16 are transparent via the latch, or if they must be latched by the trailing edge of the ALE strobe.

EPROM

The PSD301 has 256K bits of EPROM. Depending on the configuration of the data bus, the EPROM can be organized as 32K x 8 (8-bit data bus) or as 16K x 16 (16-bit data bus). The EPROM has 8 banks of memory. Each bank can be placed in any address location by programming the PAD. Bank0–Bank7 can be selected by PAD outputs ES0–ES7, respectively. The EPROM banks are organized as 4K x 8 (8-bit data bus) or as 2K x 16 (16-bit data bus).

SRAM

The PSD301 has 16K bits of SRAM. Depending on the configuration of the data bus, the SRAM organization can be 2K x 8 (8-bit data bus) or 1K x 16 (16-bit data bus). The SRAM is selected by the RS0 output of the PAD.

Control Signals

The PSD301 control signals are $\overline{\mathrm{WR}}/V_{PP}$ or R/$\overline{\mathrm{W}}$, $\overline{\mathrm{RD}}$/E, ALE, $\overline{\mathrm{BHE}}/\overline{\mathrm{PSEN}}$, Reset, and A19/$\overline{\mathrm{CSI}}$. Each of these signals can be configured to meet the output control signal requirements of various microcontrollers.

$\overline{\mathrm{WR}}/V_{PP}$ or R/$\overline{\mathrm{W}}$

In operational mode, this signal can be configured as $\overline{\mathrm{WR}}$ or R/$\overline{\mathrm{W}}$. As $\overline{\mathrm{WR}}$, all write operations to the PSD301 are activated by an active low signal on this pin. As R/$\overline{\mathrm{W}}$, the pin works with the E strobe of the $\overline{\mathrm{RD}}$/E pin. When R/$\overline{\mathrm{W}}$ is high, an active high signal on the $\overline{\mathrm{RD}}$/E pin performs a read operation. When R/$\overline{\mathrm{W}}$ is low, an active high signal on the $\overline{\mathrm{RD}}$/E pin performs a write operation.

$\overline{\mathrm{RD}}$/E

In operational mode, this signal can be configured as $\overline{\mathrm{RD}}$ or E. As $\overline{\mathrm{RD}}$, all read operations to the PSD301 are activated by an active low signal on this pin. As E, the pin works with the R/$\overline{\mathrm{W}}$ strobe of the $\overline{\mathrm{WR}}/V_{PP}$ or R/$\overline{\mathrm{W}}$ pin. When R/$\overline{\mathrm{W}}$ is high, an active high signal on the $\overline{\mathrm{RD}}$/E pin performs a read operation. When R/$\overline{\mathrm{W}}$ is low, an active high signal on the $\overline{\mathrm{RD}}$/E pin performs a write operation.

ALE or AS

ALE polarity is programmable. When programmed to be active high, a high on the pin causes the input address latches, Port A address latches, and Port C address latches to be transparent. The falling edge of ALE latches the information into the latches. When ALE is programmed to be active low, a low on the pin causes the input address latches, Port A address latches, Port C, and A19 address latches to be transparent. The rising edge of ALE latches the appropriate information into the latches. ALE is active only in the multiplexed modes.

$\overline{\mathrm{BHE}}/\overline{\mathrm{PSEN}}$

This pin's function depends on the PSD301 data bus width. If it is 8, the pin is $\overline{\mathrm{PSEN}}$; if it is 16, the pin is $\overline{\mathrm{BHE}}$. In 8-bit mode, the $\overline{\mathrm{PSEN}}$ function lets the user work with two address spaces: program memory and data memory (if $\overline{\mathrm{COMB}}$/SEP = 1). In this mode, an active low signal on the $\overline{\mathrm{PSEN}}$ pin causes the EPROM to be read. The SRAM and I/O ports read operation are done by $\overline{\mathrm{RD}}$ low (CRRWR = 0), or by E and R/$\overline{\mathrm{W}}$ high (CRRWR = 1).

Whenever a member of the 8031 family (or any other similar microcontroller) is used, the PSD301's $\overline{\mathrm{PSEN}}$ pin must be connected to the $\overline{\mathrm{PSEN}}$ pin of the microcontroller.

If COMB/SEP = 0, the address spaces of the program and the data are combined. In this configuration (except for the 8031-type case mentioned above), the $\overline{\mathrm{PSEN}}$ pin must be tied high to V_{CC}, and the EPROM, SRAM, and I/O ports are read by $\overline{\mathrm{RD}}$ low (CRRWR = 0), or by E and R/$\overline{\mathrm{W}}$ high (CRRWR = 1). See Figures 8 and 9.

Table 8. Signal Latch Status in All Operating Modes

Signal Name	Configuration Bits	Configuration Mode	Signal Latch Status

AD8/A8–
AD15/A15
	CDATA = 0,
	CADDRDAT = 0
		8-bit data, non-multiplexed	Transparent

	CDATA = 0,
	CADDRDAT = 1
		8-bit data, multiplexed	Transparent

	CDATA = 1,
	CADDRDAT = 0
		16-bit data, non-multiplexed	Transparent

	CDATA = 1,
	CADDRDAT = 1
		16-bit data, multiplexed	ALE dependent

AD0/A0– AD7/A7	CADDRDAT = 0	Non-multiplexed modes	Transparent
	CADDRDAT = 1	Multiplexed modes	ALE dependent
$\overline{\text{BHE}}$/$\overline{\text{PSEN}}$	CDATA = 0	8-bit data, $\overline{\text{PSEN}}$ is active	Transparent

	CDATA = 1,
	CADDRDAT = 0
		16-bit data, non-multiplexed mode, $\overline{\text{BHE}}$ is active	Transparent

	CDATA = 1,
	CADDRDAT = 1
		16-bit data, multiplexed mode, $\overline{\text{BHE}}$ is active	ALE dependent

A19 and PC2–PC0	CADDHLT = 0	A16–A19 can become logic inputs	Transparent
	CADDHLT = 1	A16–A19 can become multiplexed address lines	ALE dependent

Figure 8. Combined Address Space

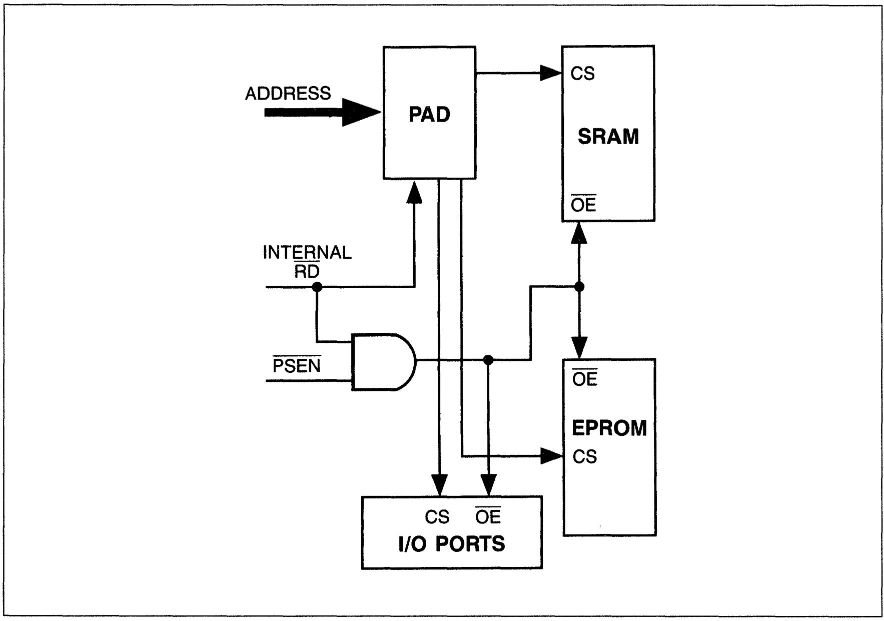
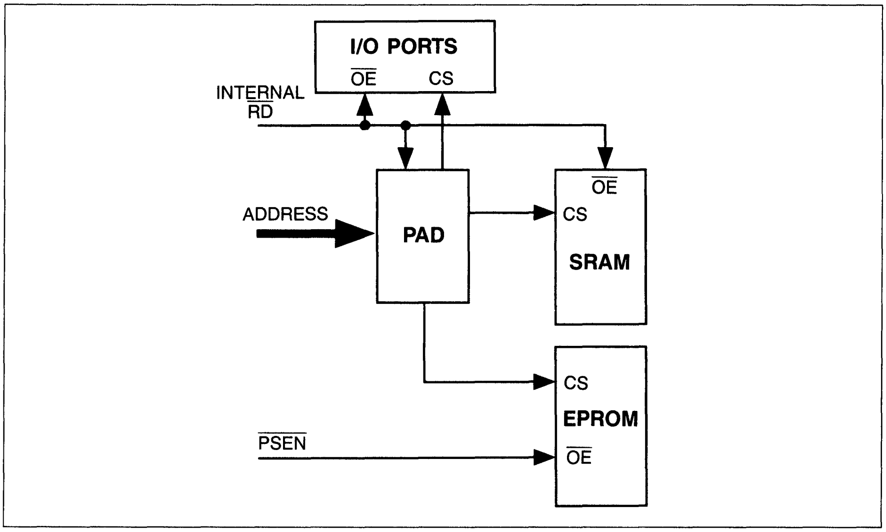


Figure 9. 8031-Type Separate Code and Data Address Spaces



In BHE mode, this pin enables accessing of the upper-half byte of the data bus. A low on this pin enables a write or read operation to be performed on the upper half of the data bus (see Table 9).

Table 9. High/Low Byte Selection Truth Table (in 16-Bit Configuration Only)

BHE	A_0	Operation
0	0	Whole Word
0	1	Upper Byte From/To Odd Address
1	0	Lower Byte From/To Even Address
1	1	None

RESET

This is an asynchronous input pin that clears and initializes the PSD301. Reset polarity is programmable (active low or active high). Whenever the PSD301 reset input is driven active for at least 100 ns, the chip is reset. The PSD301 must be reset before it can be used. Tables 10 and 11 indicate the state of the part during and after reset.

Table 10. Signal States During and After Reset

Signal	Configuration Mode	Condition
AD0/A0–AD15/A15	All	Input
PA0–PA7 (Port A)	I/O Tracking AD0/A0–AD7/A7 Address outputs A0–A7	Input Input Low
PB0–PB7 (Port B)	I/O CS7–CS0 CMOS outputs CS7–CS0 open drain outputs	Input High Tri-stated
PC0–PC2 (Port C)	Address inputs A16–A18 CS8–CS10 CMOS outputs	Input High

Table 11. Internal States During and After Reset

Component	Signals	Contents
PAD	$\overline{\text{CS0}}$–$\overline{\text{CS10}}$	All = 1[18]
	CSADIN, CSADOUT1, CSADOUT2, CSIOPORT, RS0, ES0–ES7	All = 0[18]
Data register A	n/a	0
Direction register A	n/a	0
Data register B	n/a	0
Direction register B	n/a	0

NOTE: 18. All PAD outputs are in a non-active state.

A19/$\overline{\text{CSI}}$

When configured as $\overline{\text{CSI}}$, a high on this pin deselects, and powers down, the chip. A low on this pin puts the chip in normal operational mode. For PSD301 states during the power-down mode, see Tables 12 and 13.

Table 12. Signal States During Power-Down Mode

Signal	Configuration Mode	Condition
AD0/A0–AD15/A15	All	Input
PA0–PA7	I/O	Unchanged
	Tracking AD0/A0–AD7/A7	Input
	Address outputs A0–A7	All 1's
PB0–PB7	I/O	Unchanged
	$\overline{\text{CS7}}$–$\overline{\text{CS0}}$ CMOS outputs	All 1's
	$\overline{\text{CS7}}$–$\overline{\text{CS0}}$ open drain outputs	Tri-stated
PC0–PC2	Address inputs A16–A18	Input
	$\overline{\text{CS8}}$–$\overline{\text{CS10}}$ CMOS outputs	All 1's

Table 13. Internal States During Power-Down

Component	Signals	Contents
PAD	$\overline{\text{CS0}}$–$\overline{\text{CS10}}$	All 1's (deselected)
	CSADIN, CSADOUT1, CSADOUT2, CSIOPORT, RS0, ES0–ES7	All 0's (deselected)
Data register A	n/a	All unchanged
Direction register A	n/a	
Data register B	n/a	
Direction register B	n/a	

In A19 mode, the pin is an additional input to the PAD. It can be used as a high-order address line or as a general-purpose logic input. A19 can be configured as ALE dependent or as transparent input (see Table 8). In this mode, the chip is always enabled.

System Applications

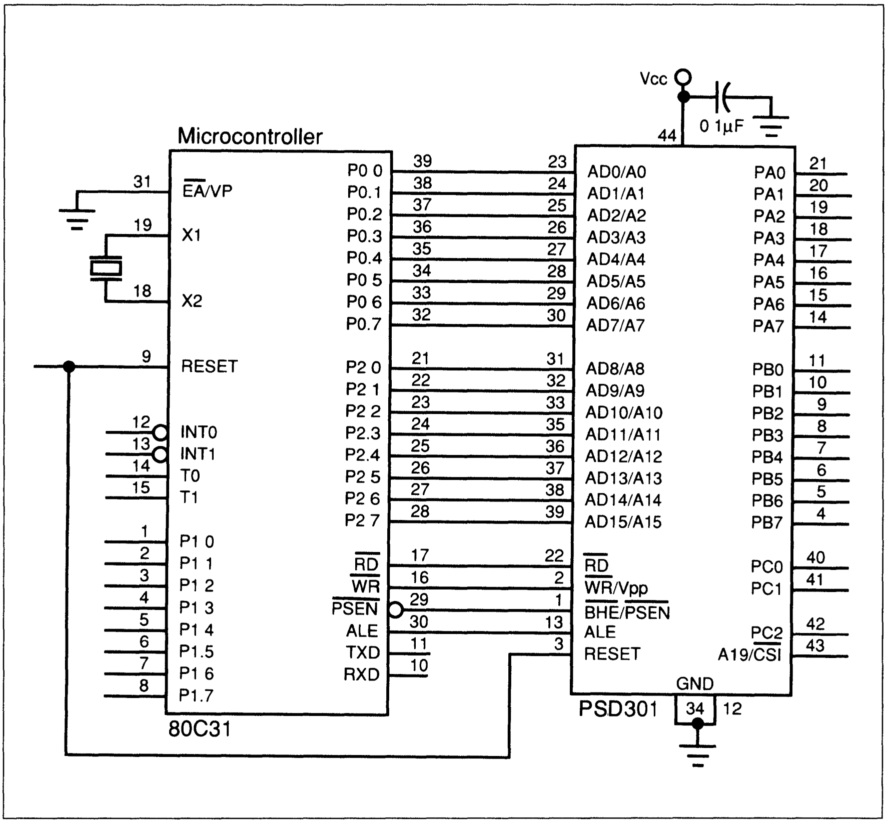
In Figure 10, the PSD301 is configured to interface with Intel's 80C31, which is a 16-bit address/8-bit data bus microcontroller. Its data bus is multiplexed with the low-order address byte. The 80C31 uses signals $\overline{\text{RD}}$ to read from data memory and $\overline{\text{PSEN}}$ to read from code memory. It uses $\overline{\text{WR}}$ to write into the data memory. It also uses active high reset and ALE signals. The rest of the configuration bits as well as the unconnected signals (not shown) are application specific and, thus, user dependent.

The configuration bits for Figure 10 are:

CRESET	1
CALE	0
CDATA	0
$\overline{\text{CADDRDAT}}$	1
COMB/SEP	0 or 1 (both valid)
CRRWR	0

All other configuration bits may vary according to the application requirements.

Figure 10. PSD301 Interface with Intel's 80C31



System Applications (Cont.)

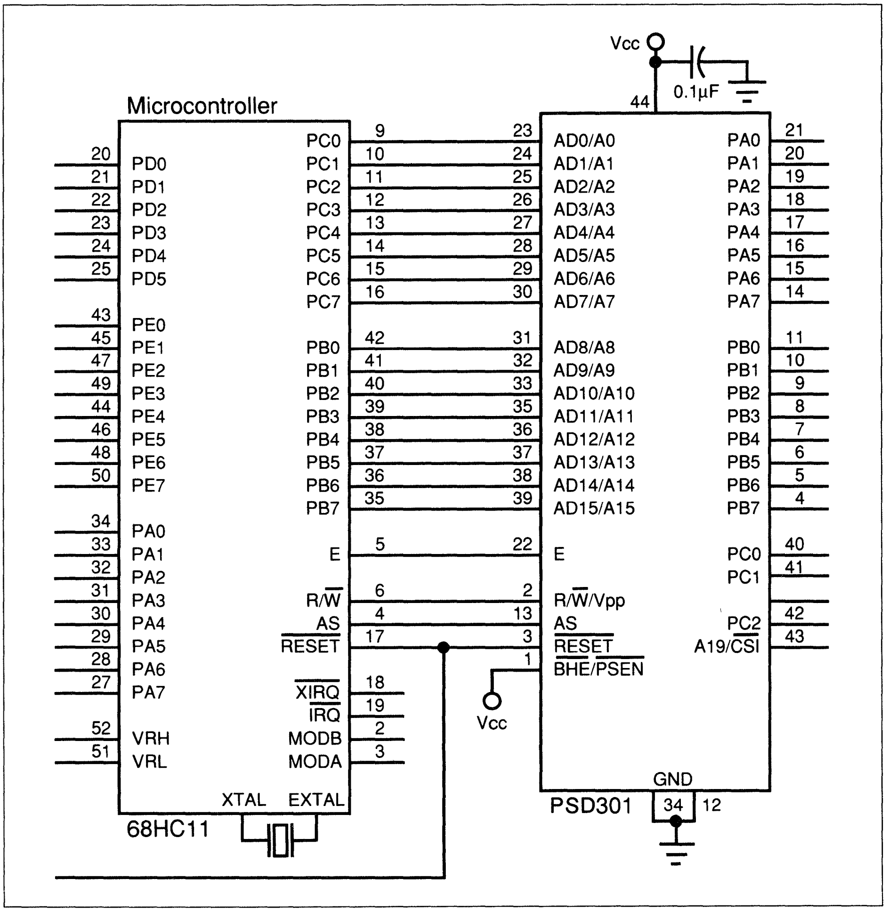
In Figure 11, the PSD301 is configured to interface with Motorola's 68HC11, which is a 16-bit address/8-bit data bus microcontroller. Its data bus is multiplexed with the low-order address byte. The 68HC11 uses E and R/$\overline{W}$ signals to derive the read and write strobes. It uses the term AS (address strobe) for the address latch pulse. RESET is an active low signal. The rest of the configuration bits as well as the unconnected signals (not shown) are application specific and, thus, user dependent.

The configuration bits for Figure 11 are:

CRESET	0
CALE	0
CDATA	0
CADDRDAT	1
COMB/SEP	0
CRRWR	1

All other configuration bits may vary according to the application requirements.

Figure 11. PSD301 Interface with Motorola's 68HC11



In Figure 12, the PSD301 is configured to work directly with Intel's 80C196KB microcontroller, which is a 16-bit address/16-bit data bus processor. Address and data lines multiplexed. In the example shown, all configuration bits are set. The PSD301 is configured to use PC0, PC1, PC2, and $\overline{\text{CSI}}$/A19 as A16, A17, A18, and A19 inputs, respectively. These signals are independent of the ALE pulse (latch-transparent). They are used as four general-purpose logic inputs that take part

System Applications (Cont.)

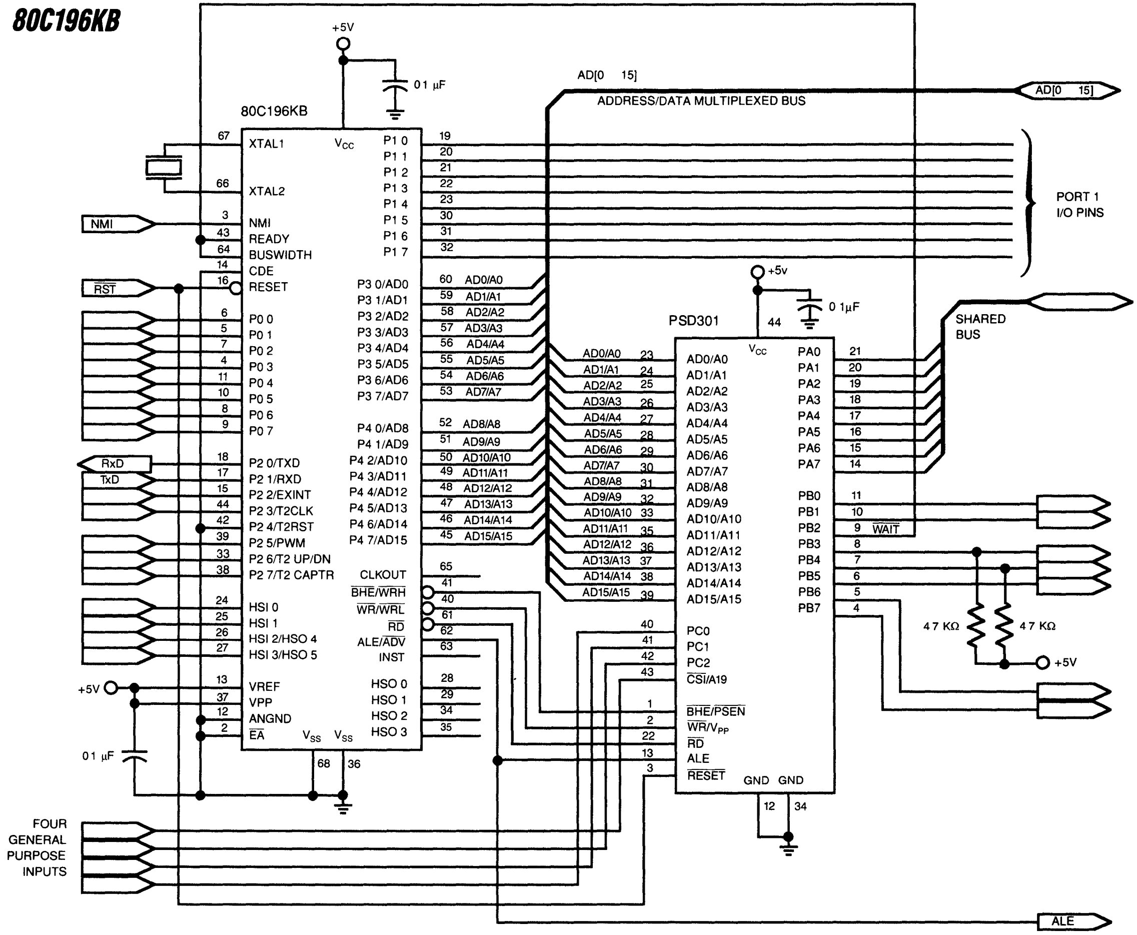
in the PAD equations implementation.

Port A is configured to work in the special track mode, in which (for certain conditions) PA0–PA7 tracks lines AD0/A0–AD7/A7. Port B is configured to generate $\overline{CS0}$–$\overline{CS7}$. In this example, PB2 serves as a $\overline{WAIT}$ signal that slows down the 80C196KB during the access of external peripherals. These 8-bit wide peripherals are connected to the shared bus of Port A. The $\overline{WAIT}$ signal also drives the buswidth input of the microcontroller, so that every external peripheral cycle becomes an 8-bit data bus cycle. PB3 and PB4 are open-drain output signals; thus, they are pulled up externally.

The configuration bits for Figure 12 are:

CRESET	0
CALE	0
CDATA	1
CADDRDAT	1
CPAF1	Don't care
CPAF2	1
CA19/$\overline{CSI}$	1
$\overline{CRRWR}$	0
$\overline{COMB}$/SEP	0
CADDHLT	0
CSECURITY	Don't care
CPCF2, CPCF1, CPCF0	0, 0, 0
CPACOD7–CPACOD0	00H
CPBF7–CPBF0	00H
CPBCOD7–CPBCOD0	18H

Figure 12. PSD301 Interface with Intel's 80C196KB



Absolute Maximum Ratings

Symbol	Parameter	Conditions	Min	Max	Unit
T_{STG}	Storage temperature		−65	+150	°C
	Voltage on Any Pin	With Respect to GND	−0.6	+7	V
V_{PP}	Programming Supply Voltage	With Respect to GND	−0.6	14	V
V_{CC}	Supply Voltage	With Respect to GND	−0.6	+7	V
	ESD Protection			>2000	V

NOTE: 19. Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect device reliability.

Operating Range

Range	Temperature	V_{CC}	Tolerance
Commercial	0°C to +70°C	+5V	±5% or ±10%
Industrial	−40°C to +85°C	+5V	±10%
Military	−55°C to +125°C	+5V	±10%

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{CC}	Supply Voltage		4.5	5	5.5	V
V_{IH}	High Level Input Voltage	V_{CC} = 4.5V to 5.5V	2			V
V_{IL}	Low Level Input Voltage	V_{CC} = 4.5V to 5.5V	0		0.8	V

DC Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Units
V_{OL}	Output Low Voltage	I_{OL} = 20 µA, V_{CC} = 5.5V			0.01	0.1	V
		I_{OL} = 8 mA, V_{CC} = 5.5V			0.15	0.45	
V_{OH}	Output High Voltage	I_{OH} = −20 µA, V_{CC} = 4.5V		4.4	4.49		V
		I_{OH} = −2 mA, V_{CC} = 4.5V		2.4	3.9		
I_{SB1}	V_{CC} Standby Current CMOS	Notes 20 and 22	Comm'l		80	150	µA
			Military		120	250	
I_{SB2}	V_{CC} Standby Current TTL	Notes 21 and 22	Comm'l		0.8	1.5	mA
			Military		1.0	2	
I_{CC1}	Active Current (CMOS) No Blocks Selected	Notes 20 and 23	Comm'l		35	55	mA
			Military		40	65	
I_{CC2}	Active Current (CMOS) EPROM Block Selected	Notes 20 and 23	Comm'l		35	55	mA
			Military		40	65	

DC Characteristics (Cont.)

Symbol	Parameter	Conditions		Min	Typ	Max	Units
I_{CC3}	Active Current (CMOS) SRAM Block Selected	Notes 20 and 23	Comm'l		65	105	mA
			Military		75	120	
I_{CC4}	Active Current (TTL) No Blocks Selected	Notes 21 and 23	Comm'l		50	80	mA
			Military		60	95	
I_{CC5}	Active Current (TTL) EPROM Block Selected	Notes 21 and 23	Comm'l		50	80	mA
			Military		60	95	
I_{CC6}	Active Current (TTL) SRAM Block Selected	Notes 21 and 23	Comm'l		80	130	mA
			Military		90	150	
I_{LI}	Input Load Current	V_{IN} = 5.5V or GND		−1	±0.1	1	µA
I_{LO}	Output Leakage Current	V_{OUT} = 5.5V or GND		−10	5	10	µA

NOTES: 20 CMOS inputs. GND ± 0 3V or V_{CC} ± 0.3V
21. TTL inputs· $V_{IL} \leqslant 0.8V$, $V_{IH} \geqslant 2.0V$
22. AC power component is 1 5 mA/MHz (power = AC + DC)
23 AC power component is 3 5 mA/MHz (power = AC + DC).

AC Characteristics

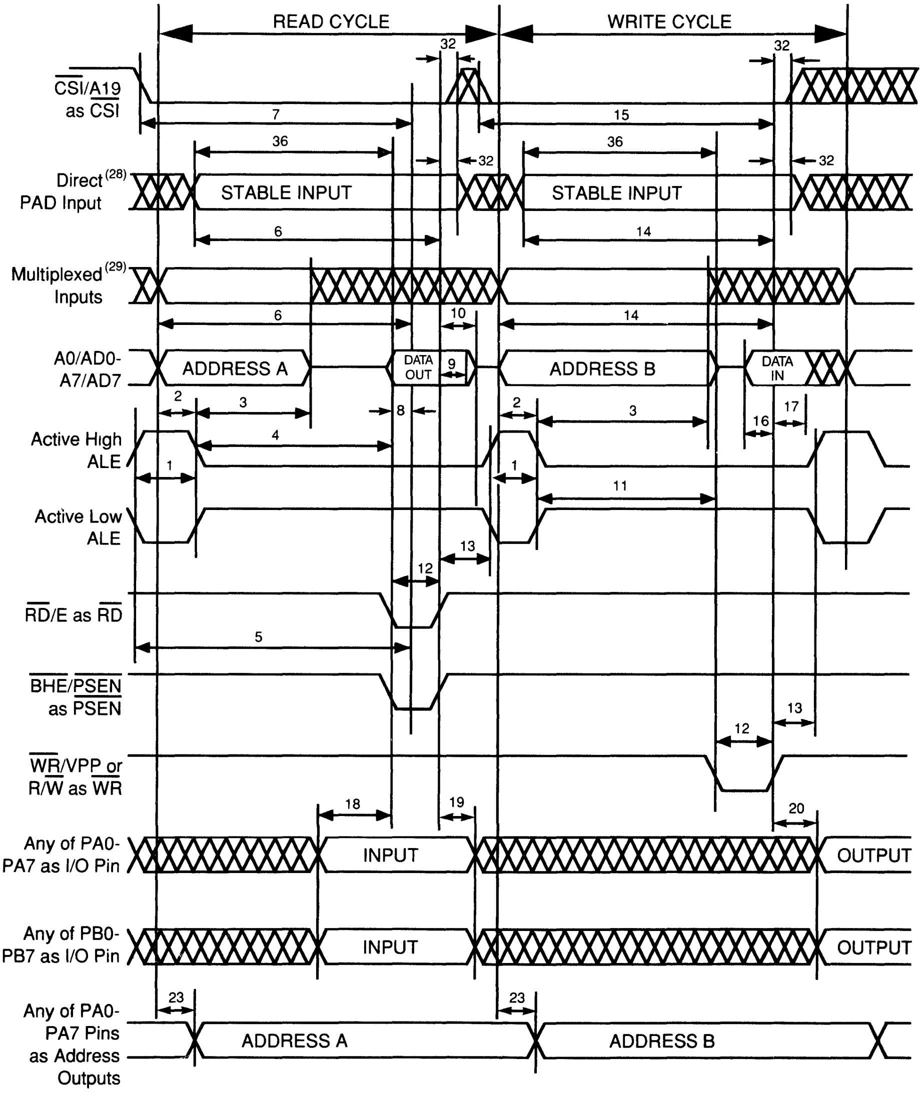
Symbol	Parameter	-12		-15		-20		Units
		Min	Max	Min	Max	Min	Max	
T1	ALE or AS pulse width	30		40		50		ns
T2	Address set-up time	15		20		25		ns
T3	Address hold time	10		15		20		ns
T4	ALE or AS trailing edge to leading edge of Read	12		15		20		ns
T5	ALE or AS leading edge to data valid		140		170		220	ns
T6	Address valid to data valid		120		150		200	ns
T7	$\overline{\text{CSI}}$ active to data valid		130		160		210	ns
T8	Leading edge of Read to data valid		40		55		60	ns
T9	Read data hold time	0		0		0		ns
T10	Trailing edge of Read to data high-Z		35		40		45	ns
T11	Trailing edge of ALE or AS to leading edge of Write	12		15		20		ns
T12	$\overline{\text{RD}}$, E, $\overline{\text{PSEN}}$, or $\overline{\text{WR}}$ pulse width	45		60		75		ns
T13	Trailing edge of Write or Read to leading edge of ALE or AS	20		30		40		ns
T14	Address valid to trailing edge of Write	120		150		210		ns
T15	$\overline{\text{CSI}}$ active to trailing edge of Write	130		160		200		ns
T16	Write data set-up time	20		30		40		ns
T17	Write data hold time	10		15		20		ns
T18	Port input set-up time	10		15		20		ns
T19	Port input hold time	10		20		30		ns

AC Characteristics (Cont.)

Symbol	Parameter	-12		-15		-20		Units
		Min	Max	Min	Max	Min	Max	
T20	Trailing edge of Write to port output valid	40		50		60		ns
T21	ADi[24] or control[27] to $\overline{\text{CSOi}}$[25] valid	20	35	25	45	30	55	ns
T22	ADi[24] or control[27] to $\overline{\text{CSOi}}$[25] invalid	20	35	25	45	30	55	ns
T23	Track mode address propagation delay: • CSADOUT1 already true or:		20		30		40	ns
	• CSADOUT1 becomes true during ALE or AS		40		50		60	ns
T24	Track mode address hold time	10		15		20		ns
T25	Track mode Read propagation delay		20		30		40	ns
T26	Track mode Read hold time	10	20	15	30	20	40	ns
T27	Track mode Write cycle data propagation delay		20		30		40	ns
T28	Track mode Write cycle write to data propagation delay	20	40	25	50	30	60	ns
T29	Hold time of Port A valid during write to $\overline{\text{CSOi}}$ trailing edge	2		4		6		ns
T30	$\overline{\text{CSI}}$ active to $\overline{\text{CSOi}}$[25] active	25	45	30	55	35	65	ns
T31	$\overline{\text{CSI}}$ inactive to $\overline{\text{CSOi}}$[25] inactive	25	45	25	55	35	65	ns
T32	Control[27] signal inactive to data invalid	5		10		15		ns
T33	R/$\overline{\text{W}}$ active to E high	20		30		40		ns
T34	E low to R/$\overline{\text{W}}$ inactive	20		30		40		ns
T35	AS inactive to E high	15		20		25		ns
T36	Direct PAD input[26] stable to leading edge of $\overline{\text{RD}}$, $\overline{\text{WR}}$, or E	15		20		25		ns

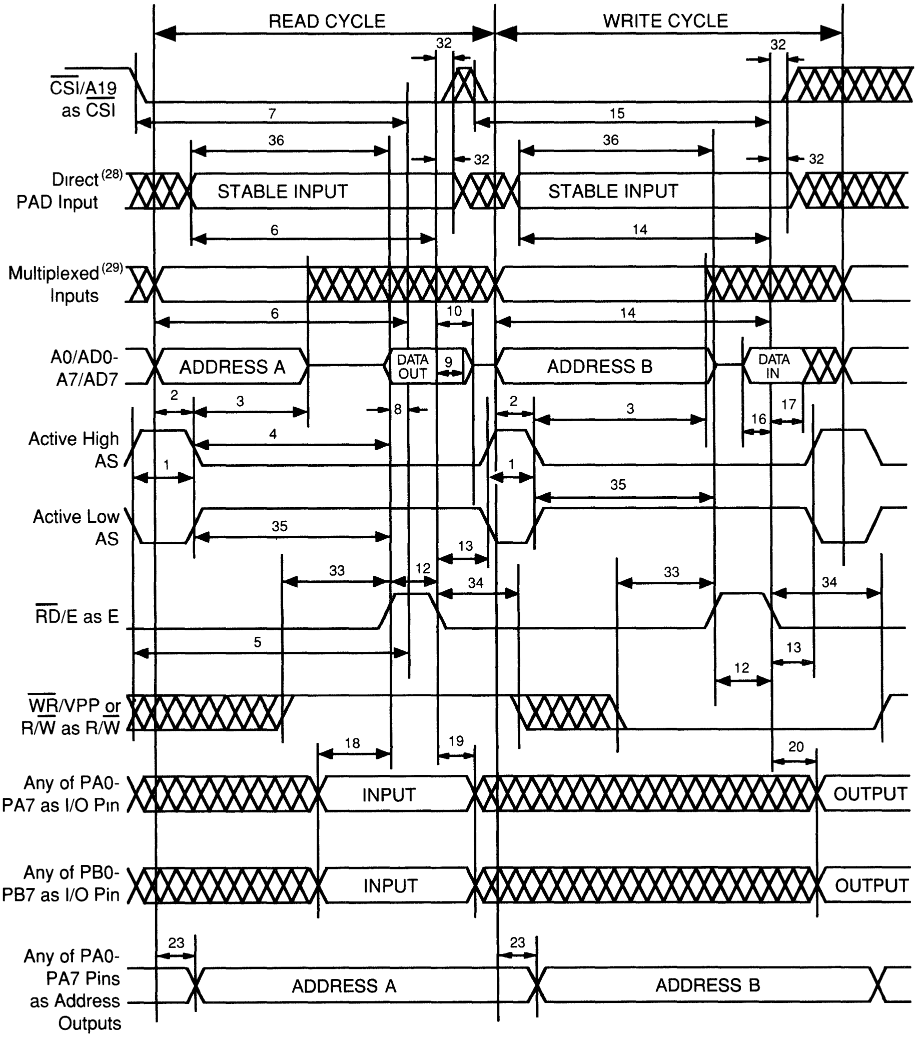
NOTES:
[24] ADi = any address line.
[25] $\overline{\text{CSOi}}$ = any of the chip-select output signals coming through Port B ($\overline{\text{CS0}}$–$\overline{\text{CS7}}$) or through Port C ($\overline{\text{CS8}}$–$\overline{\text{CS10}}$).
[26] Direct PAD input = any of the following direct PAD input lines: $\overline{\text{CSI}}$/A19 as transparent A19, $\overline{\text{RD}}$/E, $\overline{\text{WR}}$ or R/$\overline{\text{W}}$, transparent PC0–PC2, ALE (or AS).
[27] Control signals $\overline{\text{RD}}$/E or $\overline{\text{WR}}$ or R/$\overline{\text{W}}$

Figure 13. Timing of 8-Bit Multiplexed Address/Data Bus, CRRWR = 0



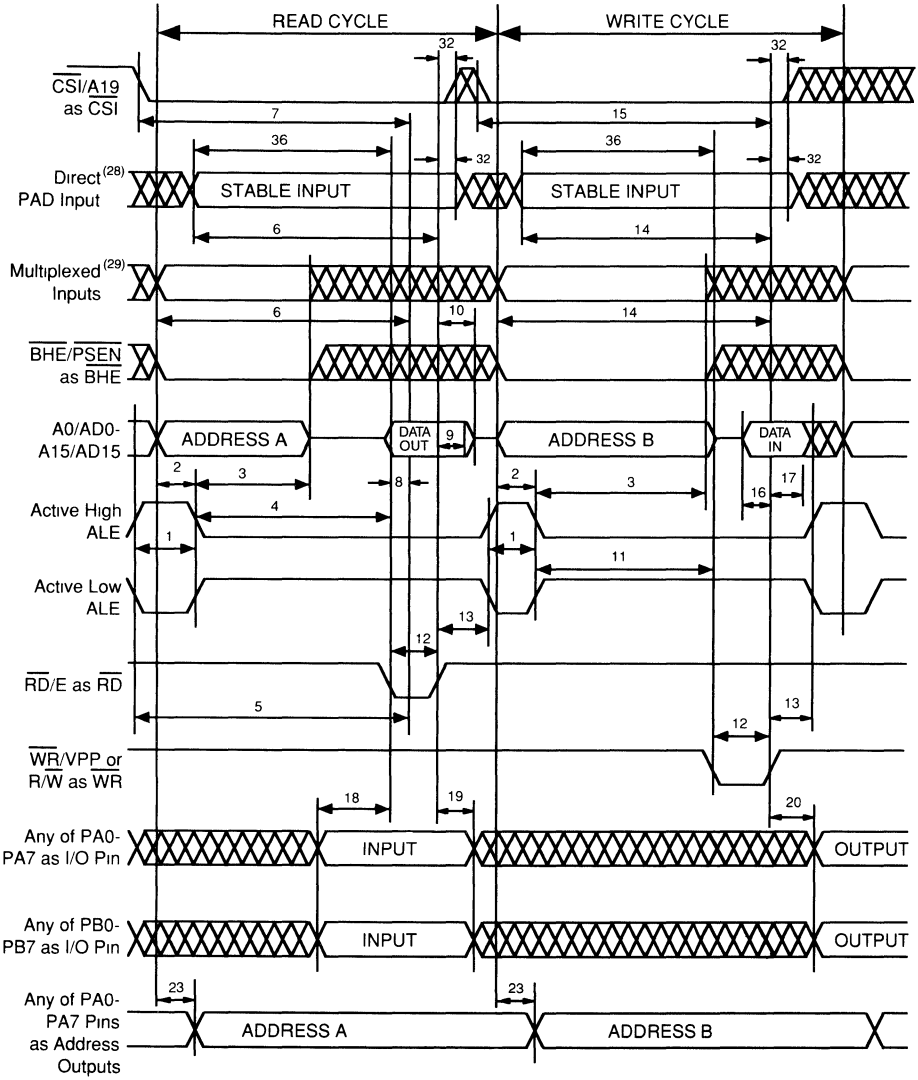
See referenced notes on page 2-58.

Figure 14. Timing of 8-Bit Multiplexed Address/Data Bus, CRRWR = 1



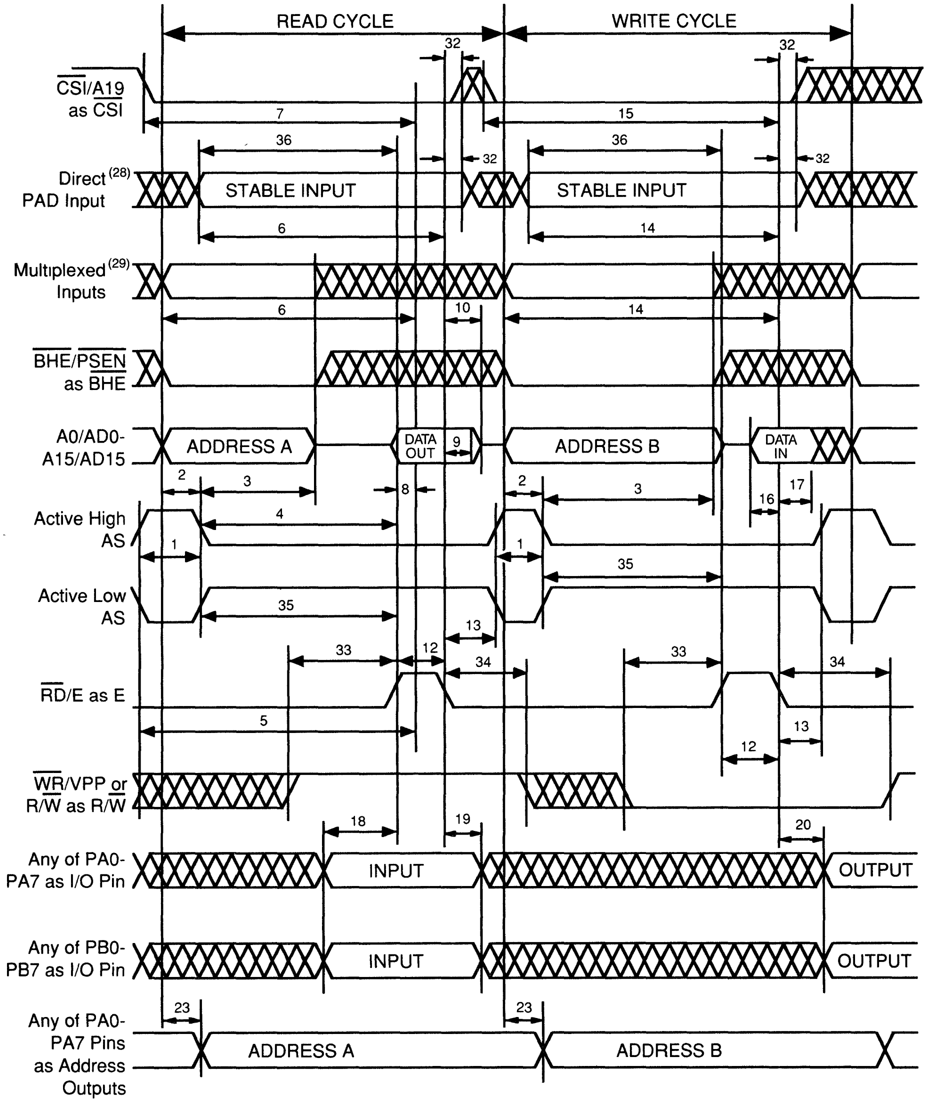
See referenced notes on page 2-58.

Figure 15. Timing of 16-Bit Multiplexed Address/Data Bus, CRRWR = 0



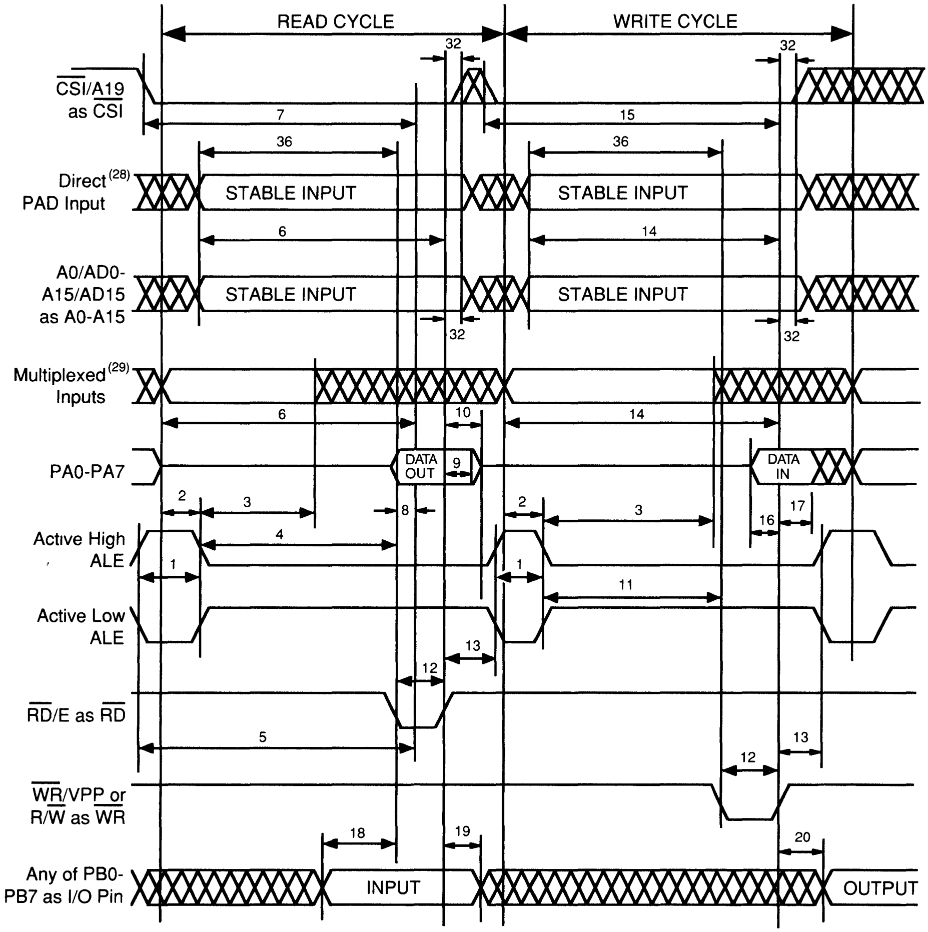
See referenced notes on page 2-58.

Figure 16. Timing of 16-Bit Multiplexed Address/Data Bus, CRRWR = 1



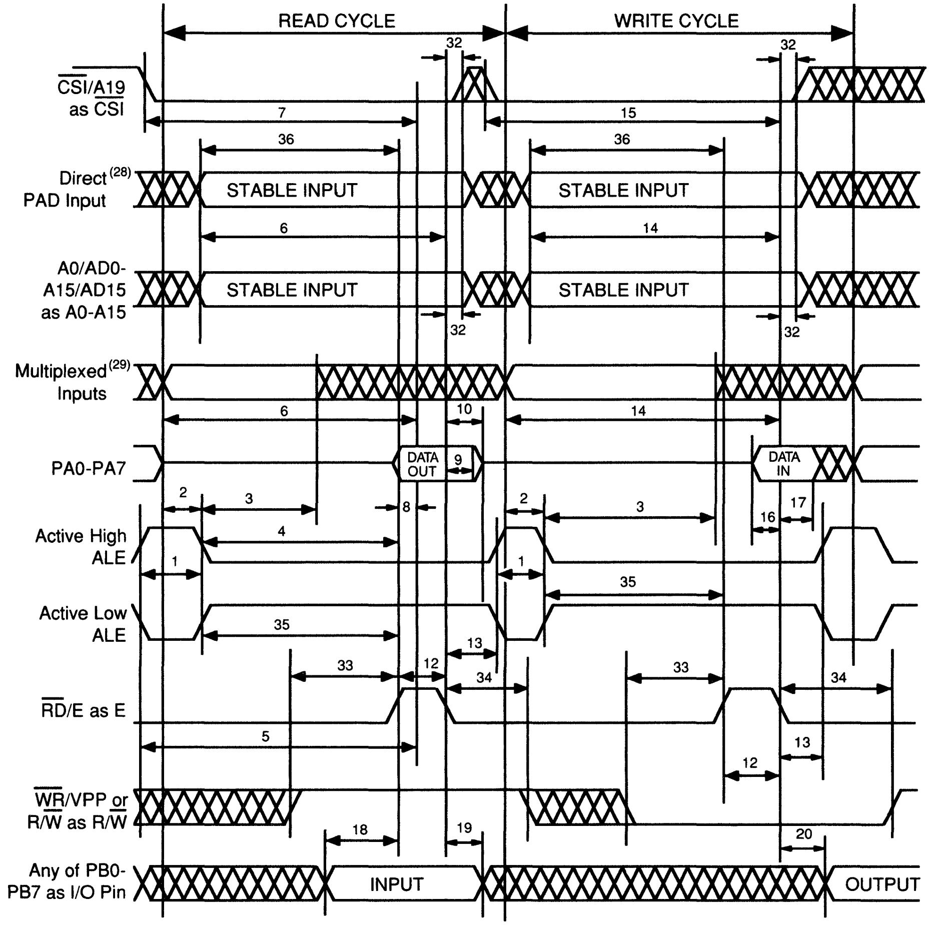
See referenced notes on page 2-58

Figure 17. Timing of 8-Bit Data, Non-Multiplexed Address/Data Bus, CRRWR = 0



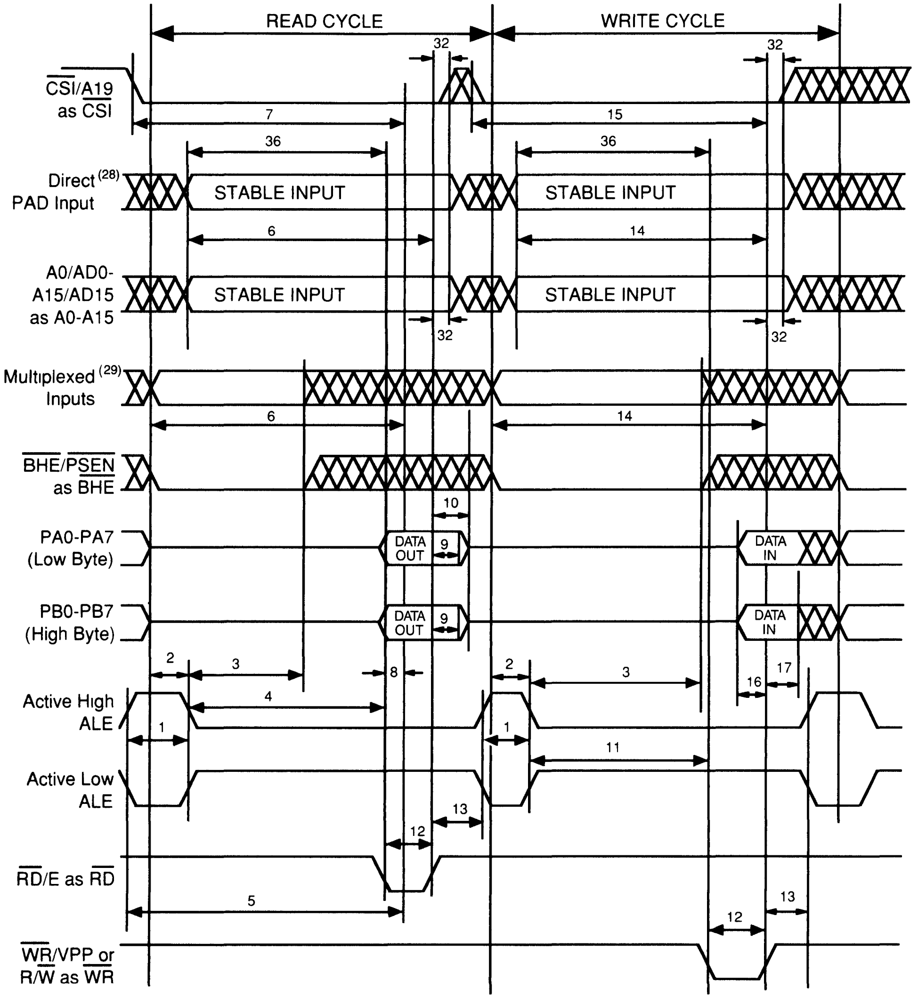
See referenced notes on page 2-58.

Figure 18. Timing of 8-Bit Data, Non-Multiplexed Address/Data Bus, CRRWR = 1



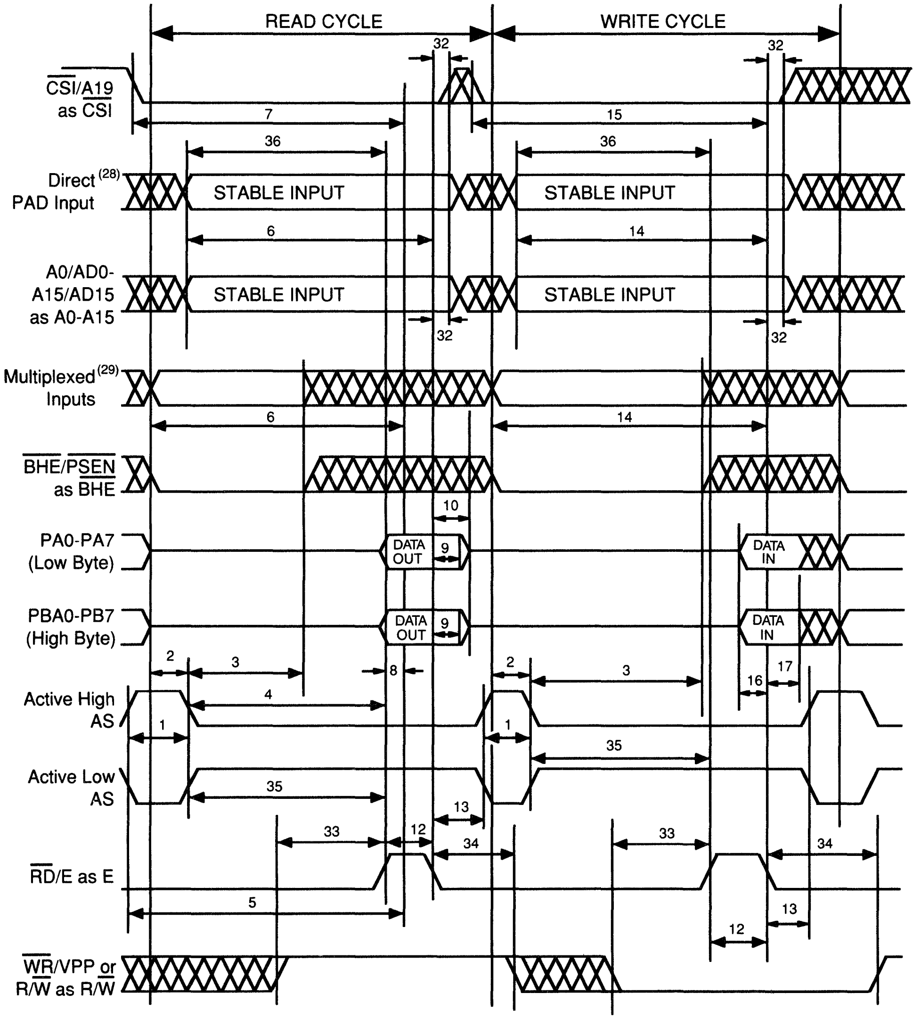
See referenced notes on page 2-58.

Figure 19. Timing of 16-Bit Non-Multiplexed Address/Data Bus, CRRWR = 0



See referenced notes on page 2-58.

Figure 20. Timing of 16-Bit Non-Multiplexed Address/Data Bus, CRRWR = 1



See referenced notes on page 2-58.

Figure 21. Chip-Select Output Timing

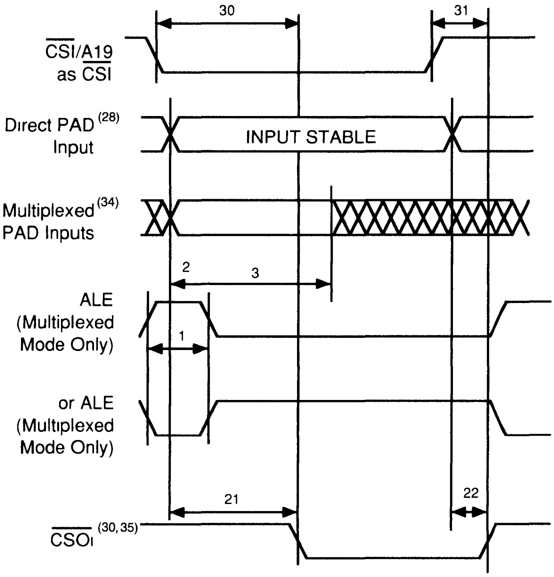
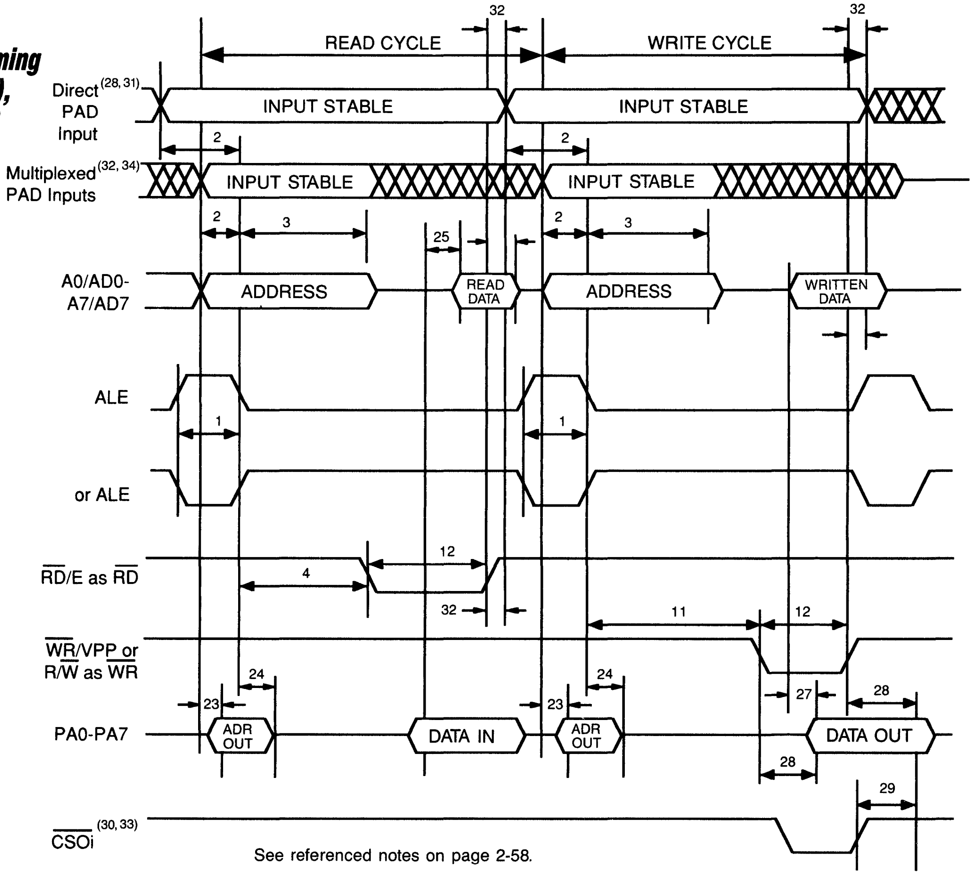
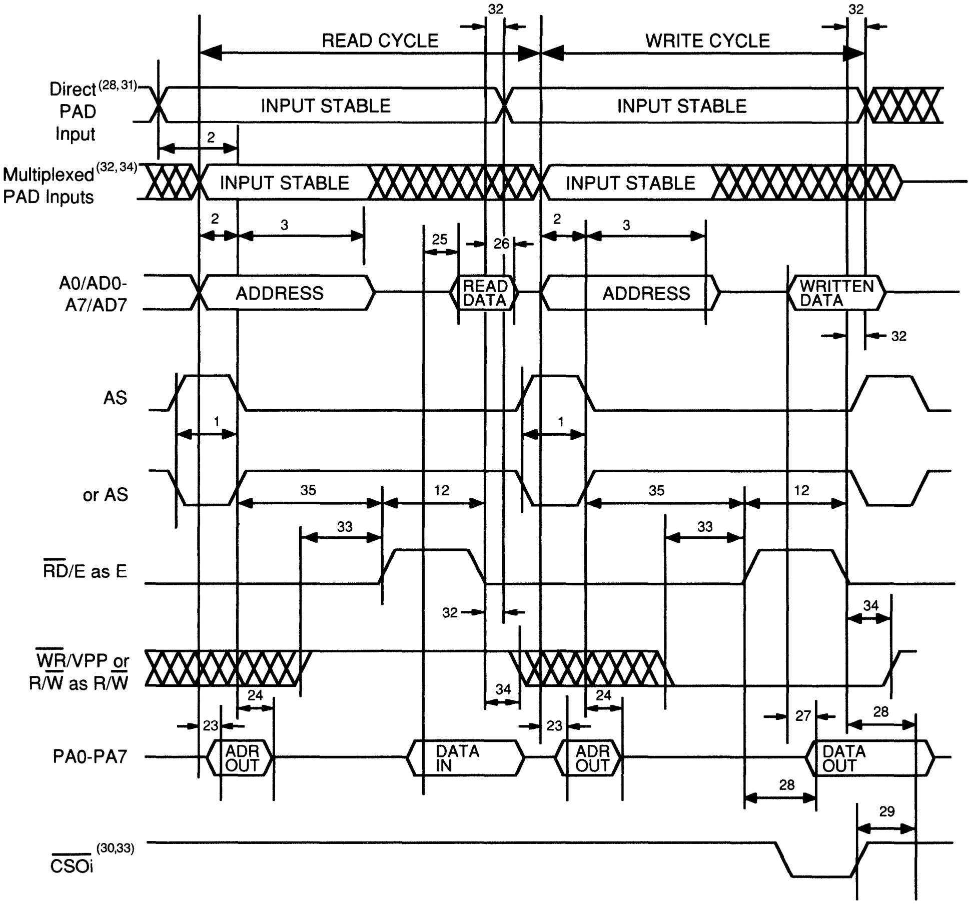

30
31
CSI/A19 as CSI
Direct PAD(28) Input
INPUT STABLE
Multiplexed(34) PAD Inputs
2
3
ALE (Multiplexed Mode Only)
1
or ALE (Multiplexed Mode Only)
21
22
CSOi(30,35)

Figure 22. Port A as AD0-AD7 Timing (Track Mode), CRRWR = 0



See referenced notes on page 2-58.

Figure 23. Port A as AD0-AD7 Timing (Track Mode), CRRWR = 1



NOTES:

28. Direct PAD input = any of the following direct PAD input lines: $\overline{\text{CSI}}$/A19 as transparent A19, $\overline{\text{RD}}$/E, $\overline{\text{WR}}$ or R/$\overline{\text{W}}$, transparent PC0–PC2, ALE in non-multiplexed modes.
29. Multiplexed inputs: any of the following inputs that are latched by the ALE (or AS): A0/AD0–A15/AD15, $\overline{\text{CSI}}$/A19 as ALE dependent A19, ALE dependent PC0–PC2.
30. $\overline{\text{CSOi}}$ = any of the chip-select output signals coming through Port B ($\overline{\text{CS0}}$–$\overline{\text{CS7}}$) or through Port C ($\overline{\text{CS8}}$–$\overline{\text{CS10}}$).
31. CSADOUT1, which internally enables the address transfer to Port A, should be derived only from direct PAD input signals, otherwise the address propagation delay is slowed down.
32 CSADIN and CSADOUT2, which internally enable the data-in or data-out transfers, respectively, can be derived from any combination of direct PAD inputs and multiplexed PAD inputs.
33. The write operation signals are included in the $\overline{\text{CSOi}}$ expression.
34. Multiplexed PAD inputs: any of the following PAD inputs that are latched by the ALE (or AS) in the multiplexed modes: A11/AD11–A15/AD15, $\overline{\text{CSI}}$/A19 as ALE dependent A19, ALE dependent PC0–PC2.
35. $\overline{\text{CSOi}}$ product terms can include any of the PAD input signals shown in Figure 3, except for reset and $\overline{\text{CSI}}$.

Table 14. Pin Capacitance[36]

T_A = 25°C, f = 1 MHz

Symbol	Parameter	Conditions	Typical[37]	Max	Units
C_{IN}	Capacitance (for input pins only)	V_{IN} = 0V	4	6	pF
C_{OUT}	Capacitance (for input/output pins)	V_{OUT} = 0V	8	12	pF
C_{VPP}	Capacitance (for $\overline{WR}/V_{PP}$ or $R/\overline{W}/V_{PP}$)	V_{PP} = 0V	18	25	pF

NOTES: 36. This parameter is only sampled and is not 100% tested.
37. Typical values are for T_A = 25°C and nominal supply voltages

Figure 24. AC Testing Input/Output Waveform

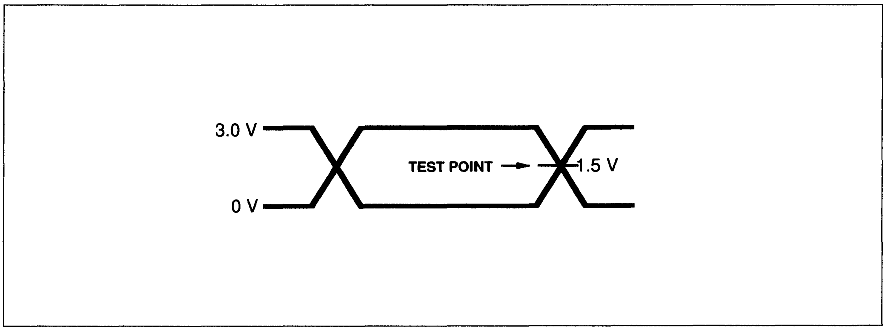
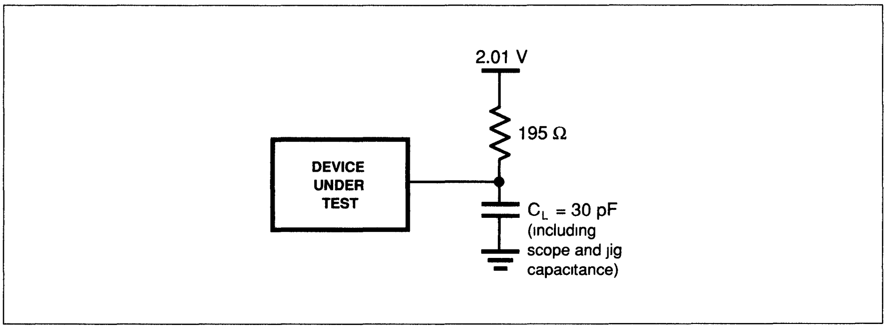


Figure 25. AC Testing Load Circuit



Erasure and Programming

To clear all locations of their programmed contents, expose the device to ultra-violet light source. A dosage of 15 W-second/cm^2 is required. This dosage can be obtained with exposure to a wavelength of 2537Å and intensity of 1200 μW/cm^2 for 15 to 20 minutes. The device should be about 1 inch from the source, and all filters should be removed from the UV light source prior to erasure.

The PSD301 and similar devices will erase with light sources having wavelengths shorter than 4000Å. Although the erasure times will be much longer than with UV sources at 2537Å, exposure to fluorescent light and sunlight eventually erases the device. For maximum system reliability, these sources should be avoided. If used in such an environment, the package windows should be covered by an opaque substance.

Upon delivery from WSI, or after each erasure, the PSD301 device has all bits in the PAD and EPROM in the "1" or high state. The configuration bits are in the "0" or low state. The code, configuration, and PAD MAP data are loaded through the procedure of programming

Information for programming the device is available directly from WSI. Please contact your local sales representative.

PSD301 Pin Assignments

44-Pin PLDCC/ CLDCC Package	Name	44-Pin CPGA Package
1	$\overline{BHE}$/$\overline{PSEN}$	A_5
2	$\overline{WR}$/V_{PP} or R/$\overline{W}$	A_4
3	RESET	B_4
4	PB7	A_3
5	PB6	B_3
6	PB5	A_2
7	PB4	B_2
8	PB3	B_1
9	PB2	C_2
10	PB1	C_1
11	PB0	D_2
12	GND	D_1
13	ALE or AS	E_1
14	PA7	E_2
15	PA6	F_1
16	PA5	F_2
17	PA4	G_1
18	PA3	G_2
19	PA2	H_2
20	PA1	G_3
21	PA0	H_3
22	$\overline{RD}$/E	G_4

44-Pin PLDCC/ CLDCC Package	Name	44-Pin CPGA Package
23	AD0/A0	H_4
24	AD1/A1	H_5
25	AD2/A2	G_5
26	AD3/A3	H_6
27	AD4/A4	G_6
28	AD5/A5	H_7
29	AD6/A6	G_7
30	AD7/A7	G_8
31	AD8/A8	F_7
32	AD9/A9	F_8
33	AD10/A10	E_7
34	GND	E_8
35	AD11/A11	D_8
36	AD12/A12	D_7
37	AD13/A13	C_8
38	AD14/A14	C_7
39	AD15/A15	B_8
40	PC0	B_7
41	PC1	A_7
42	PC2	B_6
43	A19/$\overline{CSI}$	A_6
44	V_{CC}	B_5

Ordering Information

Part Number	Speed (ns)	Package Type	Package Drawing	Operating Temperature Range	WSI Manufacturing Procedure
PSD301-12J	120	44-pin PLDCC	J2	Comm'l	Standard
PSD301-12L	120	44-pin CLDCC	L4	Comm'l	Standard
PSD301-12X	120	44-pin CPGA	X2	Comm'l	Standard
PSD301-15J	150	44-pin PLDCC	J2	Comm'l	Standard
PSD301-15L	150	44-pin CLDCC	L4	Comm'l	Standard
PSD301-15LM	150	44-pin CLDCC	L4	Military	Standard
PSD301-15LM	150	44-pin CLDCC	L4	Military	MIL-STD-883C
PSD301-15X	150	44-pin CPGA	X2	Comm'l	Standard
PSD301-15XM	150	44-pin CPGA	X2	Military	Standard
PSD301-15XMB	150	44-pin CPGA	X2	Military	MIL-STD-883C
PSD301-20J	200	44-pin PLDCC	J2	Comm'l	Standard
PSD301-20L	200	44-pin CLDCC	L4	Comm'l	Standard
PSD301-20LM	200	44-pin CLDCC	L4	Military	Standard
PSD301-20LMB	200	44-pin CLDCC	L4	Military	MIL-STD-883C
PSD301-20X	200	44-pin CPGA	X2	Comm'l	Standard
PSD301-20XM	200	44-pin CPGA	X2	Military	Standard
PSD301-20XMB	200	44-pin CPGA	X2	Military	MIL-STD-883C

PSD301

System Development Tools

System Development Tools

The PSD301 features a complete set of System Development Tools. These tools provide an integrated, easy-to-use software and hardware environment to support PSD301 device development. To run these tools requires an IBM-XT, -AT, or compatible computer, MS-DOS 3.1 or higher, 640K byte RAM, and a hard disk.

Hardware

The PSD301 System Programming Hardware consists of:

- ❑ WS6000 MagicPro Memory and PSD Programmer
- ❑ WS6013 44-pin LCC Package Adaptor (for CLDCC and PLDCC packages)
- ❑ WS6014 44-pin CPGA Package Adaptor

The MagicPro Programmer is the common hardware platform for programming all WSI programmable products. It consists of an IBM-PC plug-in programmer board and a remote socket adaptor.

Software

The PSD301 System Development Software consists of:

- ❑ WISPER, WSI's Software Environment
- ❑ MAPLE, the PSD301 Location Editor Software
- ❑ MAPPRO, the Device Programming Software

The configuration of the PSD301 device is entered using MAPLE software. MAPPRO software uses the MagicPro programmer and the socket adaptor to configure the PSD301 device, which then can be used. The development cycle is depicted in Figure 26.

Support

WSI provides a complete set of quality support services to registered System Development Tools owners, including:

- ❑ 12-month software updates
- ❑ Design assistance from WSI field application engineers and application group experts
- ❑ 24-hour electronic bulletin board for design assistance via dial-up modem.

Training

WSI provides in-depth, hands-on workshops for the PSD301 device and System Development Tools. Workshop participants learn how to program high-performance, user-configurable mappable memory subsystems. Workshops are held at the WSI facility in Fremont, California.

Ordering Information — System Development Tools

PSD301-GOLD

- ❑ WISPER Software
- ❑ MAPLE Software
- ❑ User's Manual
- ❑ WSI Support
- ❑ WS6000 MagicPro™ Programmer
- ❑ WS6013 44-pin LCC Package Adaptor
- ❑ Two PSD301-15L Samples

PSD301-SILVER

- ❑ WISPER Software
- ❑ MAPLE Software
- ❑ User's Manual
- ❑ WSI Support

WS6000

- ❑ MagicPro Programmer
- ❑ IBM-PC© plug-in Adaptor Card
- ❑ Remote Socket Adaptor

WS6013

- ❑ 44-pin LCC Package Adaptor for CLDCC and PLDCC Packages
- ❑ Used with the WS6000 MagicPro Programmer

WS6014

- ❑ 44-Pin CPGA Package Adaptor, Used with WS6000 MagicPro Programmer

WSI Support

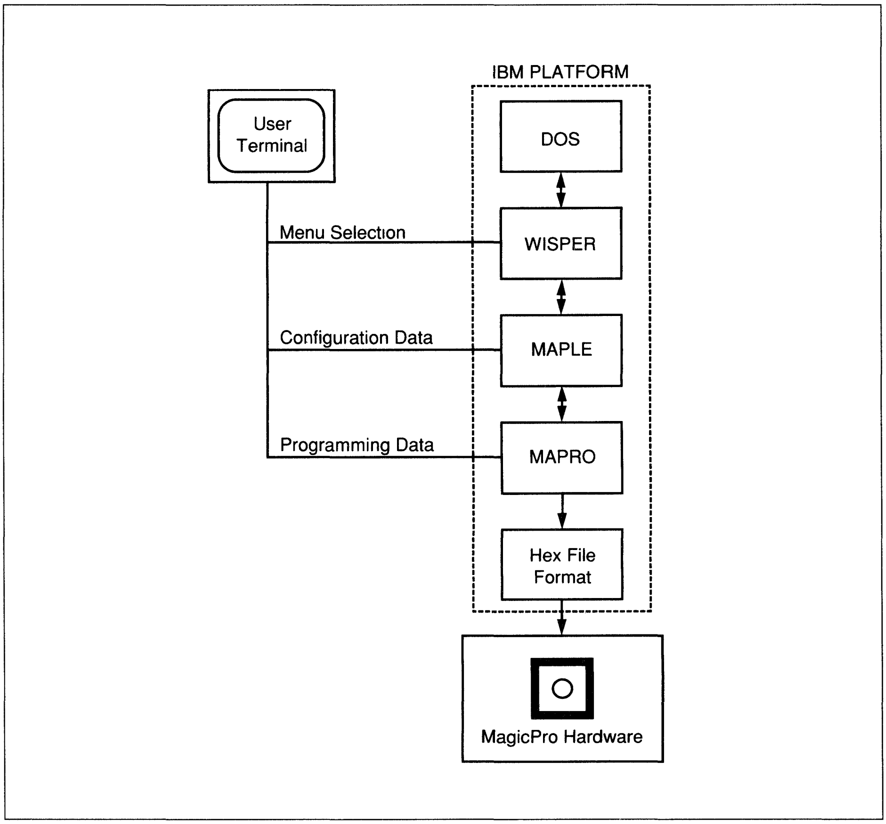
Support services include:

- ❑ 12-month Software Update Service
- ❑ Hotline to WSI Application Experts
- ❑ 24-hour access to WSI Electronic Bulletin Board

WSI Training

- ❑ Workshops at WSI, Fremont, CA
- ❑ For details and scheduling, call PSD Marketing (415) 656-5400

Figure 26. PSD301 Development Cycle



WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

PAC1000 Introduction

User-Configurable Microcontroller

Overview

In 1988 WSI introduced a new concept in programmable VLSI: the Programmable System™ Devices (PSD). The PSD is defined as a family of *User-configurable system level building blocks on-a-chip enabling quick implementation of application specific controllers and peripherals.* The first generation PSD series includes the MAP168, a User-Configurable Peripheral with Memory; the SAM448, a User-Configurable Microsequencer; and the PAC1000, a User-Configurable Microcontroller.

The PAC1000 user-configurable high-performance microcontroller is the first of a generation of products intended for applications in high-end embedded control where high-speed data processing, interface or control is needed. The PAC1000 replaces a board full of discrete components such as standard logic, FIFO, EPROM for microcode store, ALU, SEQUENCER, register files and PAL/PLD/PGA. To shorten the time-to-market for the system designer, a high-level software development language is used. This contrasts with the myriad state-machine entry, schematic entry, and place and route tools that would be needed for a discrete design using PAL, PLD, PGA or gate arrays.

The PAC1000 architecture is flexible and enables the system designer to customize the PAC1000 to optimize application performance. The PAC1000 is composed of three basic sections: a CPU for data processing, a programmable instruction control unit that determines the next address to the microcode store through polling condition codes or responding to interrupts, and a host interface to asynchronously load data from the host. Registered input/outputs are used to synchronize with the system.

As a result of integrating logic and EPROM memory into the PAC1000 and defining a high-level language for programming both, time-to-market and board space is reduced and reliability increased. The PAC1000 is currently used in applications such as Intelligent DMA controller, FDDI buffer controller, Frame buffer controller, LAN communications controller, disk controller, and I/O controller. For further details on the PAC1000 see Application Note 10.

Contents

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

PAC1000

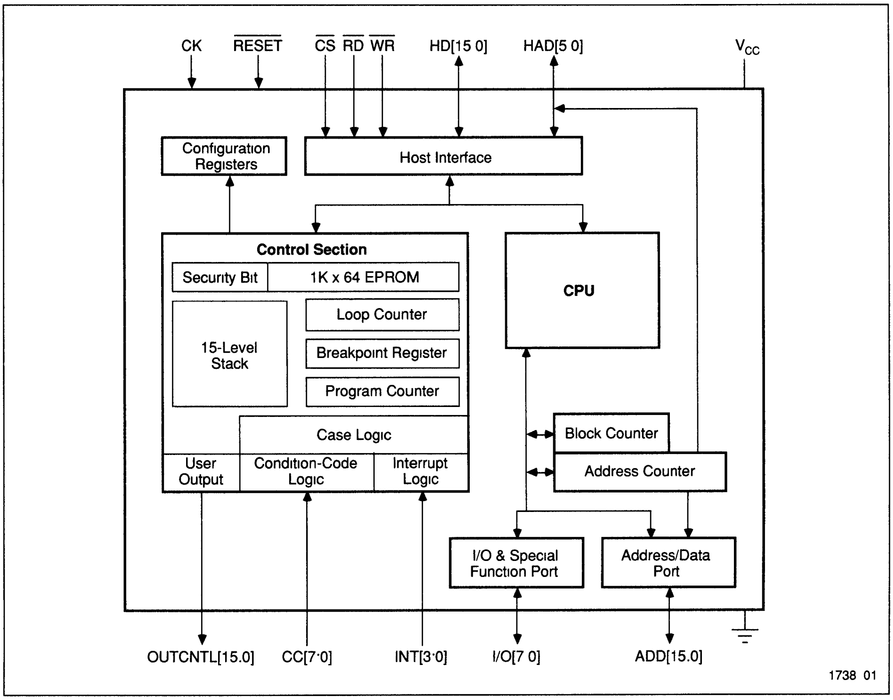
User-Configurable Microcontroller

Preliminary

Features

- ❑ First Generation Programmable System Device (PSD)
- ❑ High-Performance User-Configurable Microcontroller—20 MHz Instruction Execution, Output Port, and Address Bus
- ❑ Single-Cycle Control Architecture—One Cycle Per Instruction
- ❑ 16-bit CPU—Arithmetic Operations, Logic Operations, 33 General-Purpose Registers
- ❑ Address Generation—Up To 4 Mbytes Address Space
- ❑ High-Level Development Tools—System Entry Language, Functional Simulator, and Device Programmer
- ❑ Re-Programmable Program Store—On-Board 1Kx64-Bit EPROM
- ❑ Two Operating Modes—Host Processor Peripheral or Stand-alone Controller
- ❑ Security—For EPROM Program Memory

Figure 1. PAC1000 Block Diagram



General Description

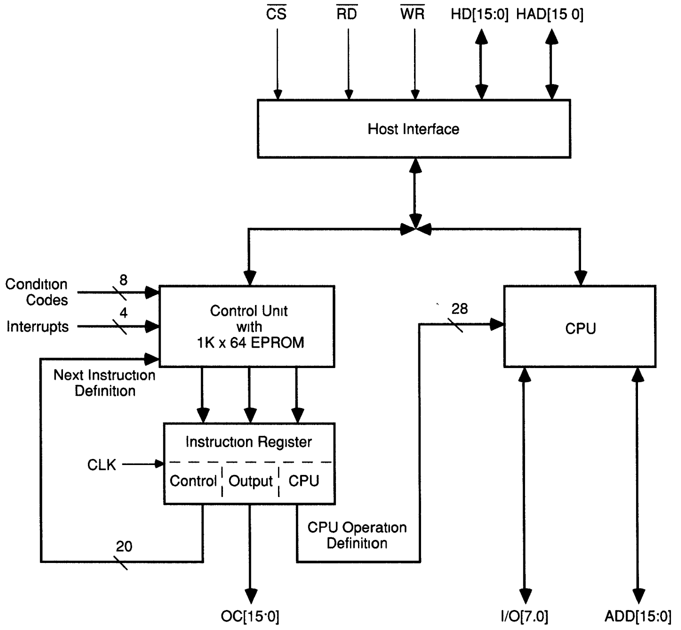
In 1988 WSI introduced a new concept in programmable VLSI, Programmable System Devices (PSD). The PSD family consists of user-configurable system-level building blocks on-a-chip, enabling quick implementation of application-specific controllers and peripherals. The first generation PSD series includes the MAP168, a User-Configurable Peripheral with Memory; the SAM448, a User-Configurable Microsequencer; and the PAC1000, a User-Configurable Microcontroller.

The PAC1000 User-Configurable Microcontroller is based upon an architecture that enables it to execute complex instructions in a single clock cycle. Each PAC1000 instruction can perform three simultaneous operations: Program Control, CPU functions, and Output Control, as shown in Figure 2. The PAC1000 can also perform address generation or event counting simultaneously with instruction execution. The PAC1000 is also capable of performing a conditional test on up to four separate conditions and multi-way branching in a single cycle.

The PAC1000, with its System Development Tools, matches the development cycle and ease of use of any standard microcontroller. The high performance and flexibility of the PAC1000 were previously available only to designers who could afford the long development cycle, high cost, high power, and large board space requirements of a building-block solution (i.e., Sequencer, Microcode Memory, ALU, Register File, PALs, etc.)

The unique capabilities of PAC1000 are easily utilized with System development tools, which include a PACSEL C-like System Entry Language, a PACSIM Functional Simulator, and a MagicPro™ Device Programmer. All System Development Tools are PC-based and will operate on an IBM-XT, AT, PS2 or compatible machine. For more information, contact your nearest WSI sales office or representative.

Figure 2. Single-Cycle Control Architecture



Important Features:

- One cycle per instruction
- 20 MHz instruction execution rate
- Every instruction executes 3 parallel operations (Control, Output, CPU)

Table 1. Pin Description

Signal	I/O	Description
HD[15:0]	I/O	*Host Data.* PAC1000 Data I/O Port via the Host Interface. Can also be configured to generate 16-bit address or status. Can serve as a general-purpose Data I/O Port.
HAD[5:0]	I/O	*Host Address.* Can be configured to output the lower six bits of the 22-bit Address Counter; can be used as a Host Interface function address, or as a general-purpose 16-bit port.
$\overline{\text{CS}}$	I	*Chip Select (active low).* Used with $\overline{\text{RD}}$ and $\overline{\text{WR}}$ to access the device via the Host Interface.
$\overline{\text{RD}}$	I	*Read Enable (active low).* Used with $\overline{\text{CS}}$ to output Program Counter, Status Register, or Data Output Register to HD[15:0] bus lines.
$\overline{\text{WR}}$	I	*Write Enable (active low).* Used with $\overline{\text{CS}}$ to write HD Bus data via the Host Interface into the PAC1000 FIFO.
CK	I	*Clock.*
CC[7:0]	I	*Condition Codes.* Condition-code inputs for use with Call, Jump, and Case instructions.
INT[3:0]	I	*Interrupts.* General-purpose, positive-edge-triggered interrupt inputs.
$\overline{\text{RESET}}$	I	*Asynchronous Reset (active low).* Resets Input/Output registers and counters, tri-states all I/O, and sets the Program Counter to 0.
OUTCNTL[15:0]	O	*Output Control.* User-defined Output Port. May be programmed to change value every cycle.
ADD[15:0]	I/O	*Address Port.* Outputs data from Address Counter or Address Output Register when configured as an output. When configured as an input, reads data to Address Input Register.
I/O[7:0]	I/O	*Input or Output Port.* Individually configurable bidirectional bus. As simple I/O, outputs come from the I/O Output Register, and inputs appear in the I/O Input Register. As special I/O functions, provides status, handshaking, and serial I/O. Alternatively, these signals can be used to extend the OUTCNTL or ADD lines.

Architectural Overview

The PAC1000 is a user-configurable microcontroller optimized for high-performance control systems. The primary architectural elements, shown in Figure 3, are the Control Section, 16-bit CPU, Host Interface, 16-bit Address Port, 16-bit Output Control, 8-bit I/O Port, and Configuration Registers.

The PAC1000 can be used as a stand-alone microcontroller or as a peripheral to a host. In the latter case, the Host Data (HD) and Host Address (HAD) buses, together with the $\overline{\text{CS}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ pins allow for direct connection to a host bus. User-defined commands to the Control Section or data to the CPU can be loaded through the Host Interface.

In the stand-alone mode, the Host Interface ports can be used as additional address, data or I/O ports using the Data Output Register (DOR) and Data Input Register (DIR). The ADD port can be used to generate addresses through the Address Output Register (AOR) or the Address Counter. A DMA channel can be formed on the Host Interface using these and the Block Counter (BC) register. In addition, the ADD port can be used as a data bus or an I/O port, depending on how the chip is configured. Each pin in the I/O port can be configured individually as input, output, or special function. The special functions allow the control of internal PAC1000 elements (counters, I/O buffers) by other board elements.

The 16-bit CPU is highly parallel and can operate on operands from the 32x16-bit register file, miscellaneous register (AOR, AIR, DOR, DIR, Q, etc.), or constants loaded from the internal program-store EPROM.

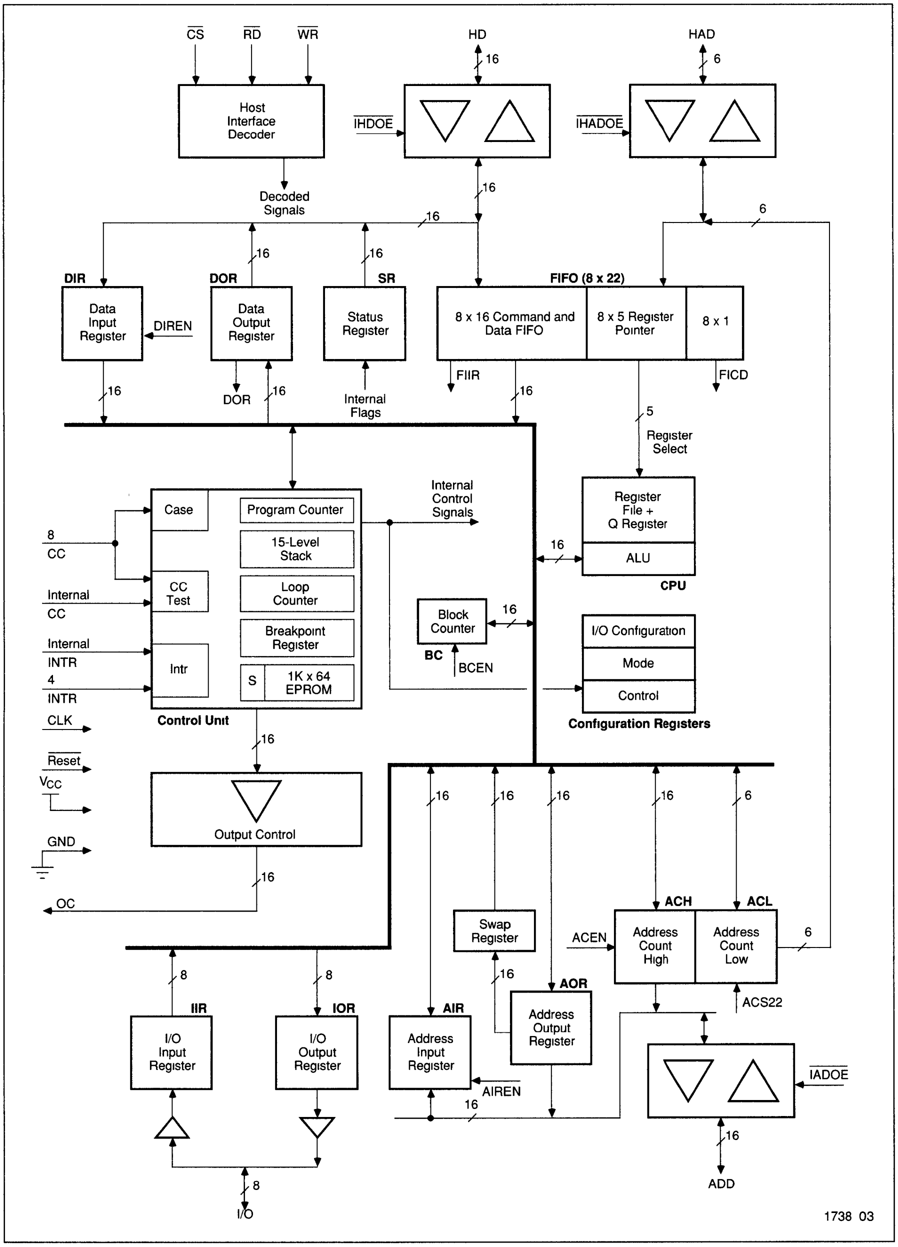
The internal and external operations of the PAC1000 are controlled by the Control Section. The 16 Output Control (OC) lines are general-purpose outputs. Each of them can be changed independently every clock cycle. They provide a very fast means to control various processes outside the chip.

In every clock cycle, one instruction is executed. Each instruction consists of up to three operations in parallel:

- ❑ Instruction Fetch—the next instruction is fetched from the 1Kx64 EPROM by the Program Control.
- ❑ Execution—the CPU executes an instruction.
- ❑ Output—placed on the Output Control (OC) lines.

Program flow can be changed through the condition-code inputs in one clock cycle or through the interrupt inputs after two clock cycles. Single-cycle 16-way branches can be done using the Case instruction, which samples four condition codes per cycle. Nested loops and subroutines can be carried out with the 15-level stack and the loop counter. The chip configuration can be changed in any cycle by loading the Configuration Register using the Program Control instruction portion.

Figure 3. Detailed Block Diagram



Operational Modes

The two basic modes of operation for the PAC1000 are either as a memory-mapped peripheral (Figure 4) or as a stand-alone controller (Figure 5).

In the peripheral mode, the host processor can asynchronously interface with the PAC1000.

Figure 4. Peripheral Mode

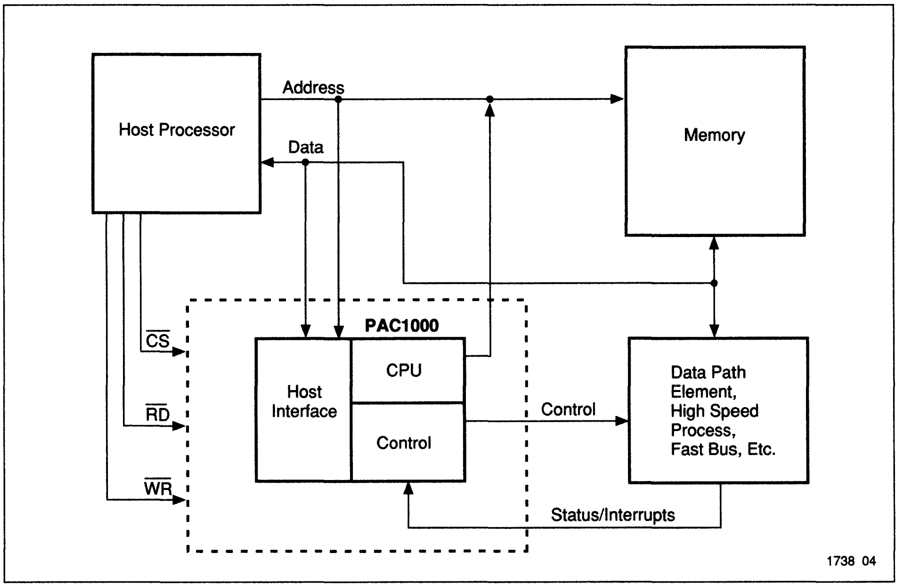
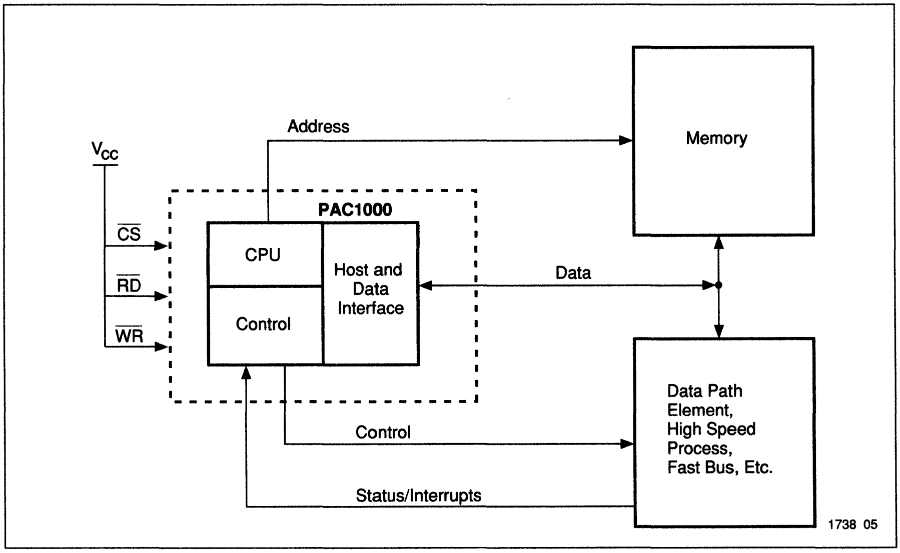


Figure 5. Stand-alone Mode



Host Interface

The Host Interface section of the PAC1000, shown in Figure 6, includes the Input Command/Data FIFO, Input/Output Data Registers, and the Status Register.

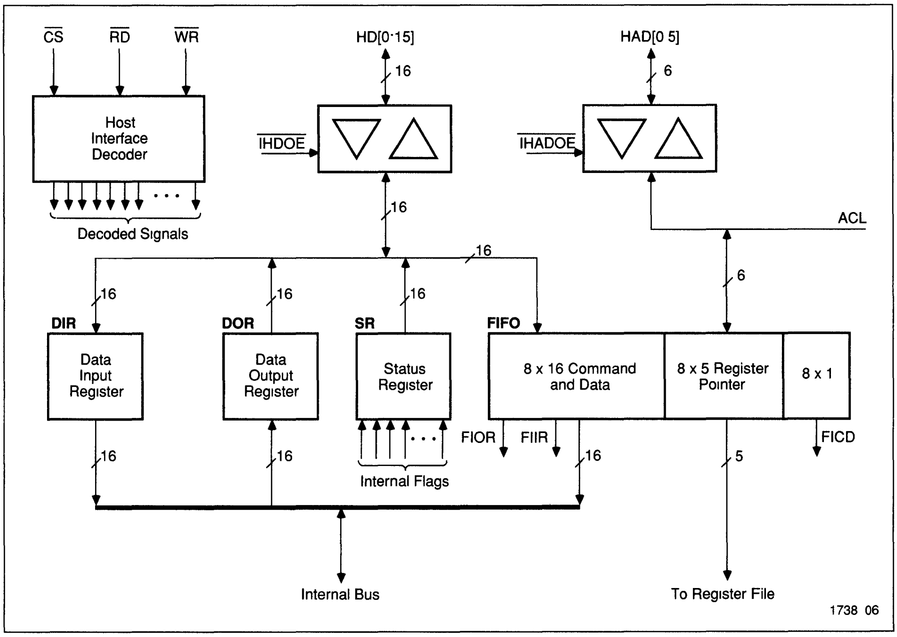
FIFO

When the PAC1000 serves as a peripheral to a host, the FIFO is used to asynchronously load commands or data into the PAC1000. In order to write into the FIFO, $\overline{CS}$ and $\overline{WR}$ must have low-to-high transitions. The information written into the FIFO is specified by the 16-bit Interface Data bus (HD) and the 6-bit Host Address bus (HAD). Since the FIFO is used only to buffer data and commands from a host, it is inoperative when the PAC1000 is in stand-alone mode.

Bit five of the HAD bus specifies whether the input to the FIFO is command (HAD5=1) or data (HAD5=0). HAD5 is connected to the FICD internal Condition Code that can be sampled by the Control Section. If a command is written, then the lower 10 bits of the HD bus are used as the branch address for one of the 1024 locations in the Program Memory EPROM. At that location a user defined command or subroutine should exist which executes the needed operation. If the information is data, then the lower 5 bits of the HAD bus specify which CPU register is to be loaded from the HD bus.

This method of operation allows the host to access the PAC1000 as a memory-mapped peripheral.

Figure 6. Host Interface Architecture



Host Interface (Con't)

An example of FIFO usage is shown in Figure 7. When command or data information is available in the FIFO, the FIFO Output Ready (FIOR) interrupt (interrupt 5) triggers. If the FIOR interrupt is masked, then the FIOR status may be polled under program control. If HAD5 equals 1, the branch address location specified by MOVE is the Program Memory Address which contains the user specified instruction or sub-routine which executes the command. A JUMP or CALL FIFO control instruction performs a jump or call to the location specified by MOVE. If HAD5 equals 0, an RDFIFO instruction can transfer the FIFO contents into the register specified by HAD[4:0].

For further explanation, refer to the diagram below. Beginning at the location specified by MOVE, a user defined program exists which is going to load data into CPU registers 0,1,2, and 3 in four consecutive cycles from the next four FIFO locations. If one of the four FIFO locations contains a command (FICD=1), interrupt level 7 occurs (highest level). Loading a command into a CPU or other data register is not allowed. If this occurs, FIXP (FIFO exception) will be generated.

Following the execution of this routine, the Control Section is ready for its next instruction.

The FIFO drives three internal flags which can also be programmed to interrupt the PAC1000. They are:

- ❑ $\overline{\text{FIIR}}$ (FIFO full) and FIXP (FIFO exception), which drive INT7.
- ❑ FIOR (FIFO output ready), which drives INT5.

Table 2. Host Interface Functions

$\overline{\text{CS}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	HAD5	HAD[4:0]	HD[15:0]	Function
0	1	0	0	Register Address	Data	Write data to FIFO
0	1	0	1	X	Command	Write command to FIFO
0	0	1	0	00100	X	Reset FIFO
0	0	1	0	00011	X	Reset status register
0	0	1	0	00010	Data	Read program counter
0	0	1	0	00001	Data	Read status register
0	0	1	0	00000	Data	Read data output register

Host Interface
(Con't)

Data I/O Registers

Input and Output Data Registers are used to communicate with the Host Data (HD) bus. CPU Registers may be loaded directly from the Data Input Register (DIR) without passing through the FIFO. Similarly, the PAC1000 may be read via the Data Output Register (DOR).

Program Counter

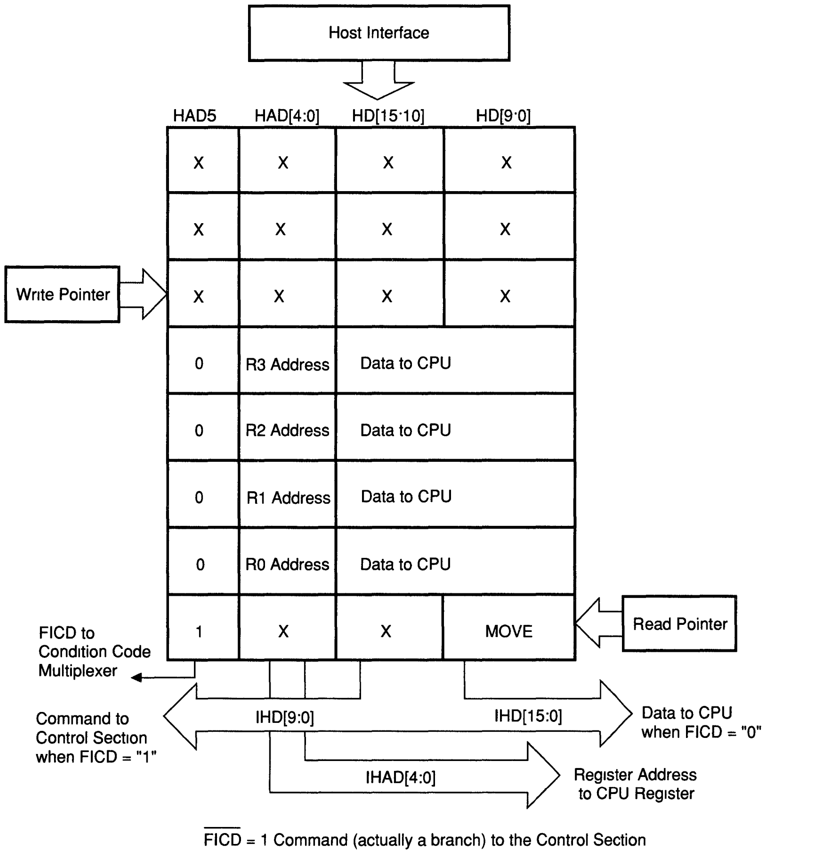
The Program Counter may be read via the Host Data bus. This allows a host to monitor the Program Memory address bus. It can also be used to drive external memory devices for expansion of the Control Port.

Status Register

The Status Register (SR), shown in Figure 8, monitors all internal status. Status bits can be set only by program execution. The SR can be read or cleared as specified in the Host Interface Functions table.

All SR flags are active high (1) and are latched at the rising edge of the clock.

Figure 7. Example of FIFO Block Diagram and Usage



$\overline{\text{FICD}}$ = 1 Command (actually a branch) to the Control Section

$\overline{\text{FICD}}$ = 0 Data to CPU Register

1738 07

Host Interface (Con't)

STAT11—(DBB) *Security Bit*, set when security is active:

1= Security active.

0= No security.

STAT10—*WSI Reserved.*

STAT9—(FIXP) *FIFO Exception*, set when the CPU receives a command or Control Section receives data:

1= Command or data received.

0= No exception occurred.

STAT8—(FIIR) *FIFO-Input Ready*, set when there is at least one vacant location in the FIFO:

1= FIFO ready for input.

0= FIFO not ready for input.

STAT7—(CY) *Carry Flag*, set when a carry (addition) or borrow (subtraction) occurs in CPU operations:

1= Carry occurred.

0= No carry occurred.

STAT6—(Z) *Zero Flag*, set when the result of a CPU operation is zero:

1= Zero occurred.

0= No zero occurred.

STAT5—(O) *Overflow Flag*, set when an overflow occurs during a two's complement operation:

1= Overflow occurred.

0= No overflow occurred.

STAT4—(S) *Sign Bit*, set when the most significant bit of the result of the previous CPU operation is negative:

1= Result is negative.

0= Result is positive.

STAT3—(STKF) *Stack Flag*, set when the stack is full:

1= Stack is full.

0= Stack is not full.

STAT2—(BRKPNT) *Breakpoint Flag*, set when the address in the breakpoint register is equal to the EPROM address:

1= Breakpoint occurred.

0= No breakpoint occurred.

STAT1—(BCZ) *Block Counter Zero*, set when the counter decrements to all 0s:

1= Block Counter reached zero.

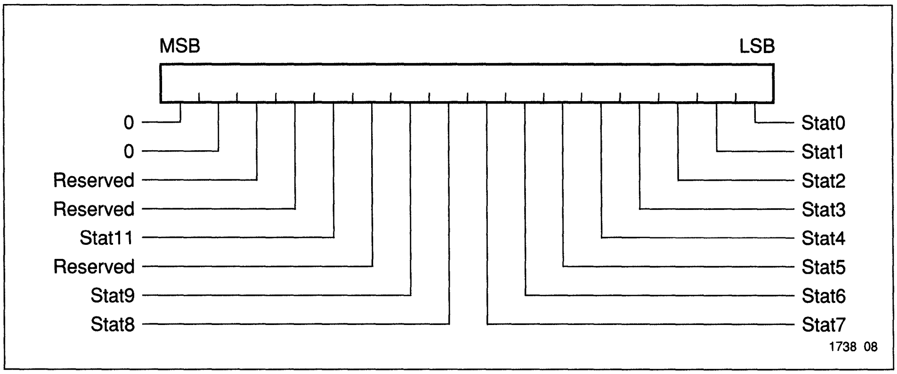
0= Block Counter is not zero.

STAT0—(ACO) *Address Counter Ones*, set when the counter increments to all 1s:

1= Address Counter reached all ones.

0= Address Counter is not all ones.

Figure 8. Status Register



Control Section

The control section, shown in Figure 9, consists of a number of blocks which are concerned with the sequencing of the control programs in the PAC1000. These are:

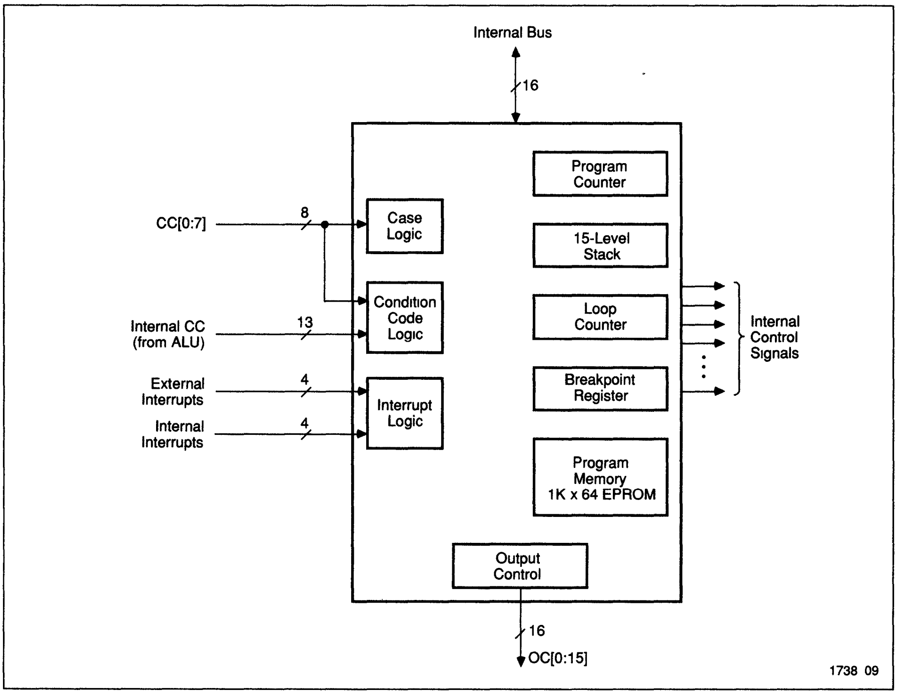
- ❑ Program Memory
- ❑ Security
- ❑ 15-Level Stack
- ❑ Program Counter
- ❑ Loop Counter
- ❑ Breakpoint Register
- ❑ Condition Codes
- ❑ Case Logic
- ❑ Interrupt Logic
- ❑ Output Control

Each block is described in detail below.

Parallel Operations

The PAC1000 can perform three simultaneous operations within a single instruction cycle, as shown in Figure 10. The ability to fetch an instruction from the Program Memory, execute it, and output a result within 50 nsec is due to a highly parallel structure.

Figure 9. Control Architecture



Control Section (Con't)

Program Memory

The Program Memory is a 1Kx64 high-speed EPROM. This on-board-memory allows the PAC1000 to operate in embedded control applications and eliminates the need for external memory components. Using an erasable memory allows program code to be modified for debug and/or field upgrades. The Program Memory is easily programmed using the WSI MagicPro™ (Memory and PSD™ Programmer).

Only sixteen Program Memory locations are reserved. The rest of the 1024 locations are available for applications.

Program memory is segmented as follows:

Address	*Function*
000H	Reset pointer program to here
000H–007H	User Defined Initialization Routine
008H–00FH	Interrupt Vector Locations
010H–3FFH	User-Defined Application Programs

Upon receiving a reset, the Program Counter is forced to address 000H. This location may contain a jump or call which branches to an initialization routine. Alternatively, the first eight locations of memory may be used as an initialization/configuration routine.

Security

User programs may be protected by setting a security bit during EPROM programming. Thereafter, the EPROM contents cannot be read externally. When the EPROM is erased, the security bit is cleared.

15-Level Stack

The 15-level Stack stores the return address following subroutine calls, interrupt service routines and the contents of the Loop Counter inside nested loops. When the stack is full, the STKF condition becomes true, and an interrupt (INT7) will occur. The interrupt service routine will overwrite the top of the stack.

Popping from an empty stack produces the previous top of stack value; pushing on a full stack overwrites the top of the stack.

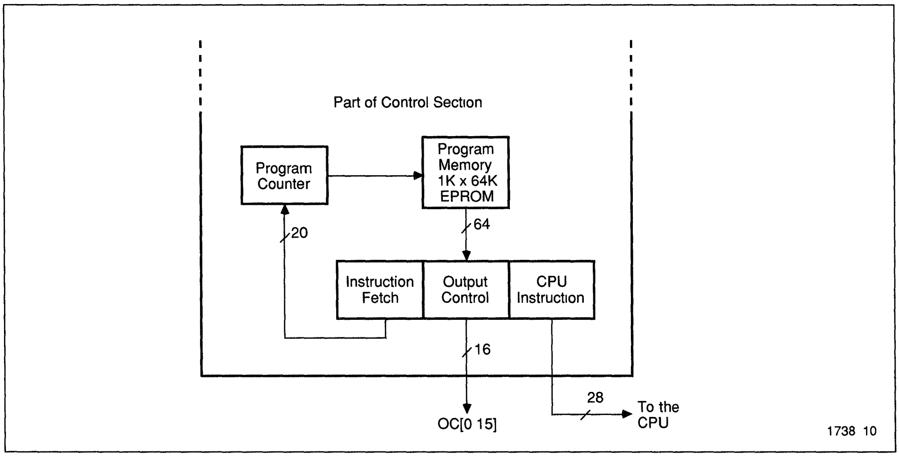
Program Counter

The 10-bit Program Counter (PC) generates sequential addressing to the 1K word Program Memory. Upon reset the PC is loaded with a 000H. From this point the value of the Program Counter is determined by program execution or interrupts.

Any JUMP or Case instruction that is executed loads the Program Counter with the destination address. CALL instructions or interrupts cause PC + 1 to be pushed onto the stack. The RETURN instruction loads the Program Counter from the stack with the value of the return address. This value may have previously been placed on the stack by a CALL or interrupt.

The PC can also be loaded from the Command/Data FIFO causing program execution to commence at an address provided by the host.

Figure 10. Parallel Operations



Control Section (Con't)

Loop Counter

The Loop Counter (LC) has two functions:

- ❑ 10-bit down counter that supports the LOOP instruction.
- ❑ Branch Register that can be loaded from the CPU Register File or Program Memory and used as an additional source of branching to Program Memory.

The LC can be loaded with values up to 1023. Loop initialization code places a value in LC. Loop termination code tests the counter for a zero value and then decrements LC. The loop count can be a constant, or it can be computed at execution time and loaded into LC from the CPU. The LC register can also be used as a CALL or JUMP execution vector. The content of the LC is automatically saved on (or retrieved from) the Stack when the program enters (or leaves) a nested loop.

A loop count will be loaded into the LC when a FOR instruction is encountered. This count can be a fixed value or it can be calculated and loaded from the CPU. The ENDFOR instruction will test the Loop Counter for a zero value. If this condition is not met, then the LC will be decremented by one. The program loop will continue until the count value equals zero. In a nested loop, the FOR instruction will load a new value to the LC and push the previous value to the stack.

Debug Capabilities

The PAC1000 provides breakpoint and single step capabilities for debugging application programs.

Breakpoint Register

The Breakpoint Register (BR) is a 10-bit register used for real time debug of the PAC1000 application program.

The Breakpoint Register can be loaded from one of two sources, either a constant value specified in the Program Memory or a calculated value loaded from the CPU. When the Program Memory address matches the contents of the Breakpoint Register an interrupt (INT 6) occurs. A service routine should exist in Program Memory which then performs the required procedure.

Single Step

Single step is a debugging mode in which the currently-executing program is interrupted by interrupt 6 after the execution of every instruction. The interrupt 6 service routine should reside in Program Memory.

Bit 8 in the Mask Register determines whether the PAC1000 is in a breakpoint mode (mask-bit 8 equals 0) or in a single step mode (mask-bit 8 equals 1).

Both breakpoint and single step use interrupt 6. The interrupt 6 service routine will typically dump the contents of the PAC1000 internal registers into external SRAM devices for examination by the user.

Condition Codes

The Condition Code (CC) logic operates on 21 individual program test conditions. Each condition can be tested for true or not true. The PAC1000 can also test up to four conditions simultaneously. For this feature refer to the section titled *Case Logic*.

Control Section (Con't)

User-Specified Conditions

User-Specified Conditions are treated in the same manner as internally generated test conditions. CC0—CC7 should be connected directly to the corresponding PAC1000 input pins. These signals must satisfy the required setup time to be serviced in the next cycle.

CPU Flags

CPU flags are internally generated. They reflect the status of the previous CPU arithmetic operation. These signals are internally latched and are valid only for one instruction (the instruction following their generation). The flags for arithmetic operations are defined as follows:

Zero (Z)—The result of the previous CPU operation is zero (Z=1).

Carry (CY)—The result of the previous CPU operation generated a carry (addition) or borrow (subtraction) (CY=1).

Overflow (O)—The previous two's complement CPU operation generated an overflow (O=1).

Sign (S)—The most significant bit of the result of the previous CPU operation is negative (S=1).

FIFO Flags

FIFO flags allow the user to synchronize and monitor the operations that are performed on the FIFO by the host or by user's program.

Upon reset the FIFO flags are cleared, signifying an empty state. The meaning of the flags are as follows:

FIFO Output Ready (FIOR)—There is at least one word in the FIFO (FIOR=1).

FIFO Input Ready (FIIR)—FIFO is not full (FIIR=1). This flag can also be connected to the host through I/07.

FIFO Command/Data (FICD)—This flag indicates if the contents of the FIFO is a command or a data. This flag is generated directly from HAD5 (FICD=1 command, FICD=0 data).

FIFO Exception (FIXP)—This flag indicates that one of two events occurred: (a) FIFO data has been read as a command, or (b) a command has been read as data.

Stack-Full Flag

STACK FULL flag (STKF=1) indicates that the stack is 15 levels full. This condition will also generate an interrupt (INT7) if not masked.

Interrupt Flag

INTERRUPT flag (INTR =1) indicates that there is a masked interrupt pending. This flag is cleared when the interrupt is cleared.

Data Register Read Flag

DATA REGISTER READ flag (DOR) is a handshake flag between the host and the PAC1000, accessible only to the PAC1000. The flag is reset (DOR=0) when the PAC1000 writes into the Data Output Register. The flag is set (DOR=1) after the host has performed a read on the Data Output Register.

Counter Flag

Counter flags reflect the status of their respective counters. The PAC1000 utilizes two counters; the Address (A) counter is a 16/22-bit auto-incrementing up counter; the

Table 3. Condition-Code Logic

Test Group	Source	Conditions and Flags
User-Specified	External	CC0–CC7
CPU	Internal	Carry (CY), Zero (Z), Overflow (O), Sign (S)
FIFO	Internal	FIFO Command/Data (FICD), FIFO Output Ready (FIOR), FIFO Input Ready (FIIR), FIFO Exception (FIXP)
Counters	Internal	Address Counter Ones (ACO), Block Counter Zero (BCZ)
Stack	Internal	Stack Full (STKF)
Interrupt	External/Internal	Interrupt (INTR) is pending
Data register read	Internal	Data Output Register(DOR) has been read

Control Section (Con't)

Block (B) counter is an auto-decrementing 16-bit down counter. The counters' clock input signal is the same as the PAC1000's clock signal. Each counter can be individually enabled or disabled. When disabled, the output retains the last count. The counter flags are defined as follows:

ACO—*A Counter Ones*, set when the A counter has reached the value FFFFH, in the 16-bit mode, or the value 3FFFFFH in the 22-bit mode.

BCZ—*B Counter Zero*, set when the B counter has reached the value 0000H.

Case Logic

THE PAC1000 hardware implements two basic types of Case instructions: Case and Priority Case.

Case Instructions

Case instructions operate on any one of four different Case groups. Each Case group consists of a combination of four test conditions which can be tested in a single cycle. In that same cycle the PAC1000 will branch to one of the addresses contained in the sixteen memory locations following the instruction, depending on the status of the four inputs to the Case group being tested.

There are four Case Groups (sets of Case Conditions):

Case Group 0 (CG0): CC0–CC3.

Case Group 1 (CG1): CC4–CC7.

Case Group 2 (CG2):
Z—Zero
O—Overflow
S—Sign
CY—Carry

Case Group 3 (CG3):
INTR—Interrupt
BCZ—B Counter Zero
FIOR—FIFO output Ready
FICD—FIFO Command/Data

(The FIXP, ACO, STKF, FIIR, and DOR condition codes do not fall into any of the four Case groups.)

Priority Case Instructions

Priority Case instructions operate on the four internal and the four external interrupt inputs. In this mode of operation, interrupts are treated as prioritized test conditions and the priority encoder is used to generate a branch to the highest priority condition. The branch address is located in one of the nine memory locations following the Priority Case instruction. Priorities in this mode of operation are the same as in the Interrupt mode of operation. Once a Priority Case instruction is executed, the occurrence of a higher priority condition will not affect program execution until another Priority Case instruction is executed. For a Priority Case instruction to be executed, MODE0 of the Mask Register must be equal to zero (MODE0=0).

Interrupt Logic

The Interrupt Logic accepts eight inputs, four of them are generated externally and four are dedicated for internal conditions. The four external, user defined, inputs (INT0–INT3) are connected to pins INT0, INT1, INT2, and INT3. These are positive, rising-edge-triggered signals that have a maximum latency of two cycles. Each interrupt has a reserved area in memory that should contain a branch to an interrupt service routine.

Table 4. Interrupt Assignments

Interrupt	*Priority*	*Effect*	*Trigger Condition*	*Reserved Address*
INT7	Highest	Internal	FIXP+ACO+STKF+$\overline{\text{FIIR}}$	00FH
INT6		Internal	BRKPT	00EH
INT5		Internal	FIOR	00DH
INT4		Internal	Software Interrupt (SWI)	00CH
INT3		External	INT3	00BH
INT2		External	INT2	00AH
INT1		External	INT1	009H
INT0	Lowest	External	INT0	008H

Control Section (Con't)

Clearing a serviced interrupt is performed automatically. When the interrupt is serviced, the internally generated vector is decoded to clear the serviced interrupt. In addition, the user can clear any pending interrupt by using the Clear Interrupt Instruction (CLI).

Interrupt Mask Register

The Interrupt Mask Register, shown in Figure 11, allows individual interrupts to be masked. Setting a Mask Register bit to a 1 masks the associated interrupt. To unmask an interrupt, the appropriate Mask Register bit must be reset to 0.

When the PAC1000 is reset,the Mask Register will mask all interrupts and the Mode Register will select the non-interrupt mode. To select the interrupt mode the MODE0 bit (see Configuration Register section in this document) should be set to 1 (MODE0=1).

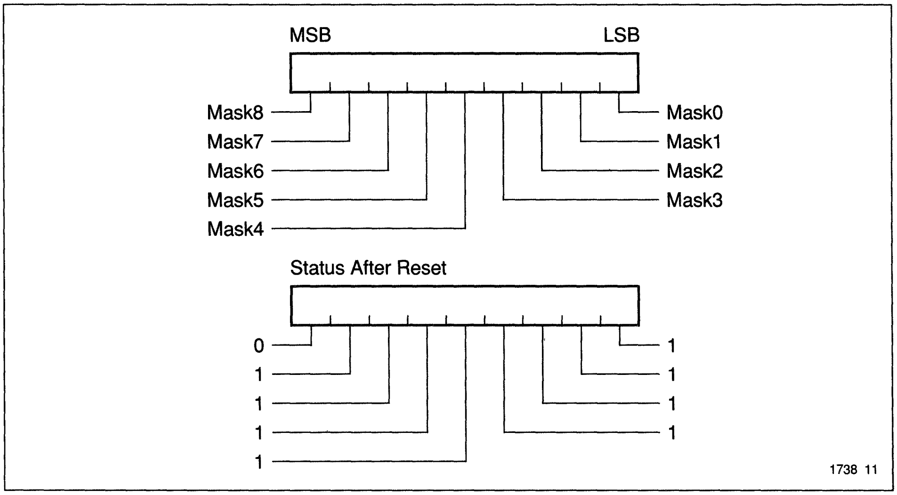
Mask8 is used to select INT6 to be either a single-step interrupt (when Mask8=1) or a breakpoint interrupt (when Mask8=0) .See the section on Debug Capabilities for further details.

Table 5. Interrupt Definitions

Interrupt	***Triggered By***
INT7[1]	FIFO Exception (FIXP)
	Address Counter contains all Ones (ACO)
	Stack Full (STKF)
	FIFO Full (Not FIFO Input Ready, $\overline{\text{FIIR}}$)
INT6[2]	Breakpoint or Single Step occurrence
INT5	FIFO Output Ready (FIOR)
INT4	Always pending; triggers when unmasked by program execution
INT3	User-defined
INT2	User-defined
INT1	User-defined
INT0	User-defined

Notes:
1. The INT7 interrupt handler checks the source of the interrupt by testing the condition code.
2. See Interrupt Mask Register, Mask8.

Figure 11. Interrupt Mask Register



Control Section (Con't)

Output Control

The Output Control bus (OUTCNTL) consists of 16 latched Output Control signals. These signals can be changed on a clock to clock basis. For every Program Memory location there is a dedicated field which specifies the value of the Output Control bus. The OUTCNTL Operation places this value on the Output Control bus. The OUTCNTL Operation can be performed in parallel with any other PAC1000 instructions.

The OUTCNTL bus can be used to control external events on a clock to clock basis.

Counters

The PAC1000 contains a 16 or 22-bit Address Counter and a 16-bit Block Counter. Each of these counters can change count on a clock to clock basis or can be internally or externally enabled or disabled on a clock to clock basis. These counters are in addition to the Loop and Program Counters of the Control Section.

Address Counter

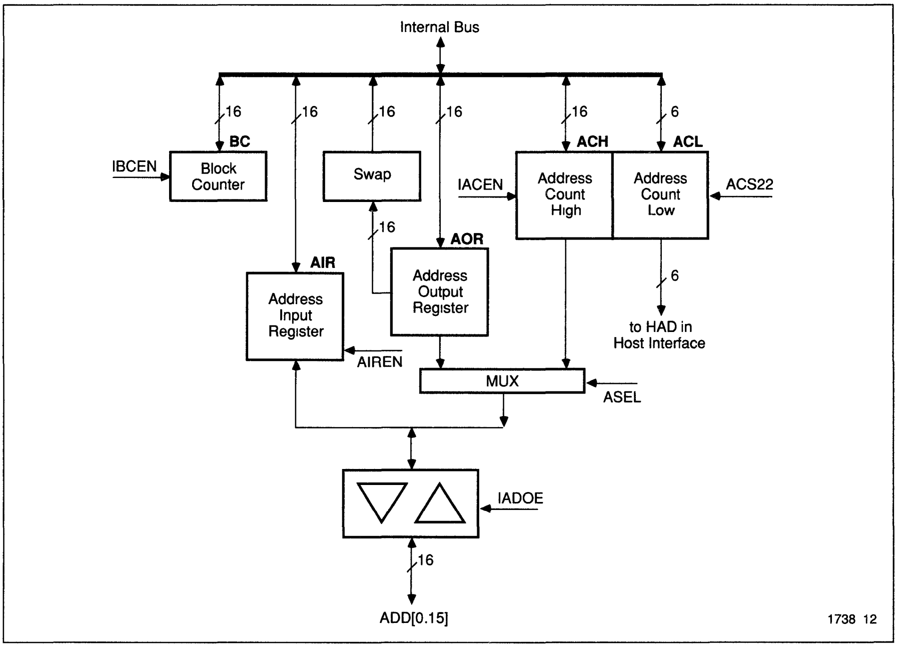
The Address Counter (AC), shown in Figure 12, is a 16- or 22-bit ascending counter that can be loaded or read by the CPU and enabled/disabled with the ACEN bit of the Control Register. (This control is also available externally through the I/01 pin; see I/O and Special Functions). While enabled, the counter will increment by one every rising edge of the clock.

The ACO flag indicates that the value of the counter is all ones. This flag stays latched until the counter is loaded with a new value. The counter will continue to count until disabled. ACO is a condition code and a member of a Case Group; see the Control Section description for more details. ACO can also generate an internal interrupt 7, if enabled.

In the 16-bit mode, the counter outputs (ACH) are available through the ADD bus. The count is gated to the ADD bus by setting the ASEL bit (CTRL9) of the Control Register.

In the 22-bit mode, the higher 16 bits (ACH) are available through the ADD bus and the six low order bits (ACL) are available through the Host Address (HAD) bus. These low order bits are multiplexed with the host address lines. The address lines from the host which drives the HAD bus must be placed in the high impedance state before the lower 6-bits (ACL) of the Address Counter can be read.

Figure 12. Address and Block Counter



Counters
(Con't)

Selecting the 16- or 22-bit count mode is performed by setting or resetting the ACS22 bit in the I/O Configuration Register.

The address Output Register is an alternate source of address outputs; it is selected by resetting the ASEL bit of the Control Register. In this mode the CPU can be used to provide address generation and the Address Counter can be used as an event counter.

Block Counter

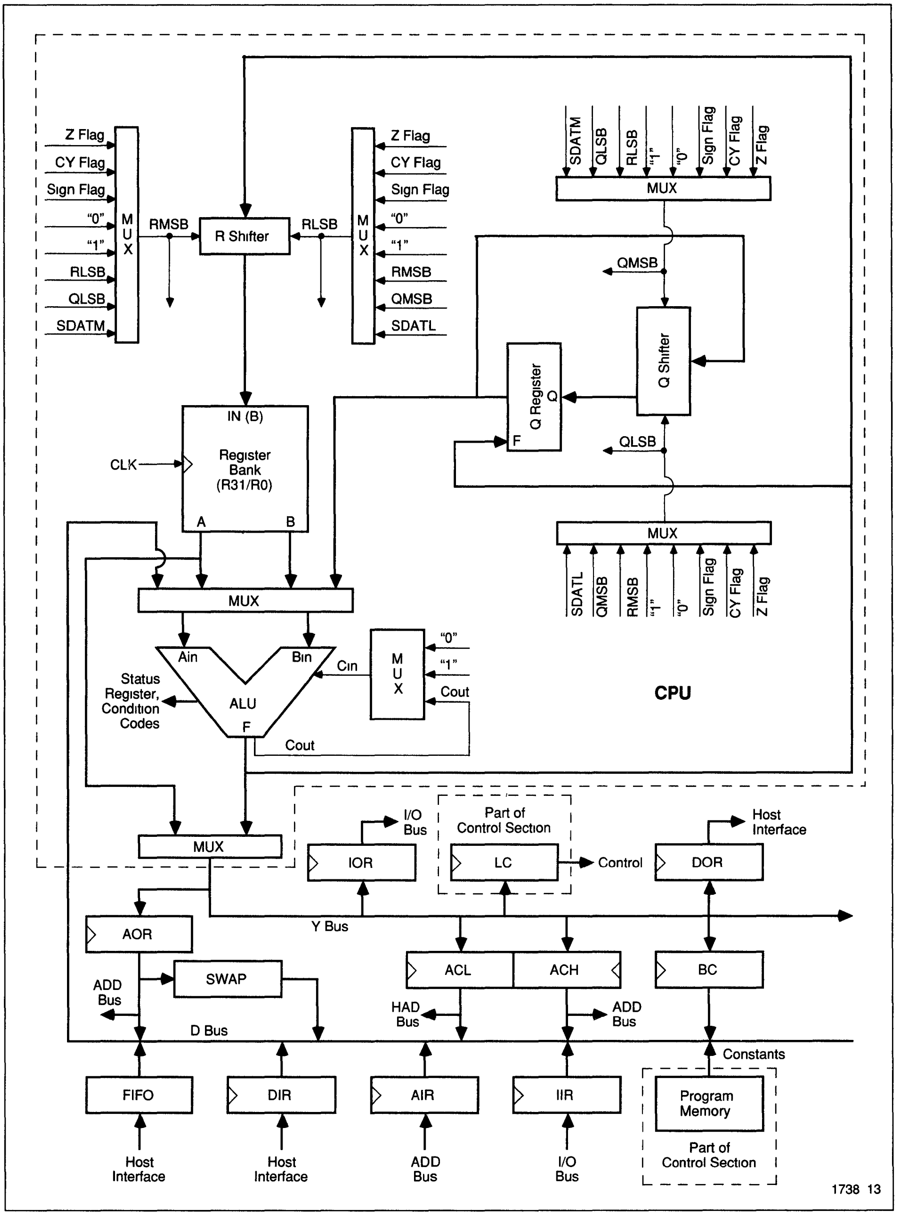
The Block Counter (BC) is a 16-bit down counter. It is enabled by the BCEN bit of the Control Register. It is useful as a counter for DMA transfers. The BCEN signal is (optionally) available externally through the I/O0 bit (see I/O and Special Functions). While enabled, the counter will decrement by one every rising edge of the clock. The BCZ flag indicates that the counter reached the zero value. After the occurrence of an all 0s condition the Block Counter will continue down counting until disabled. The flag is latched and can be cleared by loading a new value into the Block Counter. BCZ is a condition code and a member of a Case Group; see the Control Section description for more details.

Both counters may be read without disabling the count operation and loaded via the CPU.

Central Processing Unit

The CPU, shown in Figure 13, performs 16-bit operations in a single clock cycle. It contains 33 general purpose registers (R0...R31, and Q). The Q register can be used in conjunction with any of the R0...R31 registers to perform double precision shift operations. The main building blocks are the register bank (R0...R31), Q register, ALU, Y-bus devices, and D-bus devices. The register bank supplies up to two 16-bit registers, one of which is always the destination register.

Figure 13. CPU Block Diagram



Central Processing Unit (Con't)

The ALU operates on up to two external operands that are selected by its input MUX. In every instruction, 1 of the 10 D-bus devices (AOR, SWAP, ACL, ACH, BC, FIFO, DIR, AIR, IIR, and Program Store) or a member of the register bank or the Q register outputs, can be selected as an operand source to the ALU. The possibilities are shown in Figure 14. During ALU operations, three options can be selected to provide the carry-in (Cin) input: 0, 1, or the previous latched carry-out (adequate for multiple precision operations).

The ALU's output or a selected register can be loaded into one of the seven Y-bus devices (IOR, AOR, LC, DOR, ACL, ACH, or BC) every instruction cycle. This can happen in parallel with the feedback path from the ALU's output that is directed either to the Q register or to the destination register of the register bank.

Figure 14. CPU Sources and Destinations

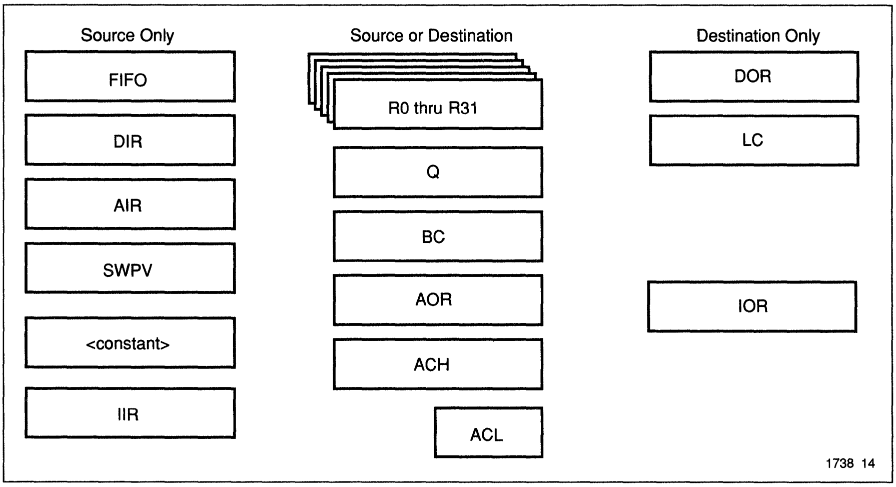


Table 6. CPU Operand Mnemonics

Mnemonic	*Description*
ACH or ACH/ACL	16- or 22-bit Auto-incrementing Counter, or General Purpose Registers
AIR	Address Input Register
AOR	Address Output Register
BC	Block Counter (16-bit auto-decrementing), or General Purpose Register
<constant>	Constant values in Program Storage
DIR	Data Input Register
DOR	Data Output Register
FIFO	Input Data from FIFO
IIR	I/O Input Register
IOR	I/O Output Register
LC	Program Loop Counter
Q	16-bit CPU Register
R0–R31	16-bit CPU Registers
SWPV	Byte Swap version of AOR

Central Processing Unit (Con't)

CPU operations can be performed on one, two or three operands. Each operation is performed in a single clock cycle. In two- or three-operand instructions, one of the operands must be a CPU internal register (R0...R31, or Q).

CPU operations are performed independently of operations in the counters, Host Interface, Output Control, and Program Control.

Arithmetic Operations

The CPU can perform the following arithmetic operations:

- ❑ Addition
- ❑ Subtraction
- ❑ Increment
- ❑ Decrement
- ❑ Compare

Logic Operations

The CPU can perform the following logic operations:

- ❑ AND
- ❑ OR
- ❑ Invert
- ❑ Exclusive OR
- ❑ Exclusive NOR

Shift Operations

Single shift operations, shown in Figure 15, can occur either to the left or to the right, with or without the Q register. Shift instructions specify the sources that are shifted into the corresponding registers.

All shift operations can be executed in the same clock cycle as an arithmetic or logic operation. The arithmetic or logic operation is executed first; the result is shifted and then stored in the register file. The shift can be either left or right.

The CPU can perform the following shift operations:

- ❑ Single-precision, left or right, within a general-purpose register (R0...R31, or Q).
- ❑ Double-precision, left or right, between an R0...R31 register and the Q register.

The LSB and MSB of the general-purpose registers are each fed by an eight-to-one multiplexer.

The sources and destinations for shift operation are given below:

Shift Right

Zero Flag (Z)

Carry Flag (CY)

Sign Flag (S)

Binary 0 (0)

Binary 1 (1)

Least-significant bit of this register (RLSB)

Least-significant bit of the Q register (QLSB)

Serial I/O port (SDATM)

Shift Left

Zero Flag (Z)

Carry Flag (CY)

Sign Flag (S)

Binary 0 (0)

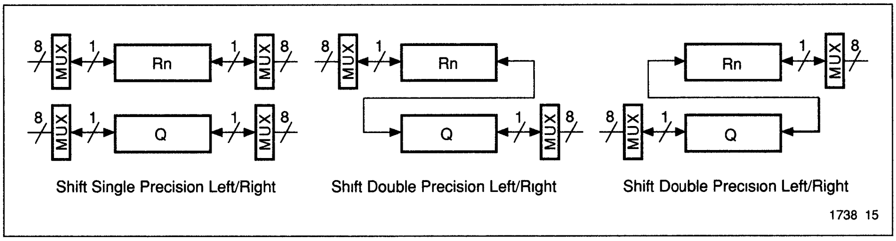
Binary 1 (1)

Most-significant bit of this register (RMSB)

Most-significant bit of the Q register (QMSB)

Serial I/O port (SDATL)

Figure 15. Shift Operations


8
MUX
1
Rn
1
MUX
8
8
MUX
1
Q
1
MUX
8
Shift Single Precision Left/Right
8
MUX
1
Rn
Q
1
MUX
8
Shift Double Precision Left/Right
Rn
1
MUX
8
8
MUX
1
Q
Shift Double Precision Left/Right
1738 15

Central Processing Unit (Con't)

Rotate Operations

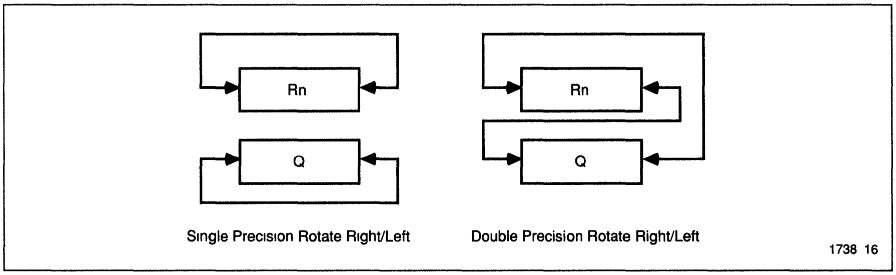
The CPU can perform the following rotate operations, as shown in Figure 16:

- ❑ Single-precision, left or right, within a general-purpose register (R0...R31, or Q).
- ❑ Double-precision, left or right, between an R0...R31 register and the Q register.

Multiple Precision Operations

The carry-out in each instruction can be used in the next instruction for multiple precision operations (e.g., ADDC). This feature enables the user to implement complex arithmetic operations such as division or multiplication in several clock cycles.

Figure 16. Rotate Operations

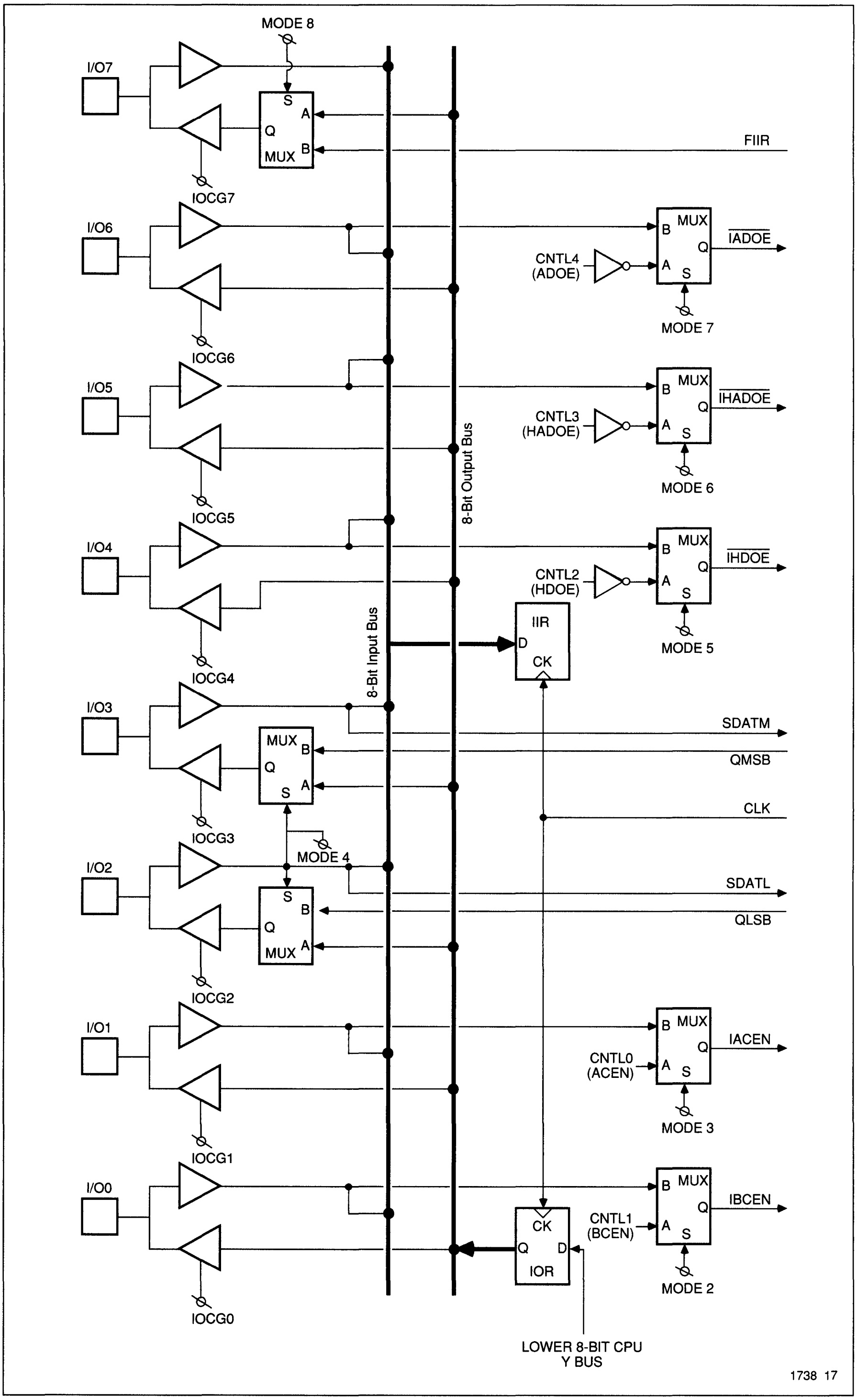

Rn
Q
Single Precision Rotate Right/Left
Rn
Q
Double Precision Rotate Right/Left
1738 16

I/O and Special Functions

The I/O bus, shown in Figure 17, consists of eight lines which can be individually programmed as inputs or outputs. These lines can also be programmed to perform Special Functions. The functions of these pins are defined by the Mode Register and I/O Configuration Register (see Configuration Register Section). The I/O and Special Functions map according to the table. The I/O lines must first be configured as inputs or outputs via the I/O Configuration Register; the Special Function option can then be enabled via the Mode Register. Individual special function control is shown in the accompanying table.

Once a Special Function has been enabled, the corresponding internal control function is automatically disabled. Conversely, when a Special Function is disabled, control of the corresponding internal control function is returned to the Control Register (see Configuration Register). Because the Inputs in the I/O Register are clocked on each cycle, the status of the special function can also be read to the CPU.

Figure 17. I/O and Special Function Bus



Configuration Registers

The Configuration Registers allow the user to control and configure different operating modes of the PAC1000. The three 10-bit Configuration Registers are the Control Register, I/O Configuration Register, and Mode Register. Each register has an associated instruction which allows individual register bits to be modified.

Control Register

The Control Register, shown in Figure 18, provides for internal control of key functions within the PAC1000 . Several of these functions can alternatively be controlled externally through the I/O bus (see I/O and Special Functions). The Control Register is modified on the falling edge of the clock.

Table 7. I/O Pins and Special Functions

Pin	Special Function	Direction	Description
I/O7	FIIR	output	FIFO Input Ready. FIFO not full.
I/O6	ADOE	input	Address Output Enable
I/O5	HADOE	input	Host Address Output Enable
I/O4	HDOE	input	Host Data Output Enable
I/O3	QMSB	bidirectional	Q Register MSB
I/O2	QLSB	bidirectional	Q Register LSB
I/O1	ACEN	input	Address Counter Enable
I/O0	BCEN	input	Block Counter Enable

Table 8. Special-Function Control

Special Function	Pin Name	I/O Configuration	Mode
FIIR	I/O7	IOCG7=1 (output)	MODE8=1
ADOE	I/O6	IOCG6=0 (input)	MODE7=1
HADOE	I/O5	IOCG5=0 (input)	MODE6=1
HDOE	I/O4	IOCG4=0 (input)	MODE5=1
QMSB	I/O3	IOCG3=1 (output)	
		IOCG3=0 (input)	MODE4=1
QLSB	I/O2	IOCG2=1 (output)	
		IOCG2=0 (input)	MODE4=1
ACEN	I/O1	IOCG1=0 (input)	MODE3 =1
BCEN	I/O0	IOCG0=0 (input)	MODE2 =1

Configuration Registers (Con't)

ASEL (CTRL9)—*Address Select.* Selects the source that will write to the Address bus:

1= Address Counter.

0= Address Output Register (AOR).

AIREN (CTRL8)—*Address Input Register Enable.* Enables and disables writing to the Address Input Register from the ADD Port:

1= Enable writing to Address Input Register (AIR).

0= Disable writing to Address Input Register (AIR).

DIREN (CTRL7)—*Data Input Register Enable.* Enables and disables writing to the Data Input Register (DIR) from the HD Port:

1= Enable writing to Data Input Register (DIR).

0= Disable writing to Data Input Register (DIR).

HDSEL1 (CTRL6) and HDSEL0 (CTRL5)—*Host Data Select.* Select the source to be connected to Host Data (HD) bus:

HDSEL1 (CTRL6)	*HDSEL0 (CTRL5)*	*Selection*
0	0	FIFO—Peripheral Mode
0	1	Data Output Register
1	0	Status Register
1	1	Program Counter

ADOE (CTRL4)—*Address Output Enable.* Selects direction of Address bus (ADD) for next clock cycle:

1= Output (see ASEL).

0= Input (see AIREN).

HADOE (CTRL3)—*Host Address Output Enable.* Selects direction of Host Address (HAD) bus for next clock cycle:

1= Output (driven from ACL Register).

0= Input (into the FIFO).

HDOE (CTRL2)—*Host Data Output Enable.* Selects Direction of Host Data (HD) bus for next clock cycle:

1= Output (See HDSEL0 and HDSEL1).

0= Input (See DIREN).

BCEN (CTRL1)—*Block Counter Enable.* Enables and disables Block Counter:

1= Enable Counting on next rising clock edge.

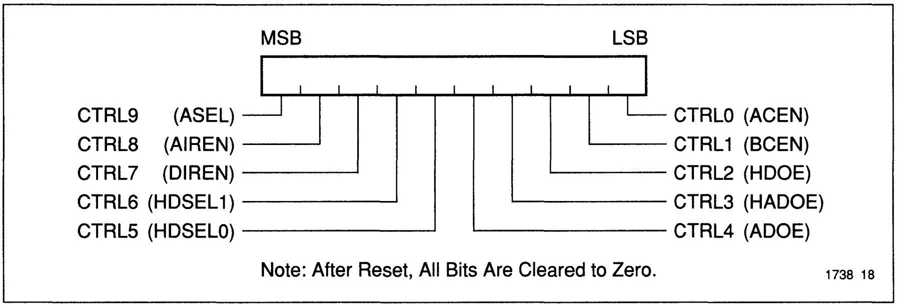
0= Disable Counting on next rising edge.

ACEN (CTRL0)—*Address Counter Enable.* Enables and disables Address Counter:

1= Enable Counting on next rising clock edge.

0= Disable Counting on next rising clock edge.

Figure 18. Control Register



Note: After Reset, All Bits Are Cleared to Zero.

Configuration Registers (Con't)

I/O Configuration Register

The I/O Configuration Register, shown in Figure 19, controls the direction of the individual lines of the I/O bus as well as configuring the Address Counter. Each I/O pin can be configured independently to be a general purpose input or output, or each can serve a special function (see I/O and Special Function). The I/O Configuration Register is also used to configure the Address Counter as a 16-bit counter with a maximum count of FFFFH or as a 22-bit counter with a maximum count of 3FFFFFH. The I/O Configuration Register is modified on the falling edge of the clock.

ACS22 (IOCG9)—Configures Address Counter as a 22- or 16-bit counter:

1= 22-bit counter.

0= 16-bit counter.

I/O7 (IOCG7)—Selects direction of I/O7 pin:

1= Output.

0= Input.

I/O6 (IOCG6)—Selects direction of I/O6 pin:

1= Output.

0= Input.

I/O5 (IOCG5)—Selects direction of I/O5 pin:

1= Output.

0= Input.

I/O4 (IOCG4)—Selects direction of I/O4 pin:

1= Output.

0= Input.

I/O3 (IOCG3)—Selects direction of I/O3 pin:

1= Output.

0= Input.

I/O2 (IOCG2)—Selects direction of I/O2 pin:

1= Output.

0= Input.

I/O1 (IOCG1)—Selects direction of I/O1 pin:

1= Output.

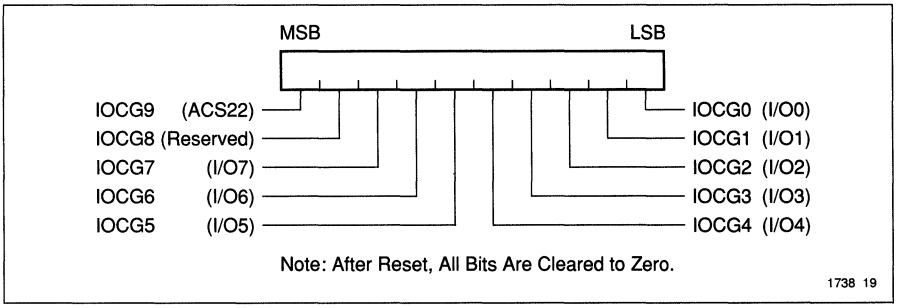
0= Input.

I/O0 (IOCG0)—Selects direction of I/O0 pin:

1= Output.

0= Input.

Figure 19. I/O Configuration Register



Configuration Registers (Con't)

Mode Register

The Mode Register, shown in Figure 20, allows the user to externally control and monitor key elements within the PAC1000 which would (alternatively) be controlled internally through the Control Register. Enabling a Special Function in the Mode Register disables the corresponding function in the Control Register. The Special Function input pins are shared with the general purpose I/O pins. The direction of the appropriate pin must be set in the I/O Configuration Register prior to programming the Mode Register.

The Mode Register can also be used to reset the FIFO as well as program the interrupt controller to generate either interrupts or Priority Test Conditions. See the discussion on "Priority Case" in the *Condition Code* section, above.

After Reset, all Mode Register bits equal zero. The Mode Register is modified on the falling edge of the clock.

The use of the Mode Register and I/O Configuration register for Special Functions is shown in the Special Function Settings table.

FIRST (MODE9)—*FIFO Reset*. (If held high, FIFO cannot receive information):

1= Initiate FIFO Reset (FIRST).

0= Complete FIFO Reset (FINRST).

FIIR (MODE8)—*FIFO Input Ready:*

1= I/O7 becomes output for the FIFO Input Ready (FIIR) flag.

0= I/O7 becomes general purpose I/O (IO7).

ADOE (MODE7)—*Address Output Enable:*

1= I/O6 becomes input for the Address Output Enable (AOE).

0= I/O6 becomes general purpose I/O (IO6).

HADOE (MODE6)—*Host Address Output Enable:*

1= I/O5 becomes input for Host Address Output Enable (HADOE).

0= I/O5 becomes general purpose I/O (IO6).

HDOE (MODE5)—*Host Data Output Enable:*

1= I/O4 becomes input for Host Data bus Output Enable HDOE).

0= I/O4 becomes general purpose I/O (IO4).

SIOEN (MODE4)—*Serial I/O Enable:*

1= I/O3 and I/O2 become MSB and LSB (respectively) of the CPU's Q register (SIO).

0= I/O3 and I/O2 become general purpose I/O ACEN(MODE3).

ACEN (MODE3)—*Address Counter Enable:*

1= I/O1 becomes input for Address Counter Enable (ACEN).

0= I/O1 becomes general purpose I/O.

BCEN (MODE2)—*Block Counter Enable:*

1= I/O0 becomes input for Block Counter Enable (BCEN).

0= I/O0 becomes general purpose I/O.

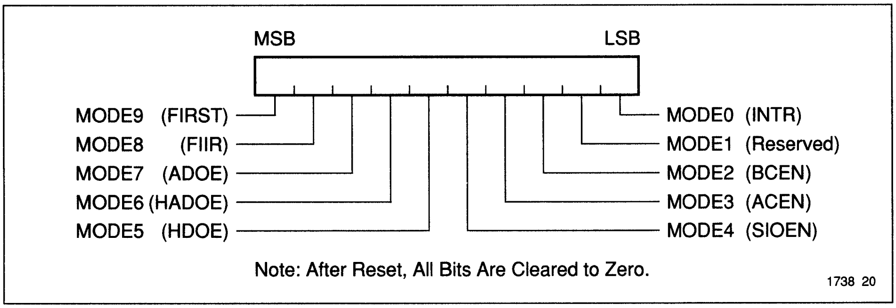
Reserved (MODE1)

INTR (MODE0)—*Interrupt/Priority-Case Mode:*

1= Select Interrupt mode (INTR).

0= Selects Priority Case mode (PCC).

Figure 20. Mode Register



State Following Reset

Whenever the PAC1000 $\overline{\text{RESET}}$ input is driven low for at least two processor clocks, the chip goes through reset. The next two tables describe the PAC1000 signal and internal register states following reset.

Table 9. Special Function Settings

Mode Bit	I/O Configuration Bit	Function
MODE8=1	IOCG7=1	FIIR flag output on I/O7
MODE7=1	IOCG6=0	$\overline{\text{ADOE}}$ provided by I/O6
MODE6=1	IOCG5=0	$\overline{\text{HADOE}}$ provided by I/O5
MODE5=1	IOCG4=0	$\overline{\text{HDOE}}$ provided by I/O4
MODE4=1	IOCG3=1	MSB of Q register output on I/O3
MODE4=1	IOCG3=0	I/O3 can be shifted into MSB of Q register or destination register
MODE4=1	IOCG2=1	LSB of Q register output on I/O2
MODE4=1	IOCG2=0	I/O2 can be shifted into LSB of Q register or destination register
MODE3=1	IOCG1=0	ACEN provided by I/O1
MODE2=1	IOCG0=0	BCEN provided by I/O0

Table 10. Signal States Following Reset

Signal	Condition
HAD[5:0]	Input
HD[15:0]	Input
IO[7:0]	Input
ADD[15:0]	Input
OC[15:0]	0000H

Table 11. Internal States Following Reset

Component	Contents
ACH Register	0
ACL Register	0
AOR Register	0
AIR Register	0
DOR Register	0
DIR Register	0
IOR Register	0
IIR Register	0
STATUS Register	0
I/O Configuration Register	0
CONTROL Register	0
Breakpoint Register	0
Mode Register	0
PC Register (Program Counter)	0
MASK Register	011111111B
BC Register	FFFFH
R31–R0 Registers	Unknown
Q Register	Unknown
LC Register	Unknown
FIFO Locations	Unknown
FIFO Flags	Empty

Electrical and Timing Specifications

Table 12. Absolute Maximum Ratings

Storage Temperature	–65°C to +150°C
Voltage to any pin with respect to GND	–0.6V to +7V
V_{PP} with respect to GND	–0.6 V to +14.0V
ESD Protection	>2000V

Stresses above those listed here may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect device reliability.

Table 13. Operating Range

Range	*Temperature*	*V_{CC}*
Commercial	0˚C to +70˚C	+5V ± 5%
Industrial	–40°C to +85°C	+5V ± 10%
Military	–55˚C to +125˚C	+5V ± 10%

Table 14. DC Characteristics

Over operating range with V_{PP}=V_{CC}

Parameter	*Symbol*	*Test Conditions*	*Min*	*Max*	*Units*
Output Low Voltage	V_{OL}	I_{OL}=8 mA		0.4	V
Output High Voltage	V_{OH}	I_{OH}=–4 mA	2.4		V
V_{CC} Standby Current CMOS	I_{SB1}	note 1		65	mA
V_{CC} Standby Current TTL	I_{SB2}	note 2		65	mA
Active Current (CMOS)	I_{CC1}	notes 1, 3			
—Commercial				130	mA
—Military				150	mA
Active Current (TTL)	I_{CC2}	notes 2, 3			
—Commercial				160	mA
—Military				180	mA
V_{PP} Supply Current	I_{PP}	V_{PP}=V_{CC}		100	µA
V_{PP} Read Voltage	V_{PP}	notes 1, 2	V_{CC}–0.4	V_{CC}	V
Input Load Current	I_{LI}	V_{IN}=5.5V or GND	–10	10	µA
Output Leakage Current	I_{LO}	V_{OUT}=5.5V or GND	–10	10	µA

Notes:
1. CMOS inputs: GND ± 0.3V or V_{CC} ± 0.3V.
2. TTL inputs: $V_{IL} \leq 0.8V$, $V_{IH} \geq 2.0V$.
3. Active current is an AC test and uses AC timing levels.

Table 15. AC Timing Levels

Inputs:	0 to 3V Reference 1.5V
Outputs:	0.4 to 2.4V

Table 16. AC Characteristics

Parameter	Symbol	12MHz[1]		16MHz[1]		20MHz[2]	
		Min	Max	Min	Max	Min	Max
CLOCK CYCLE							
Cycle Time	t_{CK}	84		62.5		50	
Clock Pulse Width High	t_{CKH}	26		24		21	
Clock Pulse Width Low	t_{CKL}	26		24		21	
HOST READ CYCLE							
Read Cycle Time	t_{RC}	50		40		30	
Address to Data Valid	t_{ACC}		45		35		30
$\overline{CS}$ to Data Valid	t_{CS}		45		35		30
$\overline{CS}$ to tristate	t_{CSZ}	0	45	0	35	0	30
HOST WRITE CYCLE							
Pulse width of $\overline{CS}$ and $\overline{WR}$ LOW	t_{PWL}	20		15		15	
Pulse width of $\overline{CS}$ and $\overline{WR}$ High	t_{PWH}	15		10		10	
Data setup to $\overline{WR}$	t_{SD}	10		10		5	
Data hold to $\overline{WR}$	t_{HD}	10		10		5	
RESET CYCLE							
$\overline{RESET}$ setup	t_{SR}	10		10		5	
$\overline{RESET}$ to tristate of ADD, HAD, HD, I/O	t_{RZ}	25		25		20	
$\overline{RESET}$ clocked to OUTCNTL low	t_{ROL}	30		30		25	
ADDRESS TIMING							
Address/Data setup	t_{SADD}	10		10		10	
Address/Data hold	t_{HADD}	8		8		5	
Clocked Counter to Address output	t_{CADD}		43		35		30
Clocked Address Register to Address	t_{RADD}		43		35		30
ADOE enable to data valid	t_{ADOE}		50		40		30
HADOE enable to data valid	t_{HADOE}		50		40		30
Address output disable	t_{CKZ}	0	25	20		0	16

Table 16. AC Characteristics (Con't)

Parameter	Symbol	12MHz[1]		16MHz[1]		20MHz[2]	
		Min	Max	Min	Max	Min	Max
DATA AND I/O TIMING							
Clock to I/O Output Valid	t_{CKIO}		35		30		30
Clock to HD Output	t_{CKHD}		35		30		30
IO data setup	t_{SIO}	10		10		10	
IO data hold	t_{HIO}	8		8		5	
HD data setup	t_{SHD}	10		10		10	
HD data hold	t_{HHD}	8		8		5	
HDOE enable to data valid	t_{HDOE}		50		40		30
Bus Output Disable	t_{CKZ}	0	25	0	20	0	16
TEST AND INTERRUPT TIMING							
Condition Code setup	t_{SCC}	60		50		40	
Condition Code hold	t_{HCC}	0		0		0	
Clock to OUTCNTL Valid	t_{COV}		33		33		25
Minimum interrupt pulse for acceptance	t_{IPWA}	15		10		10	
SPECIAL FUNCTION TIMING (I/O Bus)							
SQ15 setup	t_{SSQ15}	15		10		10	
SQ15 hold	t_{HSQ15}	0		0		0	
SQ0 setup	t_{SSQ0}	15		10		10	
SQ0 hold	t_{HSQ0}	0		0		0	
Clock to Q0 output	t_{CKQ0}		35		30		30
Clock to Q15 output	t_{CKQ15}		35		30		30
Address Counter enable setup	t_{SACEN}	20		15		10	
Address Counter enable hold	t_{HACEN}	10		5		5	
Block Counter enable setup	t_{SBCEN}	20		15		10	
Block Counter enable hold	t_{HBCEN}	10		5		5	
External output enable to data valid	t_{SFV}		30		25		20
External output enable to high impedance	t_{SFZ}		30		25		20

Notes:
1. Operating temperature range: Commercial, Industrial, Military
2. Operating temperature range: Commercial

Figure 21. Clock Cycle Timing

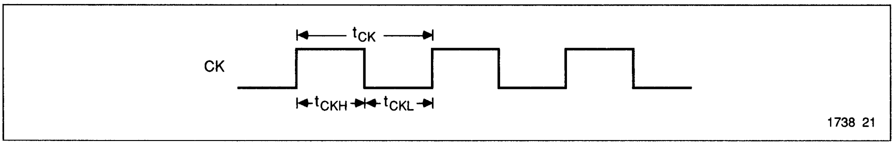


Figure 22. Host Read Cycle Timing

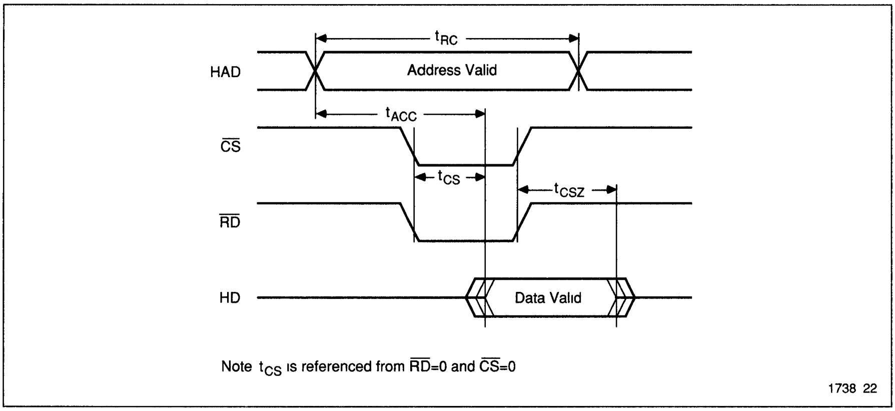


Figure 23. Host Write FIFO Cycle Timing

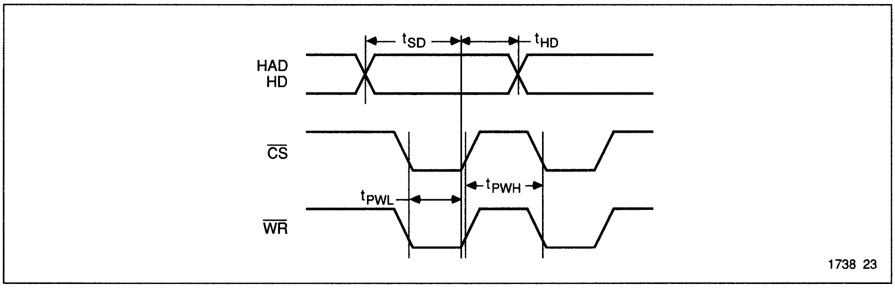


Figure 24. Reset Cycle Timing

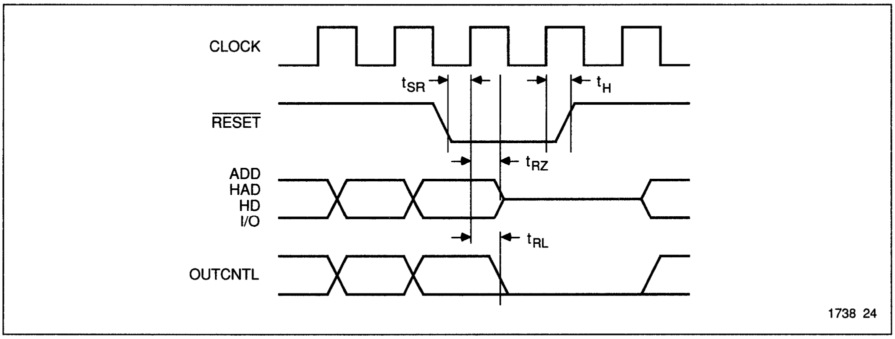
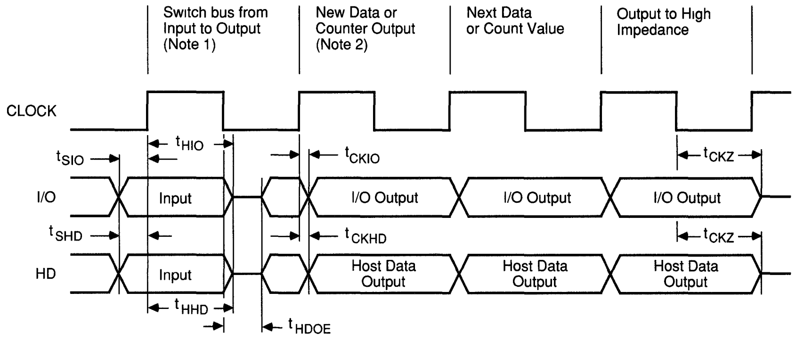


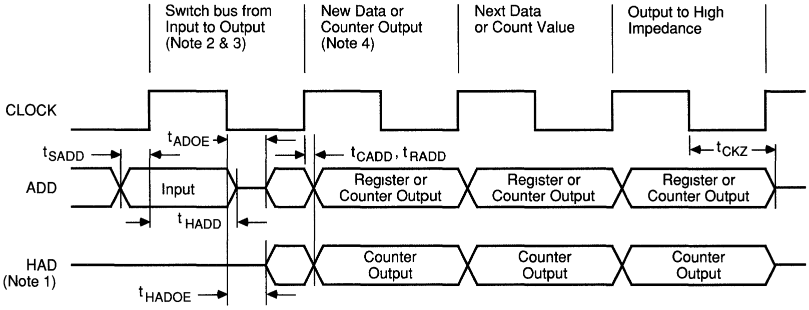
Figure 25. Data and I/O Timing



Notes 1 A bus directional change (input-to-output or output-to-input) takes place on the falling edge of the clock
2 New data or count value is latched on the rising edge of the clock

1738 25

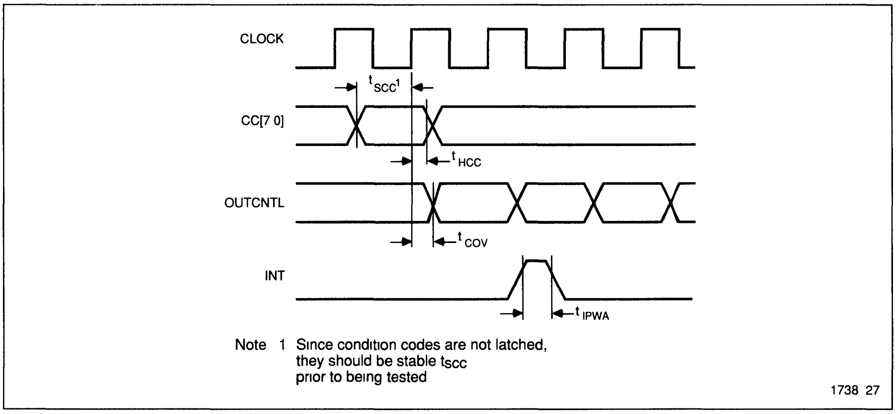
Figure 26. Address Timing


Switch bus from Input to Output (Note 2 & 3)
New Data or Counter Output (Note 4)
Next Data or Count Value
Output to High Impedance
CLOCK
tSADD
tADOE
tCADD, tRADD
tCKZ
ADD
Input
Register or Counter Output
Register or Counter Output
Register or Counter Output
tHADD
HAD (Note 1)
Counter Output
Counter Output
Counter Output
tHADOE

Notes 1 The Host Address (HAD) bus is used to output the lower six bits of the 22-bit counter
2. A bus directional change takes place on the falling edge of the clock (input-to-output or output-to-input)
3 Selection of the source to be output on a bus occurs on the falling edge of the clock (i e , counter or address register)
4. New data or count value is latched on the rising edge of the clock

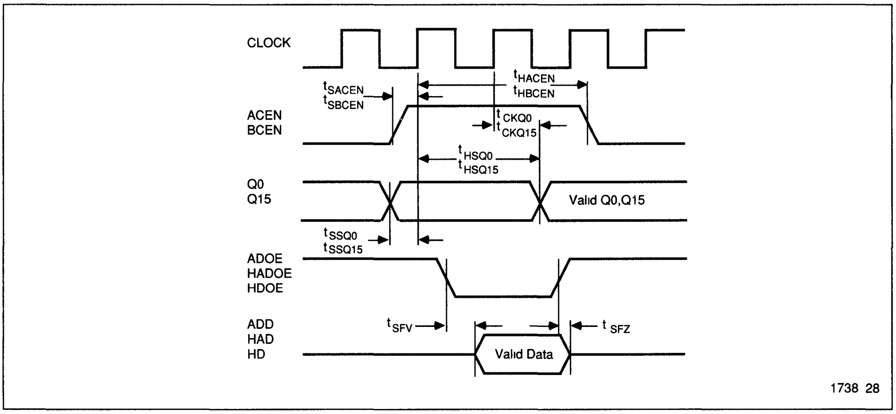
1738 26

Figure 27. Test and Interrupt Timing



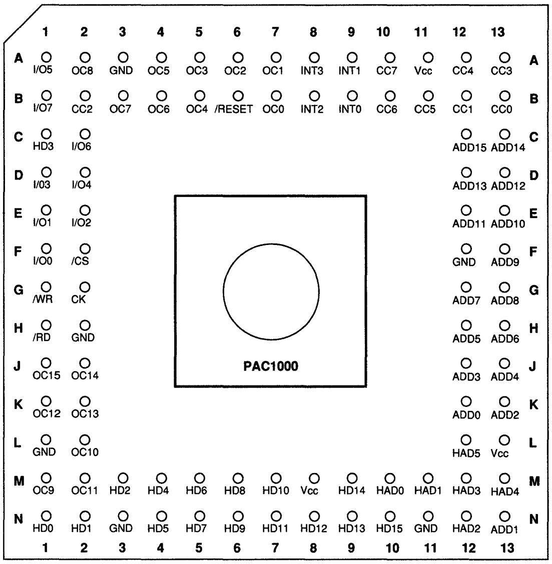
Note 1 Since condition codes are not latched, they should be stable tSCC prior to being tested

Figure 28. Special Function Timing



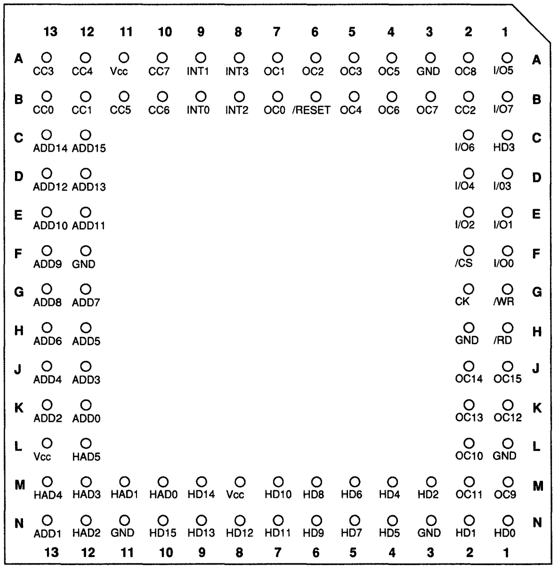
Pin Assignments

Figure 29. 88-Pin Ceramic PGA Pin Assignments



	1	2	3	4	5	6	7	8	9	10	11	12	13	
A	I/O5	OC8	GND	OC5	OC3	OC2	OC1	INT3	INT1	CC7	Vcc	CC4	CC3	A
B	I/O7	CC2	OC7	OC6	OC4	/RESET	OC0	INT2	INT0	CC6	CC5	CC1	CC0	B
C	HD3	I/O6										ADD15	ADD14	C
D	I/O3	I/O4										ADD13	ADD12	D
E	I/O1	I/O2										ADD11	ADD10	E
F	I/O0	/CS										GND	ADD9	F
G	/WR	CK										ADD7	ADD8	G
H	/RD	GND										ADD5	ADD6	H
J	OC15	OC14										ADD3	ADD4	J
K	OC12	OC13										ADD0	ADD2	K
L	GND	OC10										HAD5	Vcc	L
M	OC9	OC11	HD2	HD4	HD6	HD8	HD10	Vcc	HD14	HAD0	HAD1	HAD3	HAD4	M
N	HD0	HD1	GND	HD5	HD7	HD9	HD11	HD12	HD13	HD15	GND	HAD2	ADD1	N
	1	2	3	4	5	6	7	8	9	10	11	12	13	

TOP (THROUGH PACKAGE) VIEW



	13	12	11	10	9	8	7	6	5	4	3	2	1	
A	CC3	CC4	Vcc	CC7	INT1	INT3	OC1	OC2	OC3	OC5	GND	OC8	I/O5	A
B	CC0	CC1	CC5	CC6	INT0	INT2	OC0	/RESET	OC4	OC6	OC7	CC2	I/O7	B
C	ADD14	ADD15										I/O6	HD3	C
D	ADD12	ADD13										I/O4	I/O3	D
E	ADD10	ADD11										I/O2	I/O1	E
F	ADD9	GND										/CS	I/O0	F
G	ADD8	ADD7										CK	/WR	G
H	ADD6	ADD5										GND	/RD	H
J	ADD4	ADD3										OC14	OC15	J
K	ADD2	ADD0										OC13	OC12	K
L	Vcc	HAD5										OC10	GND	L
M	HAD4	HAD3	HAD1	HAD0	HD14	Vcc	HD10	HD8	HD6	HD4	HD2	OC11	OC9	M
N	ADD1	HAD2	GND	HD15	HD13	HD12	HD11	HD9	HD7	HD5	GND	HD1	HD0	N
	13	12	11	10	9	8	7	6	5	4	3	2	1	

BOTTOM VIEW

1738 29

Table 17. PGA Pin Assignments

Name	Pin	Name	Pin	Name	Pin
$\overline{\text{CS}}$	F2	GND	H2	I/O0	F1
$\overline{\text{RD}}$	H1	GND	L1	I/O1	E1
$\overline{\text{RESET}}$	B6	GND	A3	I/O2	E2
$\overline{\text{WR}}$	G1	GND	F12	I/O3	D1
ADD0	K12	GND	N3	I/O4	D2
ADD1	N13	GND	N11	I/O5	A1
ADD10	E13	HAD0	M10	I/O6	C2
ADD11	E12	HAD1	M11	I/O7	B1
ADD12	D13	HAD2	N12	INT0	B9
ADD13	D12	HAD3	M12	INT1	A9
ADD14	C13	HAD4	M13	INT2	B8
ADD15	C12	HAD5	L12	INT3	A8
ADD2	K13	HD0	N1	OC0	B7
ADD3	J12	HD1	N2	OC1	A7
ADD4	J13	HD10	M7	OC10	L2
ADD5	H12	HD11	N7	OC11	M2
ADD6	H13	HD12	N8	OC12	K1
ADD7	G12	HD13	N9	OC13	K2
ADD8	G13	HD14	M9	OC14	J2
ADD9	F13	HD15	N10	OC15	J1
CC0	B13	HD2	M3	OC2	A6
CC1	B12	HD3	C1	OC3	A5
CC2	B2	HD4	M4	OC4	B5
CC3	A13	HD5	N4	OC5	A4
CC4	A12	HD6	M5	OC6	B4
CC5	B11	HD7	N5	OC7	B3
CC6	B10	HD8	M6	OC8	A2
CC7	A10	HD9	N6	OC9	M1
CK	G2			VCC	A11
				VCC	L13
				VCC	M8

Figure 30. 100-Pin Plastic or Ceramic Quad Flatpack (Gullwing) Pin Assignments

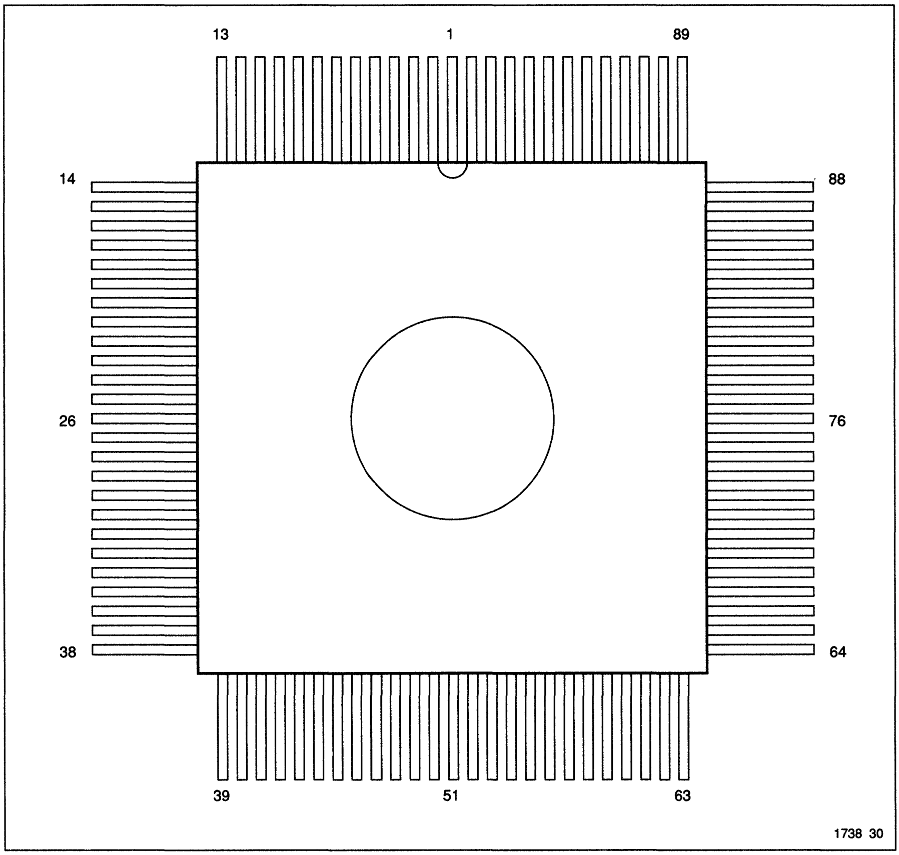


Table 18. Plastic or Ceramic Quad Flatpack (Gullwing) Pin Assignments

Pin	Name	Pin	Name	Pin	Name	Pin	Name
1	$\overline{\text{RD}}$	26	HD11	51	ADD7	76	OC1
2	GND	27	HD12	52	ADD8	77	OC2
3	GND	28	VCC	53	ADD9	78	$\overline{\text{RESET}}$
4	OC15	29	VCC	54	GND	79	N/C
5	OC14	30	HD13	55	GND	80	OC3
6	OC12	31	HD14	56	ADD10	81	OC4
7	OC13	32	HD15	57	ADD11	82	OC5
8	GND	33	HAD0	58	ADD12	83	OC6
9	GND	34	GND	59	ADD13	84	GND
10	OC10	35	GND	60	ADD14	85	GND
11	OC9	36	HAD1	61	ADD15	86	OC7
12	OC11	37	HAD2	62	CC0	87	OC8
13	N/C	38	N/C	63	CC1	88	CC2
14	HD0	39	HAD3	64	CC3	89	IO5
15	HD1	40	ADD1	65	CC4	90	IO7
16	HD2	41	HAD4	66	CC5	91	IO6
17	GND	42	HAD5	67	VCC	92	HD3
18	GND	43	VCC	68	VCC	93	IO4
19	HD4	44	VCC	69	CC6	94	IO3
20	HD5	45	ADD0	70	CC7	95	IO2
21	HD6	46	ADD2	71	INT0	96	IO1
22	HD7	47	ADD3	72	INT1	97	$\overline{\text{CS}}$
23	HD8	48	ADD4	73	INT2	98	IO0
24	HD9	49	ADD5	74	INT3	99	CK
25	HD10	50	ADD6	75	OC0	100	$\overline{\text{WR}}$

Instruction Set Overview

The PAC1000 architecture can perform three operations simultaneously in each instruction cycle. The operations are specified in the System Entry Language (PACSEL) using a single statement. PACSEL instructions can perform three operations:

- ❑ Program Control (PROGCNTL)
- ❑ CPU
- ❑ Output Control (OUTCNTL)

Each *instruction* is executed in a single cycle; the three *operations* are executed in parallel.

The syntax of a PACSEL statement has a label and three components:

```
[label:] PROGCNTL, CPU,
OUTCNTL;
```

The PROGCNTL component determines program flow and determines the next statement to be executed; the CPU component determines which operation is to be performed by the CPU; the OUTCNTL component determines the state of the control outputs.

A comma (,) is used to separate the instructions and a semi-colon marks the end of a statement. In general, each statement is executed in a single cycle.

In PACSEL statements, the PROGCNTL, CPU, OUTCNTL components can come in any order or any combination of Macro or Assembler operators. That is, you may mix Assembler operators among Macro operators. Tables at the end of this section summarize the Macro and Assembler operators.

In some cases, the same mnemonic is used to specify identical operations in both Macro and Assembler level.

You may:

- ❑ Specify all the components in the same statement in order to perform the operations in parallel:

  ```
  PROGCNTL, CPU, OUTCNTL;
  ```

- ❑ Specify components one at a time:

  ```
  CPU;
  PROGCNTL;
  OUTCNTL;
  ```

- ❑ Use components in any combination:

  ```
  PROGCNTL, CPU;
  PROGCNTL, OUTCNTL;
  CPU, OUTCNTL;
  ```

WSI recommends that, in general, you maintain a consistent ordering of these components and consistent groupings of Assembler-level and Macro operators, e.g. in separate files. This manual uses the PROGCNTL, CPU, OUTCNTL ordering.

When PROGCNTL is omitted, the implied instruction is CONTinue, that is, proceed to the next control instruction. When CPU is omitted, the implied instruction is NOP. When OUTCNTL is omitted, the implied instruction is MAINTain, that is, maintain the most recent OUTCTL, in Assembler order.

A summary of PACSEL Assembler and Macro statements follows.

Table 19. PACSEL Assembler Language Summary

Mnemonic	Arguments	Meaning
The PROGCNTL Operators		
ACSIZE	<16/22>	SET A COUNTER SIZE
CALL	<LABEL | LCPTR | FIFO>	UNCOND BRANCH SUBRTN
CALLC	<COND> <LABEL | FIFO>	COND BRANCH SUBRTN
CALLNC	<COND> <LABEL | FIFO>	INV COND BRANCH SUBRTN
CCASE	<CG> <VALUE>	BRANCH SUBRTN CASEBLK
CLI	<MASK>	CLEAR INTERRUPT
CONT(D)		CONTINUE
CPI	<VALUE>	PRIORITIZED SUB RTN
DI	<MASK>	DISABLE INTERRUPT
DSS		DISABLE SINGLE STEP MODE
EI	<MASK>	ENABLE INTERRUPT
ESS		ENABLE SINGLE STEP MODE
JCase	<CG> <VALUE>	UNCOND BRANCH CaseBLK
JMP	<LABEL | LCPTR | FIFO>	UNCONDITIONAL BRANCH
JMPC	<COND> <LABEL | FIFO>	CONDITIONAL BRANCH
JMPNC	<COND> <LABEL | FIFO>	INVERT COND BRANCH
JPI	<VALUE>	PRIORITIZED BRANCH
LDBP	<VALUE>	LOAD BP REG
LDBPD		LOAD BP COMP VALUE
LDLC	<VALUE>	LOAD COUNTER
LDLCD		LOAD CTR COMPUTED VAL
LOOPNZ	<LABEL>	REPEAT BRANCH,CNTRNZ
PLDLC	<VALUE>	PUSH VALUE & LDCTR
PLDLCD		PUSH VAL&LDCTR CM VL
POP		POP STACK
POPLC		POP STACK TO CNTR
PUSHLC		PUSH CNTR
RESTART		BRANCH TO 0
RET		RETURN
RC	<COND>	CONDITIONAL RETURN
RNC	<COND>	INV COND RETURN
RSTCON	<MASK>	RESET CONTROL REG
RSTIO	<MASK>	RESET I/O CONFIG REG
RSTMODE	<MASK>	RESET MODE REG
SETCON	<MASK>	SET CONTROL REG
SETIO	<MASK>	SET I/O CONFIG REG
SETMODE	<MASK>	SET MODE REG

Table 19. PACSEL Assembler Language Summary (Con't)

Mnemonic	*Arguments*	*Meaning*
The CPU Operators		
ADC	<ARG1> <ARG2> [<ARG3>]	ADD WITH CARRY
ADD	<ARG1> <ARG2> [<ARG3>]	ADD
AND	<ARG1> <ARG2> [<ARG3>]	BITWISE AND
CMP	<ARG1> <ARG2>	COMPARE
DEC	<ARG1> [<ARG2>]	DECREMENT
INC	<ARG1> [<ARG2>]	INCREMENT
INV	<ARG1> [<ARG2>]	INVERT
MOV	<DEST> <SRC>	MOVE SRC TODEST
NOP(D)		NO OPERATION
OR	<ARG1> <ARG2> [<ARG3>]	BITWISE OR
RDFIFO		READ FIFO DATA TO REG
SBC	<ARG1> <ARG2> [<ARG3>]	SUB WITH CARRY
SHLRQ	<REG> <RARG> <QARG>	SHIFT LEFT REG & Q
SHLR	<REG> <RARG>	SHIFT LEFT REG
SHRRQ	<REG> <RARG> <QARG>	SHIFT RIGHT REG & Q
SHRR	<REG> <RARG>	SHIFT RIGHT REG
SUB	<ARG1> <ARG2> [<ARG3>]	SUBTRACT
XOR	<ARG1> <ARG2> [<ARG3>]	EXCLUSIVE OR
XNOR	<ARG1> <ARG2> [<ARG3>]	EXCLUSIVE NOR
The MACRO Operators		
DIV	<ARG1> <ARG2> <ARG3>	DIVISION
MUL	<ARG1> <ARG2> <ARG3>	2'S COMP MULTIPLY
The OUTCNTL Operators		
MAINT(D)		MAINTAIN PREV VALUE
OUT	<VALUE>	OUTPUT

Table 20. PACSEL Macro Language Summary

The PROGCNTL Operators

```
ACSIZE <16/22>
CALL <label | LCPTR | FIFO> [ON] [NOT] [<cond>]
CASE n, PROGCNTL, CPU, OUTCNTL;
CLEAR <int level> [...<int level>]
CONFIGURE <pm1> [<pm2>...<pm10>]
CONT
DISABLE <int level> [<int level>...<int level>]
ELSE
ENABLE <int level> [<int level>...<int level>]
ENDFOR
ENDIF
ENDPSWITCH
ENDSWITCH
ENDWHILE
FOR <value>
GOTO <label | LCPTR | FIFO> [ON] [NOT] [<cond>]
IF [NOT] <cond>
INPUT <i/o pin> [<i/o pin>...<i/o pin>]
LOADBP <value>
OUTPUT <i/o pin> [<i/o pin>...<i/o pin>]
PRIORITY m, PROGCNTL, CPU, OUTCNTL;
PSWITCH
RESET <p1> [<p2>...<p10>]
RETURN [ON] [NOT] [<cond>]
SET <p1> [<p2>...<p10>]
SWITCH <case group>
WHILE [NOT] <cond>
```

Table 20. PACSEL Macro Language Summary (Con't)

The CPU-Operator Assignment

```
move
    <dest> := <src>
arithmetic expression
    <dest> := <arg1> <+/-> <arg2> <+/-> <arg3>
logical expression
    <dest> := <arg1> <logical operator> <arg3>
increment, decrement, invert, unary minus
    <dest> := <opr> <src>
macro expression
    <dest> := <arg1> [* | /] <arg2>
shift RAM
    <Rx> = Rx <shft opr> <shft arg>
shift RAM and q
    <QRx> = Q <shft opr> <shft arg> Rx <shft opr> <shft arg>
```

The OUTCNTL Operator

```
OUT <arg1> [<arg2>...<arg16>]
```

System Development Tools

PAC1000 System Development Tools are a complete set of PC-based development tools. They provide an integrated easy-to-use software and hardware environment to support PAC1000 development and programming.

The tools run on an IBM-XT, AT, PS2 or compatible computer running MS-DOS version 3.1 or later. The system must be equipped with 640 Kbytes of RAM and a hard disk.

Hardware

The PAC1000 System Programming Hardware consists of:

- ❑ WS6000 MagicPro Memory and PSD Programmer (XT, AT only)
- ❑ Package Adaptors (88-Pin Ceramic Pin-Grid Array and 100-Pin Ceramic Quad Flatpack—Gullwing) for the MagicPro Remote Socket Adaptor Unit

The MagicPro Programmer is the common hardware platform for programming all WSI programmable products. It consists of the IBM-PC plug-in Programmer Board and the Remote Socket Adaptor Unit.

Software

The PAC1000 System Development Software consists of the following:

- ❑ WISPER Software—PSD Software Interface
- ❑ IMPACT Software—Interface Manager for PAC1000
- ❑ PACSEL Software—System Entry Language
- ❑ PACSIM Software—Functional Simulator
- ❑ PACPRO Software—Device Programming Software

WISPER and IMPACT software provide a menu-driven user interface enabling other tools to be easily invoked by the user.

The system design is entered into PACSEL source program files using an editor chosen by the user. PACSEL supports assembly-level and high-level Macro programming.

The PACSEL Assembler produces object code format in single or multiple modules, which are then linked by the PACSEL Linker into a single load file with a format suitable for PACSIM and PACPRO.

The PACSIM functional simulator enables the user to test and debug programs by examining the state of PAC1000 internal registers before and during a complete functional simulation of the device.

PACPRO software programs PAC1000 devices by using the MagicPro hardware and the socket adapter.

The programmed PAC1000 is then ready to be used.

Support

WSI provides a complete set of quality support services to registered owners. These support services include the following:

- ❑ 12-month Software Updates.
- ❑ Hotline to WSI Application Experts—For direct design assistance.
- ❑ 24-Hour Electronic Bulletin Board—For design assistance via dial-up modem.

Training

WSI provides in-depth, hands-on workshops for the PAC1000 and the System Development Tools. Workshop participants will learn how to develop and program their own high-performance microcontrollers. Workshops are held at the WSI facility in Fremont, California.

Ordering Information— PAC1000

Part Number	Speed (MHz)	Package Type	Package Drawing	Operating Temperature	Manufacturing Procedure
PAC1000-12F*	12	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Commercial	Standard
PAC1000-12FI*	12	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Industrial	Standard
PAC1000-12FM*	12	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Military	Standard
PAC1000-12FMB*	12	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Military	MIL-STD-883C
PAC1000-12Q*	12	100-Pin Plastic Quad Flatpack, Gullwing	Q1	Commercial	Standard
PAC1000-12X	12	88-Pin Ceramic Pin-Grid Array	X1	Commercial	Standard
PAC1000-12XI	12	88-Pin Ceramic Pin-Grid Array	X1	Industrial	Standard
PAC1000-12XM	12	88-Pin Ceramic Pin-Grid Array	X1	Military	Standard
PAC1000-12XMB	12	88-Pin Ceramic Pin-Grid Array	X1	Military	MIL-STD-883C
PAC1000-16F*	16	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Commercial	Standard
PAC1000-16FI*	16	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Industrial	Standard
PAC1000-16FM*	16	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Military	Standard
PAC1000-16FMB*	16	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Military	MIL-STD-883C
PAC1000-16Q*	16	100-Pin Plastic Quad Flatpack, Gullwing	Q1	Commercial	Standard
PAC1000-16X	16	88-Pin Ceramic Pin-Grid Array	X1	Commercial	Standard
PAC1000-16XI*	16	88-Pin Ceramic Pin-Grid Array	X1	Industrial	Standard
PAC1000-16XM*	16	88-Pin Ceramic Pin-Grid Array	X1	Military	Standard
PAC1000-16XMB*	16	88-Pin Ceramic Pin-Grid Array	X1	Military	MIL-STD-883C
PAC1000-20F*	20	100-Pin Ceramic Quad Flatpack, Gullwing	F3	Commercial	Standard
PAC1000-20X*	20	88-Pin Ceramic Pin-Grid Array	X1	Commercial	Standard
PAC1000-20Q*	20	100-Pin Plastic Quad Flatpack, Gullwing	Q1	Commercial	Standard

*: These products are advanced information.

Ordering Information—System Development Tools

Part Number	Contents
PAC1000-GOLD	WISPER Software IMPACT Software PACSEL Software PACSIM Software PACPRO Software User's Manual WSI-Support WS6000 MagicPro Programmer
PAC1000-SILVER	WISPER Software IMPACT Software PACSEL Software PACSIM Software PACPRO Software User's Manual WSI-Support
WS6000	MagicPro Programmer IBM PC plug-in Adaptor Card Remote Socket Adaptor
WS6010	88-Pin CPGA Adaptor Used with the WS6000 MagicPro Programmer
WS6012	100-Pin Ceramic Quad Flatpack (Gullwing) Adaptor Used with the WS6000 MagicPro Programmer
WSI-Support	Support Services, including: □ 12-month Software Update Service □ Hotline to WSI Application Experts □ 24-hour access to WSI Electronic Bulletin Board
WSI-Training	Workshops at WSI, Fremont, CA For details and scheduling, call PSD Marketing, (415) 656-5400

Programmable System™ Device

SAM448 Introduction

User-Configurable Microsequencer

Overview

In 1988 WSI introduced a new concept in programmable VLSI: the Programmable System™ Device (PSD). The PSD is defined as a family of *User-configurable system level building blocks on-a-chip enabling quick implementation of application specific controllers and peripherals.* The first generation PSD series includes the MAP168, a User-Configurable Peripheral with Memory; the SAM448, a User-Configurable Microsequencer; and the PAC1000, a User-Configurable Microcontroller.

The SAM448 is a microsequencer intended for use in digital systems that require events to be controlled at high speed. A microsequencer is basically an instruction oriented device executing one internal instruction on each system cycle. This can be done in a linear flow or the sequencer can test the state of logic inputs or internal events and respond to program branching on a result. In addition, it has the capability of driving output signals on a cycle by cycle basis.

The SAM448 can operate at a high clock speed (30 MHz) so sequential operations can be performed much faster than with lower end microcontrollers. A classic application of the SAM448 would be in the generation of pulse waveforms for video line and frame synchronization with blanking output controls for both line and frame flyback. The device could also control the load and shift activity in the video output registers and supervise the video memory address counters. All these activities are sequential in nature so microcode could be developed for the SAM device and programmed into the device's on-chip EPROM.

Prior to the development of the SAM448 Microsequencer, a designer would most likely develop a system from discrete EPROM or ROM plus 74LS TTL logic with dedicated LIFO and registers. The actual development of such a design would escalate in chip count to eventually cover an entire printed circuit card. With the advent of Programmable Logic Devices (PLDs), the development of a microsequencing circuit became simpler. However, a typical system still required five to six PLDs. In addition, and EPROM was needed to hold the microcode. Because microcode is usually rather wide, a number of EPROMs were needed.

The SAM448 provides the optimum solution when implementing a microsequencer of medium complexity. It has been designed to be cascadable in width and depth so more complex microsequencer designs may be achieved.

Microcode EPROM Architecture

The core of the SAM448 is a microcode EPROM organized as a 448 locations deep and 36 bits wide. On each clock cycle, the current 32 bit wide instruction is clocked into the pipeline register. The 32 bit word is split into a number of fields. The F field consists of 16 bits and drives the output lines as user defined output pins. The remaining 20 bits are subdivided into one 8-bit Q field which generally directs processing to the next address of the EPROM. The 8-bit D field can be used to hold a constant or direct value but it can also be used for next address generation. The OP field is three bits in width and contains the current instruction to be executed. The remaining field is the

Microcode EPROM Architecture (Cont.)

E field and performs a 3-State control function on the pipeline register. When HIGH, the output pins are enabled and when LOW the outputs are in a high impedance state. This feature enables one SAM448 device to share the same outputs with a second for vertical cascading.

The EPROM locations are connected such that the first 192 locations (0 to 191) are in a linear sequence. The remaining locations are organized in four rows of 192 to 255. This permits a one of four branch control. The internal branch control logic will make the decision as to which branch to take depending on the state of the user defined inputs and the value of the next state address.

Branch Control Logic

The branch control logic determines the location from where the next instruction will be fetched. The next address can come from the Q or D field of the instruction currently in the pipeline register, the top of the stack or LIFO or the Branch Select EPLD. The Branch Select EPLD can be programmed to view inputs or the logical combination of inputs to invoke a branch when a logic state becomes true.

Stack

The stack or Last In First Out (LIFO) memory is 15 locations deep and 8 bits wide and can be used to hold the value of a return address so successful CALL to and RETURN from subroutines may be invoked. A loop counter is included in the SAM448 architecture and the stack can be used to hold the contents of this loop counter when nested loops are invoked. The eight input lines may also be pushed onto the stack to externally load the counter.

Loop Counter

To make provision for a number of operations to be repeated a defined number of times, a loop counter called CREG has been included in the design. This eight bit counter is loaded from the D field by a dedicated instruction LOADC or from the stack in the case of nested loops. The counter decrements to zero and then holds at zero. So repetitive routines may be achieved by a LOOPNZ instruction.

Instruction Set

The instruction set for the SAM448 consists of 12 instructions to handle multiway branching, subroutines, nested for-next loops and dispatch functions. With only 12 instructions a designer can become familiar with creating SAM448 designs very quickly. The WSI State Machine Input Language (ASMILE) support software enables designs to be generated quickly and efficiently.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

SAM448

User-Configurable Microsequencer

Features

- □ First Generation Programmable System Device
- □ User-Programmable Microsequencer for Implementing High-Performance State Machines
- □ On-Chip Reprogrammable EPROM Microcode Memory Up to 448 Words Deep
- □ 15 × 8 Stack
- □ Loop Counter
- □ Prioritized, Multi-Way Control Branching
- □ 8 General-Purpose Branch Control Inputs
- □ 16 General-Purpose Control Outputs
- □ Cascadable to Expand Outputs or States
- □ Low-Power CMOS Technology
- □ Footprint Efficient 28 Pin 300 Mil Dip or 28 Lead CLDCC/PLDCC Package
- □ 30 MHz Minimum Clock Frequency
- □ High Level PC-XT/AT, PS2 or Compatible Design Support Software (SAM+PLUS):
  - — WSI PSD Integrated Software Environment
  - — State Machine Input Language
  - — Microcode Assembler
  - — Functional Simulator

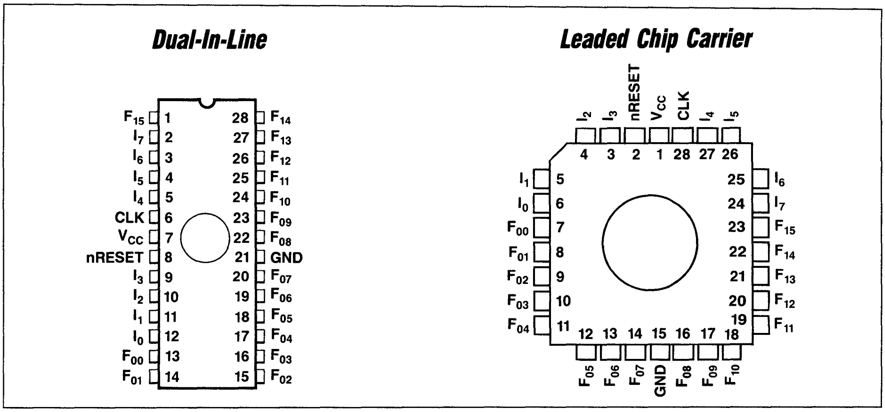
Description

In 1988 WSI introduced a new concept in programmable VLSI, Programmable System™ Devices (PSD). The PSD is defined as a family of *User-configurable system level building blocks on-a-chip enabling quick implementation of application specific controllers and peripherals.* The first generation PSD series includes the MAP168, a User-Configurable Peripheral with Memory; the SAM448, a User-Configurable Microsequencer; and the PAC1000, a User-Configurable Microcontroller.

The SAM448 is a first generation PSD and is WSI's first user programmable microsequencer. On-chip EPROM (up to 448 words) is integrated with Branch Control Logic, Pipeline Register, Stack, and Loop Counter. This generic microcoded architecture provides an efficient vehicle for implementing a broad range of high performance controllers spanning the spectrum from basic state machines to traditional bit-slice controller applications.

The SAM448 has eight general purpose input pins, a clock pin and a reset pin.
It has 16 user-definable outputs packaged in a 28-pin 300 mil Dip or 28 Lead CLDCC/ PLDCC package. One-Time-Programmable plastic versions are available to minimize volume production costs.

Pin Configuration (Top View)



Dual-In-Line

Leaded Chip Carrier

Description (Cont.)

Programming the SAM448 device is accomplished on a standard WSI PSD WISPER development system installed with the optional SAM+PLUS software package and device adapters. New users can purchase a separate WISPER-SAM development system with programming hardware included. SAM+PLUS allows designs to be entered in either state machine or microcoded formats. SAM+PLUS automatically performs logic minimization and design fitting for the device. The design may then be simulated or programmed directly to achieve customized working silicon within minutes.

Using WSI's proprietary high performance CMOS EPROM technology allows SAM448 to operate at a 25-MHz typical clock frequency while still enjoying the benefits of low CMOS power consumption. This technology also facilitates 100% generic testability which eliminates the need for post-programming testing.

Ideal application areas for SAM448 include programmable sequence generators (state machines), bus and memory control functions, graphics and DSP algorithm controllers, and other complex, high performance machines. The devices may be cascaded easily to obtain greater output requirements (horizontal cascade) or greater microcode memory depth (vertical cascade) or both.

SAM as a State Machine

The SAM448 architecture allows easy implementation of synchronous state machines. SAM's internal EPROM memory together with its Pipeline Register allows storage of up to 448 unique states. SAM's Branch Control Logic allows single clock, multi-way branching in response to the eight inputs, current device state, and user-defined transition conditions. Design entry is simplified with WSI's State Machine Input Language (ASMILE) supported by the SAM+PLUS development system. This high level language uses IF-THEN-ELSE statements to define state transitions and a truth table to define or tri-state the outputs on a state-by-state basis.

SAM as a Microcoded Sequencer

SAM's architecture has several advanced features that enable it to be used as a sophisticated microcoded sequencer. SAM's on-chip EPROM (448 words) is integrated with a microcoded sequencer consisting of Branch Control Logic, Stack, and Loop Counter. The eight general-purpose inputs, the Counter, the Stack, and the Pipeline Register feed the Branch Control Logic. The Branch Control Logic gives flexible multi-way microcode branch capability in a single clock, enhancing throughput beyond that of conventional controllers or sequencers.

SAM+PLUS development software offers high level microcode entry featuring a compact assortment of powerful instructions (OP-codes) allowing easy implementation of conditional branches, subroutine calls, multiple level for-next loops, and dispatch functions (branching to an externally specified address).

Functional Description

The SAM architecture is shown in Figure 1. The primary elements are the Microcode EPROM, 36-bit Pipeline Register, Branch Control Logic, 15 × 8-bit Stack, and 8-bit Loop Counter.

The Branch Control Logic generates the address of the next state and applies this address to the Microcode Memory. The outputs of the Microcode Memory represent the user-defined outputs and internal control values associated with the next state. On the leading edge of the clock these new values are clocked into the Pipeline Register and become the current state. The new values in the Pipeline Register—along with the Counter, Stack and Inputs—are used by the Branch Control Logic to generate the new next-state address.

Microcode EPROM and Pipeline Register

The Microcode EPROM is organized into 448, 36-bit words or locations, each of which can be viewed as a single state. 16 of these bits (the F-field) are available at device pins as user-defined outputs.

The other 20 bits are internal control signals that are divided into 4 fields: the 8-bit Q-field normally provides the next-state address; the 8-bit D-field is a general purpose field used either as a constant or as an alternative next-state address; the OP-field contains the instruction; and, the

Functional Description (Cont.)

E-field contains a single bit which enables or tri-states the device outputs.

As shown in Figure 2, the Microcode Memory is organized as 256 rows or addresses. Addresses 0 through 191 contain a single 36-bit word which is associated with the desired next-state. This state information will be clocked into the Pipeline Register on the next rising edge of the clock and the outputs will become valid one T_{CO} (clock to output delay) later.

Addresses 192–255, on the other hand, access four unique 36-bit words which correspond to four possible next states. (The extension .0, .1, .2, and .3 are used to distinguish those four states.) These 64 addresses are known as Multi-Way Branch locations and are used to perform single clock 4-way branches. Whenever the next-state address falls within the Multi-Way Branch locations, the Branch Control Logic will make the necessary 1-of-4 selection based on the next-state address and user-defined input conditions.

Figure 1. SAM448 Block Diagram

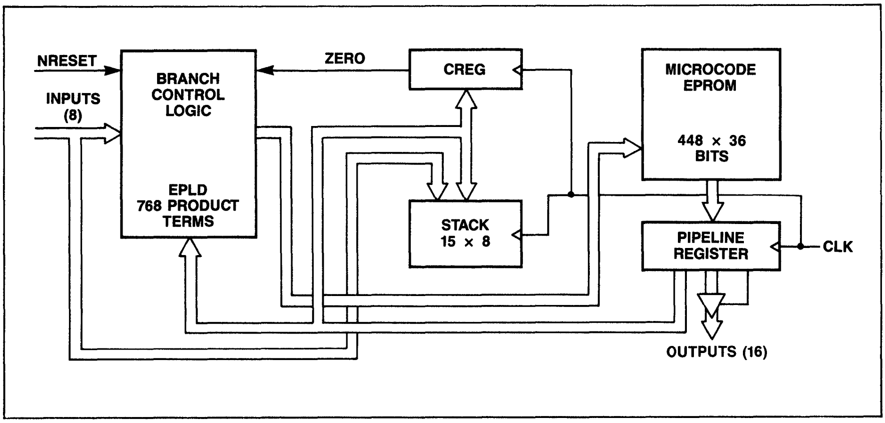


Figure 2. SAM Microcode Memory

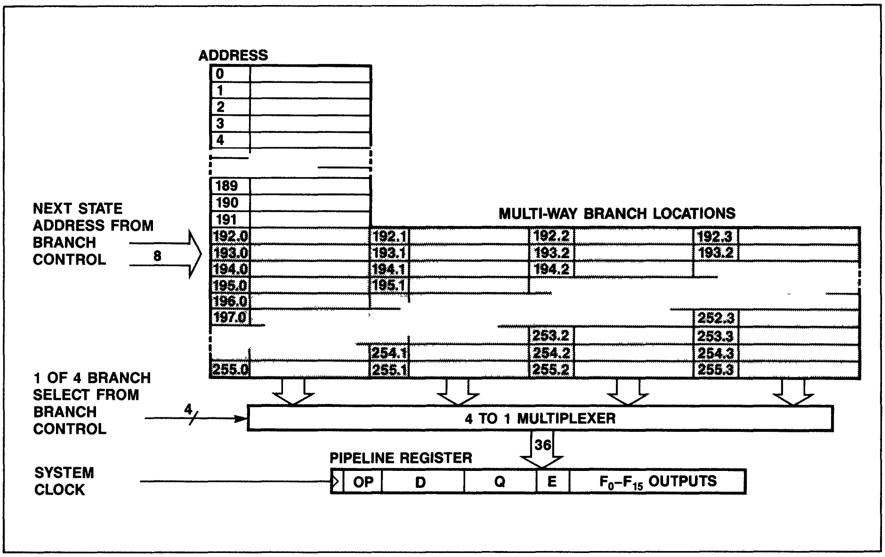
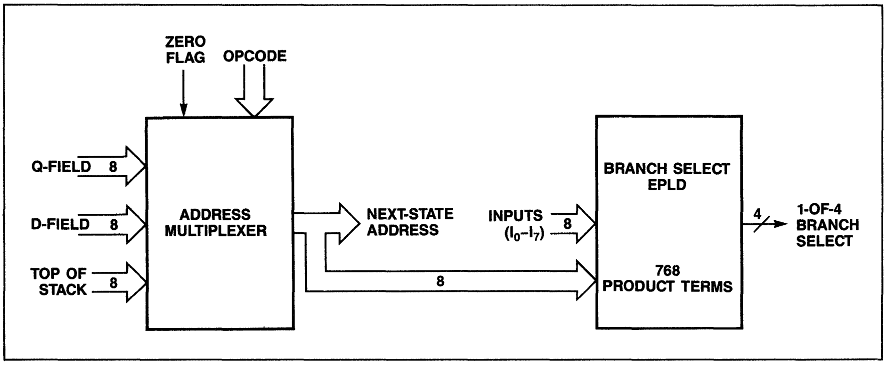


Figure 3. SAM Branch Control Logic



Branch Control Logic Block

At the heart of the high-performance sequencing ability of the SAM family is the Branch Control Logic. This block determines the next-state to be clocked into the Pipeline Register based on the current status of the Pipeline Register, the Counter, the Stack, and the eight input pins.

The Branch Control Logic is divided into two segments: the Address Multiplexer and the Branch Select EPLD.

The Address Multiplexer provides the next-state address to the Microcoded Memory. The next-state address can come from the Q-field, the D-field, or the Top-of-Stack. The selection between these three resources is based on the instruction in the Pipeline Register and the condition of the Zero Flag from the Counter.

The Branch Select EPLD is used to perform up to a 4-way branch based on user-defined input conditions. This block is a 768 product-term programmable logic device with 16 inputs and four outputs. When the next-state address falls within the multi-way branch block of memory (any address greater than 191) the Branch Select EPLD performs the necessary 1-of-4 selection. When the next-state address is less than 192, the Branch Select EPLD is turned off since no selection is required.

The conditions controlling the multi-way branch are defined by the user with a simple IF, THEN, ELSE format like the following:

```
    IF (cond3) THEN select 201.3
ELSEIF (cond2) THEN select 201.2
ELSEIF (cond1) THEN select 201.1
ELSE                select 201.0
```

The conditions are prioritized so that if the first condition is not met (cond3), then microword 201.3 will be selected and clocked into the Pipeline Register regardless of the results of cond2 and cond1. If none of the three conditions are met, then the microword 201.0 will be clocked into the Pipeline Register.

The three conditional expressions are user defined and may contain any logical equation based on the inputs that can be reduced to four product-terms. For example, the expression

```
 I1 * /I2 * /I4
+I3 * /I4 * /I5 * /I6 * /I7
+I0
+I2 * /I4 * /I5
```

contains four product-terms and is a valid condition. There is a unique set of 12 product-terms for each of the 64 multi-way branch locations for a total of 768 product-terms. (See Figure 4.)

The SAM448 has been designed so that the number of available product-terms should never be the limiting factor on a design. Prioritization provides an effective product-term count of more than 12 per location. A trade-off between number of product-terms and number of possible branches can be made by simply placing identical state information in two locations as shown in Figure 5.

Figure 4. SAM Branch Logic for Address 192 Through 255

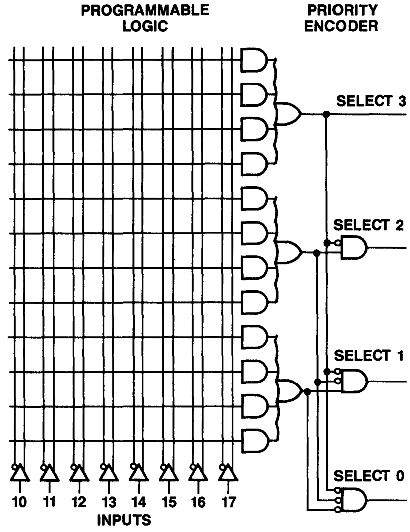
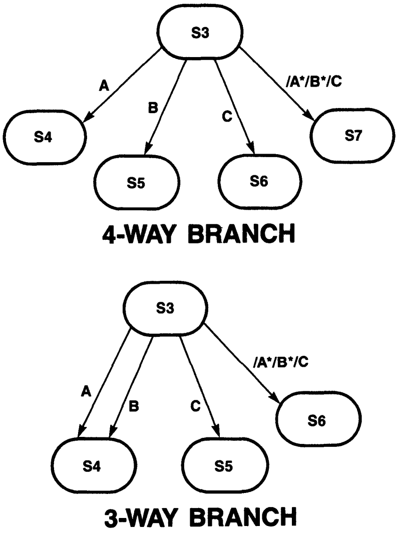


Figure 5. Multi-Way Branching



Functional Description (Cont.)

Stack

The Stack of the SAM448 is a Last In First Out (LIFO) arrangement consisting of 15 8-bit words. The Top-of-Stack may be used as the next-state address or popped into the Counter. Values may be pushed onto the stack either from the D-field in the Pipeline Register or from the Counter enabling efficient implementation of subroutines, nested loops, and other iterative structures. The eight input lines may also be pushed onto the stack to allow external address specification in a dispatch function or to externally load the counter.

The PUSHing or POPing of the stack occurs on the leading edge of the clock. The stack is "zero filled" so that a POP from an empty stack will return all eight bits set to zero. On the other hand, a push to an already full stack will write over the Top-of-Stack leaving the other 14 values unchanged.

Loop Counter

The SAM448 contains an 8-bit Loop Counter, referred to as the Count Register (CREG), which is useful for controlling timing loops and affecting a variety of branch operations. The CREG is a down counter and may be loaded directly from the D-field of the Pipeline Register or from the Top-of-Stack. The value of the CREG may be saved and restored by pushing and popping it to and from the Stack.

The CREG is loaded or decremented on the leading edge of the clock. It is designed so that it will not decrement once it reaches zero to prevent roll-over. A Zero Flag indicates when the counter has reached zero and is used with the LOOPNZ command to control program flow (see Instruction Set Description). Single instruction delay loops are easily constructed and, in combination with the Stack, nested loops or delays of arbitrary length may be generated.

Instruction Set

The instruction set of the SAM448 consists of a compact assortment of powerful commands. Assembly language constructs allow efficient implementation of multi-way branching, subroutines, nested for-next loops, and dispatch functions. The complete instruction set is described at the end of this data sheet. These instructions are only used with assembly language design entry and are automatically supplied when using the WSI State Machine Input Language (ASMILE).

Output Enable Control

Each microcode word contains an OE bit (the E-field) which enables the outputs when E = 1 and causes a high-impedance when E = 0. These bits are accessible through high-level constructs in the WSI Development Software. This capability allows the vertical cascading of SAM448 devices to increase the number of states.

nRESET Pin

The nRESET pin acts as a master reset for the SAM448 causing it to empty the Stack, clear the Counter, and load the microword found at address 0 into the Pipeline Register. The nRESET signal is useful for system reset or for synchronizing several SAMs that are cascaded vertically or horizontally.

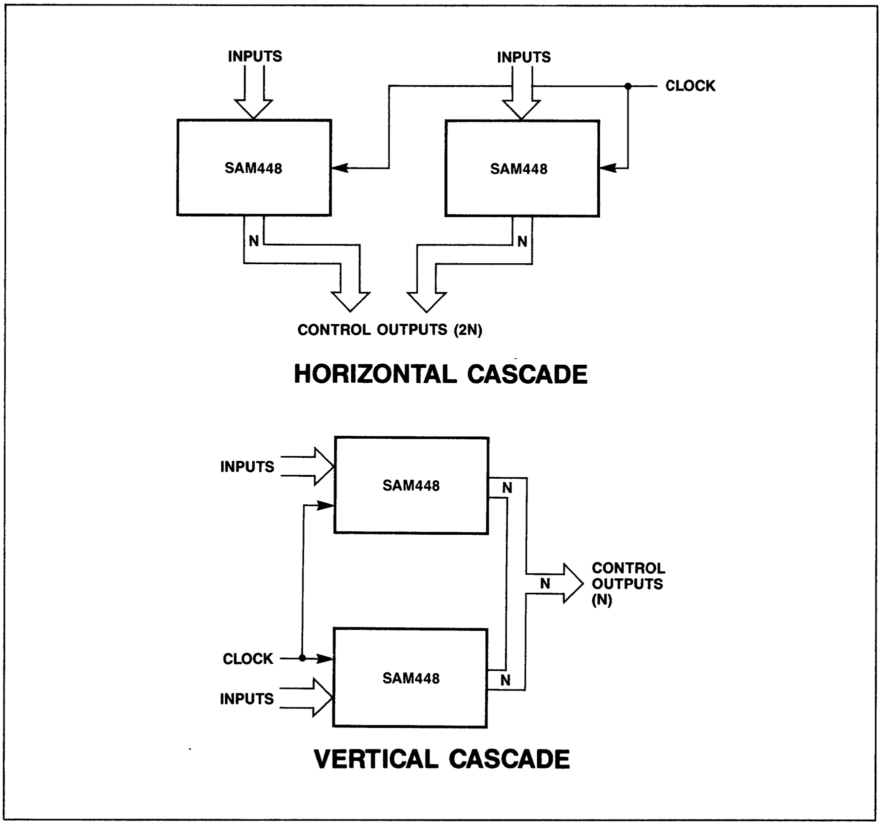
The nRESET signal must be held low for at least three clock rising edges to perform a valid clear. A nRESET of one clock rising edge causes the SAM448 to enter into a supervisor mode and a nRESET of two clock edges results in an undefined state.

The outputs of the boot address (00 Hex) will appear at the pins from the fourth clock edge after nRESET goes low, until the third clock edge after nRESET returns to high.

Horizontal and Vertical Cascading

Just as with memory and bit slice devices, the SAM devices can be cascaded to provide greater functionality. If an application requires more output lines, two or more SAMs can be cascaded horizontally. Likewise, if an application requires more states, two or more SAMs can be cascaded vertically. In either case, no speed penalty is incurred. Designs utilizing horizontal cascading are fully supported by the SAM+PLUS development software. Vertical cascading requires the designer to make certain tradeoffs to split the design.

Figure 6. SAM448 Cascading



Functional Testing

The SAM448 is fully functionally tested and guaranteed through complete testing of each programmable EPROM bit and all internal logic elements thus ensuring 100% programming yield.

The erasable nature of the SAM448 allows test programs to be used and then erased during early stages of production flow. This facility to use application-independent, general purpose tests is called generic testing and is unique among user-defined LSI logic devices. The devices also contain on board test circuitry to allow verification of function and AC specification once encapsulated in non-windowed packages.

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply Voltage	Note 6	4.75 (4.5)	5.25 (5.5)	V
V_I	Input Voltage		0	V_{CC}	V
V_O	Output Voltage		0	V_{CC}	V
T_R	Input Rise Time (Note 6)			500 (100)	ns
T_F	Input Fall Time (Note 6)			500 (100)	ns

DC Operating Characteristics

V_{CC} = 5V ± 5%, 0°C to +70°C for Commercial
V_{CC} = 5V ± 10%, −40°C to +85°C for Industrial
V_{CC} = 5V ± 10%, −55°C to +125°C for Military

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High Level Input Voltage		2.0		V_{CC}+0.3	V
V_{IL}	Low Level Input Voltage		−0.3		0.8	V
V_{OH}	High Level TTL Output Voltage	I_{OH} = −8 mA DC	2.4			V
V_{OH}	High Level CMOS Output Voltage	I_{OH} = −4 mA DC	3.84			V
V_{OL}	Low Level TTL Output Voltage	I_{OL} = 8 mA (4 mA) DC			0.45	V
I_I	Input Leakage Current	V_I = V_{CC} or GND			±10	µA
I_{OZ}	3-State Output Off-State Current	V_O = V_{CC} or GND			±10	µA
I_{CC1}	V_{CC} Supply Curent (Standby) (Note 6)	V_1 = V_{CC} or GND I_0 = 0 CLK = V_{CC}		30	65 (90)	mA
I_{CC2}	V_{CC} Supply Current (Active) (Note 6)	No Load 50% CLK f = 20 MHz		55	120 (170)	mA

Absolute Maximum Ratings

(See Design Recommendations)

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply Voltage	With Respect to GND (Note 2)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage		−2.0	14.0	V
V_I	DC Input Voltage		−2.0	7.0	V
I_{CCMAX}	DC V_{CC} or GND Current		−250	250	mA
I_{OUT}	DC Output Current, per Pin		−25	25	mA
P_D	Power Dissipation			1200	mW
T_{STG}	Storage Temperature	No Bias	−65	150	°C
T_{AMB}	Ambient Temperature	Under Bias	−10	85	°C

Capacitance

(Note 3)

Symbol	Parameter	Conditions	Typ	Unit
C_{IN}	Input Capacitance	V_{IN} = 0V f = 1.0 MHz	10	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V f = 1.0 MHz	15	pF
C_{CLK}	Clock Pin Capacitance	V_{IN} = 0V f = 1.0 MHz	10	pF
C_{RST}	nRESET Pin Capacitance		75	pF

AC Characteristics

V_{CC} = 5V ± 5%, 0°C to +70°C for Commercial
V_{CC} = 5V ± 10%, −40°C to +85°C for Industrial
V_{CC} = 5V ± 10%, −55°C to +125°C for Military (Note 7)

Symbol	Parameter	Conditions	SAM448-30		SAM448-25		SAM448-20		Unit
			Min	Max	Min	Max	Min	Max	
f_{CYC}	Maximum Frequency	C_1 = 35 pF	30		25		20		MHz
t_{CYC}	Minimum Clock Cycle			33.3		40		50	ns
t_{SU}	Input Setup Time		16.5		20		22		ns
t_H	Input Hold Time		0		0		0		ns
t_{CO}	Clock to Output Delay	C_1 = 35 pF		16.5		20		22	ns
t_{CZ}	Clock to Output Disable or Enable			16.5		20		22	ns
t_{CL}	Minimum Clock Low Time		11		12		15		ns
t_{CH}	Minimum Clock High Time		11		12		15		ns
t_{SUR}	nRESET Setup Time		16.5		18		18		ns
t_{HR}	nRESET Hold Time		5		5		5		ns

NOTES:
1. Typical values are for T_A = 25°C, V_{CC} = 5V.
2. Minimum DC input is −0.3V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns.
3. Capacitance measured at 25°C. Sample tested only.
4. If the nRESET is held low for more than 3 clock edges, then the outputs associated with the boot address (00 Hex) will remain at the pins until the third clock edge after nRESET goes high.
5. For 1.0 < V_1 < 3.8, the nRESET pin will source up to 200 µA.
6. Figures in ( ) pertain to military and industrial temperature versions.
7. The specifications noted above apply to military operating range devices. MIL-STD-883 compliant product specifications are provided in military product drawings available on request from WSI marketing at Tel. 415-656-5400. These military product drawings should be used for the preparation of source control drawings.

Figure 7. Timing Waveforms

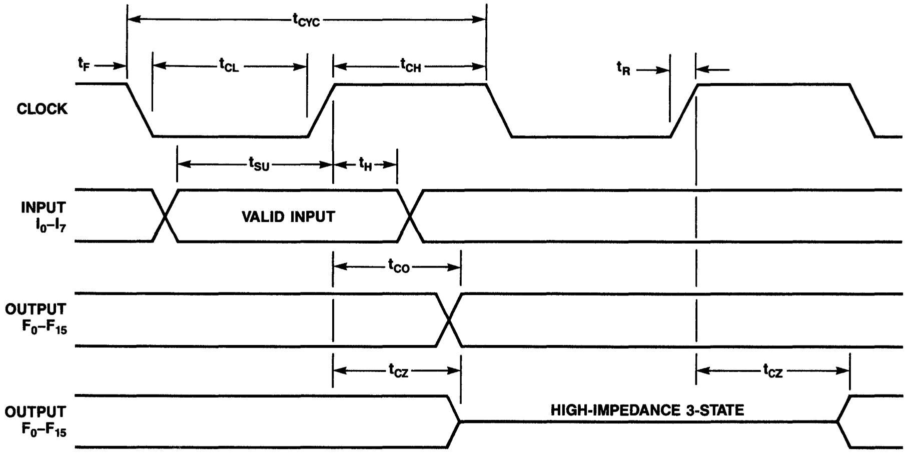
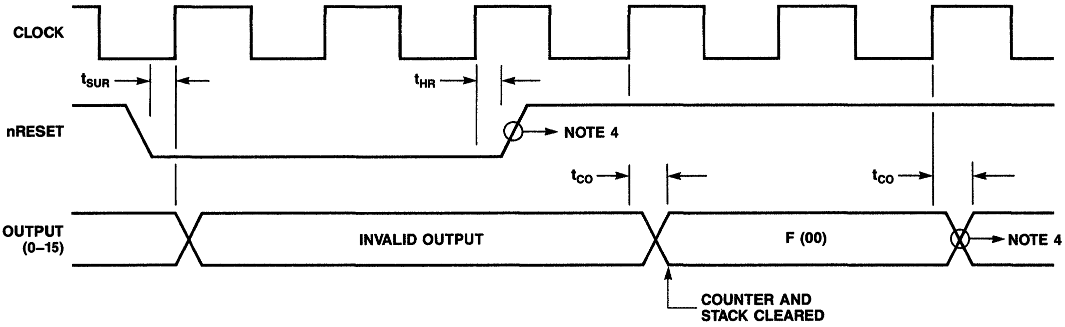


Figure 8. Reset Timing Waveforms



Design Security

The SAM448 contains a programmable design security feature that controls the access to the data programmed into the device. If this programmable feature is used, a proprietary design implemented in the device cannot be copied nor retrieved. This enables a high level of design control to be obtained since programmed data within EPROM cells in invisible. The bit that controls this function, along with all other program data, may be reset simply by erasing the device.

Design Recommendations

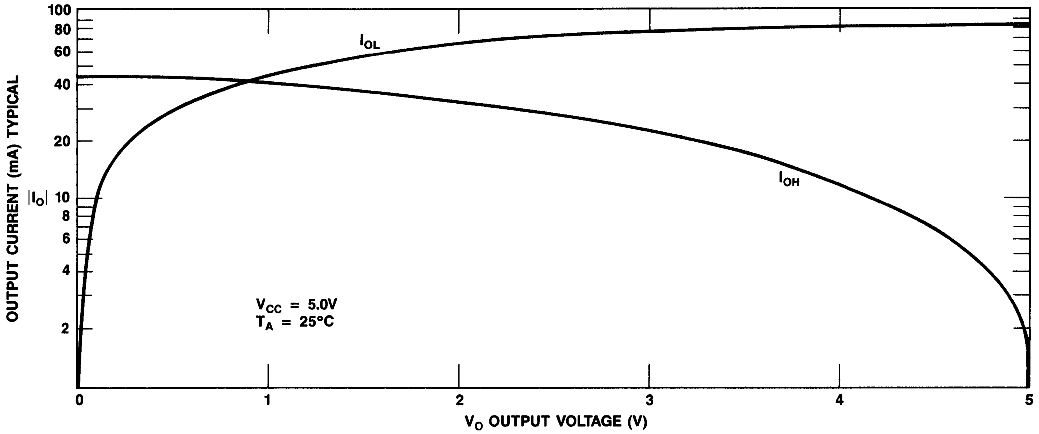
Operation of the SAM448 with conditions above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only, and functional operation of the device at these or any other conditions above those indicated in the operational sections of this data sheet is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. These devices contain circuitry to protect the input against damage to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high-impedance circuit.

For proper operation, it is recommended that opaque labels be placed over the device window. Input and output pins must be constrained to the range GND $\leqslant$ (V_{IN} or V_{OUT}) $\leqslant$ V_{CC}. Unused inputs must always be tied to an appropriate logic level (e.g., either V_{CC} or GND). A power supply decoupling capacitor of at least 0.1 µF must be connected directly between the V_{CC} pin and GND.

When operating in noisy environments it is possible that a glitch on the nRESET pin one T_{SUR} before the clock edge could initiate a supervisor mode. To prevent this possibility, it is recommended to connect a capacitor of at least 0.1 µF from the nRESET input to ground.

All general purpose inputs to the SAM448 should be synchronized to be guaranteed to meet the setup time. Input transitions which occur less than one T_{SU} before the leading clock edge can cause the SAM448 to enter an undefined state.

Figure 9. Output Drive Current



Instruction Set Description

Following is a description of the instruction set available with the SAM448. These instructions can be used in conjunction with the Assembly Language entry to access the various features of the SAM448. They are automatically supplied when using the WSI State Machine Input Language (ASMILE).

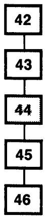
In the following description label1 and label2 indicate arbitrary labels located in the assembly (.ASM) file. These labels will be converted by the software into the 8-bit address of that label. The parameter constant is any 8-bit number (0–255 Decimal, 0–FF Hex) representing an address, a mask, or a constant.

The instructions influence the control of the Stack, the Counter, and the Address Multiplexer. These effects are summarized in the Instruction Table. Throughout the examples it is assumed for simplicity that the destination labels do not lie within the Multi-Way Branch Block of memory so that branching based on inputs is not performed. It is valid, however, for any of these labels to lie within the Multi-Way Branch Block so that 4-way branching based on the inputs can be performed. See the MULTI-WAY BRANCH section at the end of this data sheet for more details.

The SAM+PLUS development system allows the designer to use the high level Assembly Language without worrying about the actual values that are placed in the various fields.

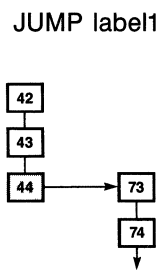
CONTINUE simply causes execution to continue with the next sequential instruction found in the Assembly Language file (.ASM).

CONTINUE

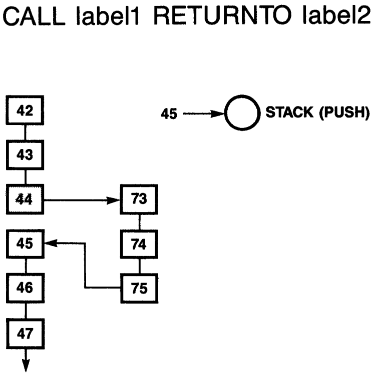


Instruction Set Description (Cont.)

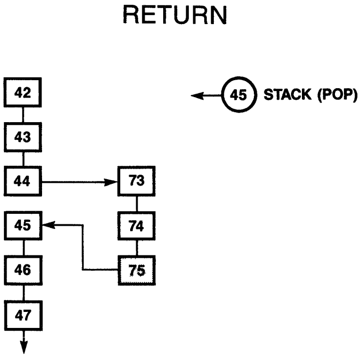
The JUMP instruction causes execution to branch to the indicated location. If address 44 contains the instruction ‘JUMP label1,’ then the next state will come from label1 which in this case is located at address 73.



The CALL/RETURNTO instruction is typically used to call a subroutine. In general it will push the address of label2 onto the Stack and cause label1 to be the next-state address. Leaving the RETURNTO designation off will cause label2 to default to the next instruction in the .ASM file. In the example, address 44 contains the command ‘CALL label1’ where label1 is located at address 73. This causes the address of the following instruction, in this case 45, to be pushed onto the Stack, and the next state to come from address 73. The RETURN command at address 75 returns the execution to address 45.



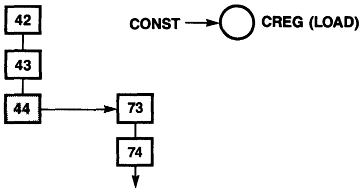
The RETURN command is used to return from a subroutine call or in general to cause the next-state address to come from the top of the Stack. In the example, the command at address 44 CALLed the subroutine at address 73 and PUSHed the value 45 onto the Stack. The RETURN command at address 75 will transfer execution to address 45 and POP that value off the Stack.



Instruction Set Description (Cont.)

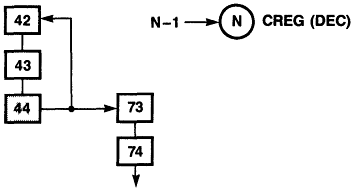
The LOAD Counter command loads the Counter with the value specified and transfers execution to label1. The LOADC command is typically used to initialize the Counter for a repetitive loop. In the example, address 44 has the command 'LOADC 73D GOTO label1' which causes the decimal value 73 to be loaded into the Counter and the next state to come from label1. In this case label1 is located at address 73. If the GOTO designation is left off label1 will default to the next instruction in the .ASM file.

LOADC constant GOTO label1



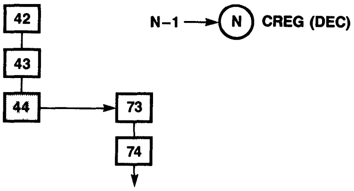
The LOOP on Non-Zero/ON ZERO goto command jumps to one of two addresses based on the value of the Zero Flag and decrements the Counter if not zero. This instruction is typically used to implement for-next loops. In the example, address 44 has the command 'LOOPNZ label1 ONZERO label2' where label1 is located at address 42 and label2 is located at address 73. If the Counter is not at zero then the next state will come from address 42 and the Counter will be decremented. If the Counter is already at zero then the instruction at address 73 will be executed and the Counter will stay at zero. If the ONZERO designation is left off, the default for label2 will be the next instruction in the .ASM file.

LOOPNZ label1 ONZERO label2



The DEcrement Counter on Non-Zero GOTO command will decrement the Counter if it is non-zero and jump to label1. In the example, address 44 has the command 'DECNZ GOTO label1' where label1 is located at address 73. The Counter is decremented and the next instruction comes from address 73. The default for label1 is the next instruction in the .ASM file.

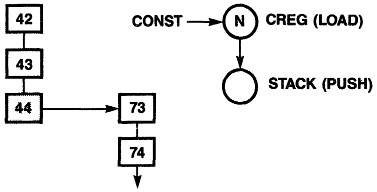
DECNZ GOTO label1



Instruction Set Description (Cont.)

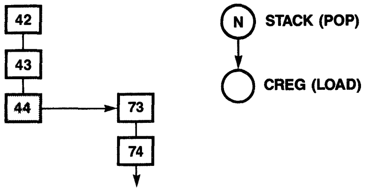
The PUSH Counter LOAD Counter command will push the current value of the Counter onto the Stack, load a constant into the Counter, and jump to label1. This instruction is useful for implementing nested for-next loops. In the example, the instruction at address 44 is 'PUSHLOADC 153D GOTO label1' where label1 is located at address 73. The value in the Counter will be pushed onto the Stack, the decimal value 153 will be loaded into the Counter, and the next instruction will come from address 73. The default for label1 is the next instruction in the .ASM file.

PUSHLOADC constant GOTO label1



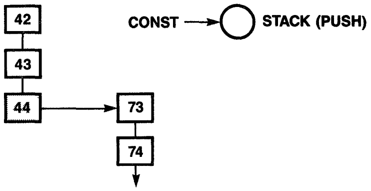
The POP Stack to Counter GOTO command will pop the top of Stack into the Counter and jump to label1. This command is typically used in conjunction with the PUSHLOADC to implement nested for-next loops. In the example, address 44 has the command 'POPC GOTO label1' where label1 is located at address 73. The default for label1 is the next instruction in the .ASM file.

POPC GOTO label1



The PUSH constant to Stack GOTO command will push the value constant onto the Stack and jump to label1. In the example, address 44 has the command 'PUSH 34D GOTO label1' where label1 is located at address 73. The decimal value 34 is pushed onto the Stack and the next state comes from address 73. The default for label1 is the next instruction in the .ASM file.

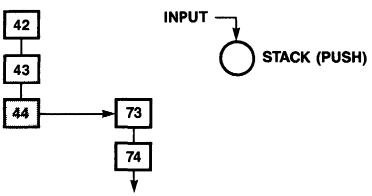
PUSH constant GOTO label1



Instruction Set Description (Cont.)

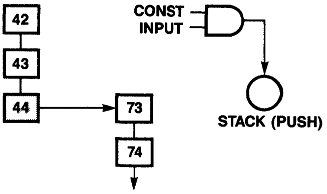
The PUSH Input GOTO command will push the eight inputs (I7–I0) onto the Stack. In the example address 44 has the instruction 'PUSHI GOTO label1' where label1 is located at address 73. At the leading edge of the clock the eight inputs are pushed onto the Stack. In a typical example, address 73 would have a RETURN instruction which would cause execution to jump to the address represented by the recently PUSHed input pins. This implements a dispatch function. The default for label1 will be the next instruction in the .ASM file. This instruction can also be used to load the Counter with an externally specified variable. In this case address 73 would have a POPC instruction.

PUSHI GOTO label1



The AND PUSH Input GOTO command is identical to the PUSHI command except the inputs are first bit-wise ANDed with a constant. This allows the masking of irrelevant inputs before PUSHing an address for a dispatch routine.

ANDPUSHI constant GOTO label1



The POP and XOR Stack to Counter GOTO command will pop the top of Stack, bitwise XOR it with a constant, load the result into the Counter, and jump to label1. In the example, address 44 has the command 'POPXORC 25D GOTO label1' where label1 is located at address 73. The top of Stack is POPed off the Stack, XORed with the decimal number 25, and loaded into the Counter. The next state comes from address 73. Since a XOR function does a comparison, this command can be used to compare the input to a constant and then branch based on the result with a LOOPNZ command. If the GOTO designation is left off the default for label1 will be the next instruction in the .ASM file.

POPXORC constant GOTO label1

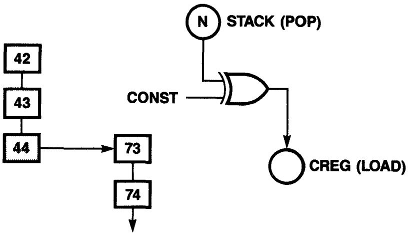


Figure 10. Instruction Set Summary

Instruction	Definition	Next-State Address	Stack	Counter
CONTINUE	Continue with Next Instruction	label1	None	HOLD
JUMP	Jump to a Label	label1	None	HOLD
CALL	Call Subroutine	label1	label2	HOLD
RETURN	Return From Subroutine	STACK	POP	HOLD
LOADC	Load CREG	label1	None	Constant
LOOPNZ	Loop/Dec. on Non-Zero	label 1 or 2	None	DECREMENT
DECNZ	Decrement CREG on Non-Zero	label1	None	DECREMENT
PUSHLOADC	Push CREG to Stack and Load CREG	label1	CREG	Constant
POPC	Pop Stack to CREG	label1	POP	STACK
PUSH	Push Constant to Stack	label1	Constant	HOLD
PUSHI	Push Inputs to Stack	label1	INPUTS	HOLD
ANDPUSHI	Push Masked Inputs to Stack	label1	INP * const	HOLD
POPXORC	XOR Stack with Constant and Send Result to CREG	label1	POP	STACK ⊕ Constant

NOTE: The value label1 is placed ın the Q-field. The values label2 and constant are placed in the D-fıeld.

Multi-Way Branching

The multi-way branching capability can be super imposed upon the instruction set providing another dimension of capability. Figure 11 shows how this translates into the flow diagrams. If location 44 had the instruction 'JUMP label1' where label1 is located at address 201, then the next-state would come from address 201. But address 201 is within the Multi-Way Branch Block so the Branch Select EPLD must decide which of the four words to send to the pipeline register. This selection is based on user-defined functions of the inputs.

Similarly, location 44 could contain any of the 13 available commands so that the multi-way branch capability can enhance each instruction. If location 44 was a CALL to a subroutine, then address 201 could contain the starting instruction for 4 unique subroutines. The actual routine executed would depend on the condition of the inputs as defined by the user.

The actual Assembly Language code required to implement this example is as follows:

```
44D: [Output Spec] CALL label1;
201D: IF cond1 THEN [out 1] JUMP 102D;
ELSEIF cond2 THEN [out 2] JUMP 73D;
ELSEIF cond3 THEN [out 3] JUMP 53D;
ELSE [out 4] JUMP 34D;
```

Figure 11. Jump to a Multi-Way Branch Address

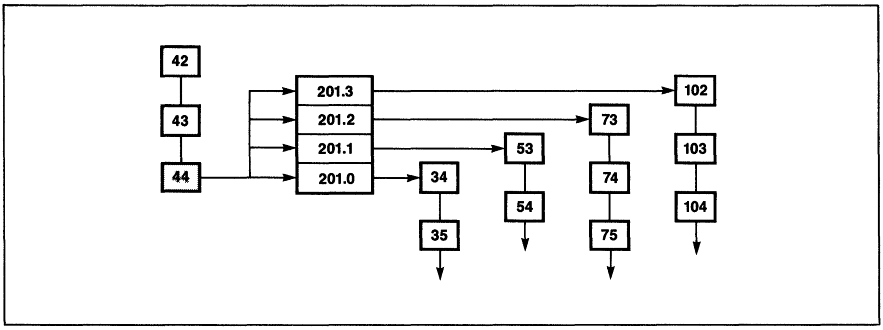
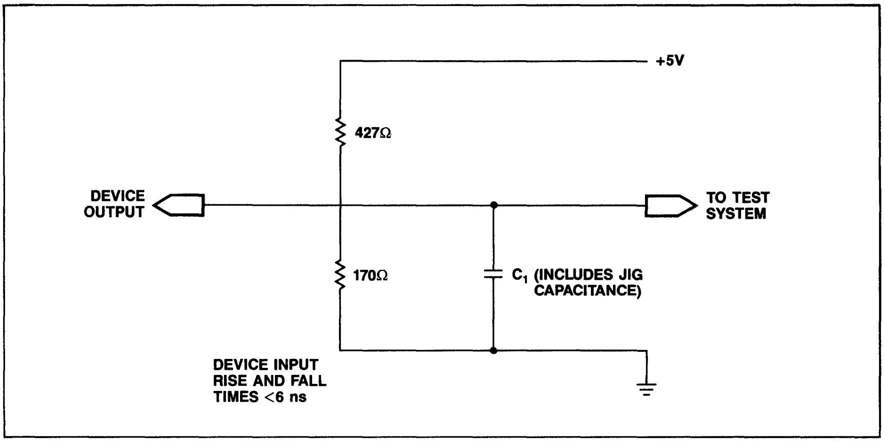
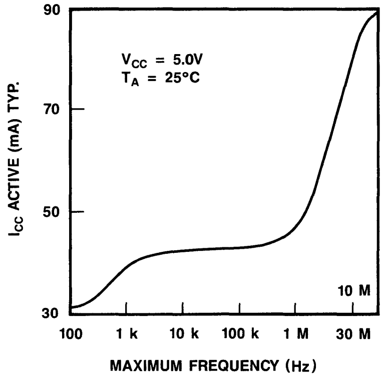


Figure 12. AC Test Conditions


+5V
427Ω
DEVICE OUTPUT
TO TEST SYSTEM
170Ω
C_1 (INCLUDES JIG CAPACITANCE)
DEVICE INPUT RISE AND FALL TIMES <6 ns

Power supply transients can affect AC measurements; simultaneous transitions of multiple outputs should be avoided for accurate measurement. Do not attempt to perform threshold tests under AC conditions. Large amplitude, fast ground current transients normally occur as the device outputs discharge the load capacitances. These transients flowing through the parasitic inductance between the device ground pin and the test system ground can create significant reductions in observable input noise immunity.

Figure 13. I_{CC} vs. F_{MAX}



Product Grades

Application	Temperature Range	Marking Designator
Commercial	0°C to +70°C	
Industrial	−40°C to +85°C	I
Military	−55°C to +125°C	M
MIL-STD-883C, Class B	−55°C to +125°C	MB

WAFERSCALE INTEGRATION, INC.

SAM448

System Development Tools

SAM system development tools are a complete set of PC-based development tools for the SAM448. Installed on an IBM-XT, AT or compatible computer, these tools provide an integrated easy-to-use software and hardware environment to support SAM448 development. These tools may be purchased as a complete development system or as individual software and hardware products. SAM system development tools contain all necessary programming hardware and software required to build high-performance state machines.

Host Requirements

The host system requirements for installing and using the SAM448 system development tools are an IBM-XT, AT, or compatible computer running MS-DOS version 3.1 or later. The system must be equipped with 640 Kbytes of RAM and a hard disk.

Hardware

The SAM448 system programming hardware consists of the following:

- MagicPro — Memory and System Programmer
- WS6008 — 28 Pin Dip Socket Adaptor for MagicPro Remote Socket Adaptor Unit
- WS6009 — 28 Pin LCC Socket Adaptor for MagicPro Remote Socket Adaptor Unit

The MagicPro Programmer is the common hardware platform for programming all WSI programmable products. It consists of the IBM-PC® plug-in Programmer Board and the Remote Socket Adaptor Unit.

Software

The SAM448 System Development Software consists of the following:

- WISPER Software — WSI Integrated Software and Programming Environment
- SAMPLUS Software — Interface Manager for SAM Tools
- ASMILE Software — System Entry Language
- SAMSIM Software — Functional Simulator
- SAMPRO Software — Device Programming Software

The complete SAM448 development cycle is illustrated in Figure 1.

WISPER and SAMPLUS software provide a menu-driven user interface enabling other tools to be easily invoked by the user.

The system design is entered into ASMILE (WSI State Machine Input Language) source program files using an editor chosen by the user. ASMILE supports Microcode entry and State Machine entry.

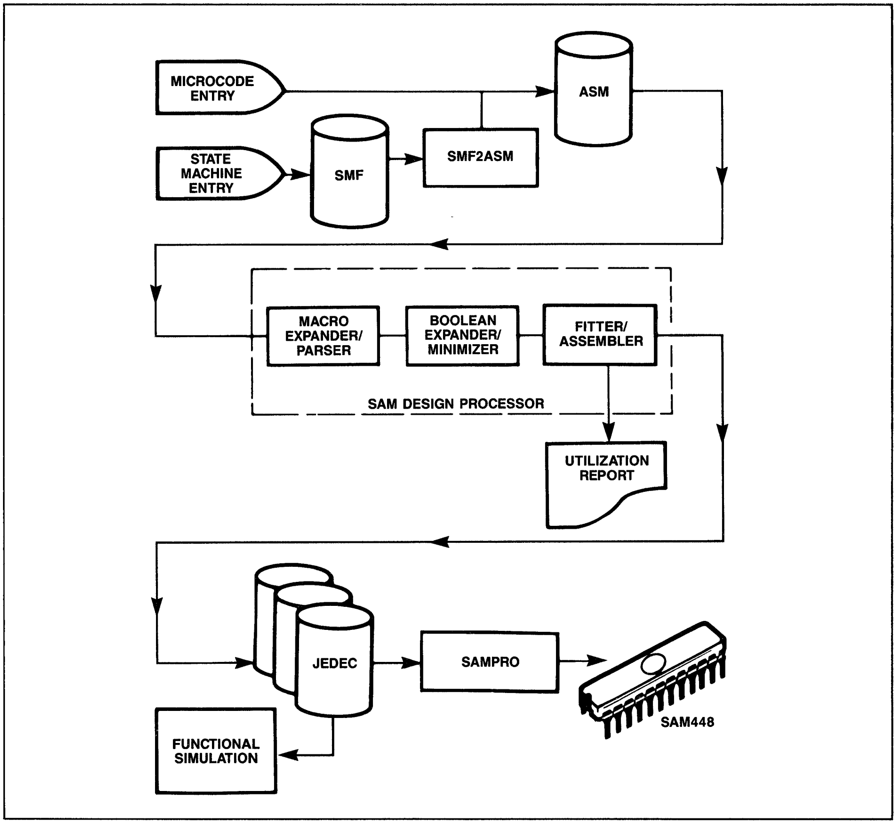
The ASMILE produces object code format which can be loaded to SAMSIM and SAMPRO

The SAMSIM functional simulator enables the user to test and debug programs by examining the state of SAM448 internal states before and during a complete functional simulation of the device.

SAMPRO software programs SAM448 devices by using the MagicPro hardware and the socket adaptor.

The programmed SAM448 is then ready to be used.

Figure 1. SAM Development Cycle



Documentation

SAM448 Software User's Manual.

WSI-Support

WSI provides a complete set of quality support services (WSI-Support) to registered system development tools owners. These services include the following:

- 12-Month Software update service — Up-to-date software maintenance, access to latest software and product information.
- Hotline to WSI Application Experts — Direct system development assistance
- 24-Hour Electronic Bulletin Board Service — Design assistance via our auto-answer dial-up modem service.

Training Workshops

WSI provides "Do-It-Yourself Systems" Technical Training Workshops that provide an in-depth tutorial on SAM448 and SAM system development tools.

Workshop participants will learn how to build their own high-performance state machine using the SAM448. SAM Development Training Workshops are held at the WSI Fremont facility.

Ordering Information — System Development Tools

SAM448-Gold package consists of the following:

- Software
  - — WISPER Software
  - — SAMPLUS Software
  - — ASMILE Software
  - — SAMSIM Software
  - — SAMPRO Software
  - — User's Manual
  - — WSI-Support
- Hardware
  - — WS6000 MagicPro Programmer

SAM448-Silver package consists of the following:

- Software
  - — WISPER Software
  - — SAMPLUS Software
  - — ASMILE Software
  - — SAMSIM Software
  - — SAMPRO Software
  - — User's Manual
  - — WSI-Support

WS6000 MagicPro™ Memory and PSD Programmer

- Includes IBM PC plug-in adaptor card and Remote Socket Adaptor

Adaptors

- WS6008 28 Pin Dip Socket Adaptor
- WS6009 28 Pin CLLCC/CLDCC/PLDCC Socket Adaptor

WSI-Support

- Includes 12-month Software Update Service to registered system owners
- Includes Hotline to WSI Application experts
- Includes 24-hour access to WSI's Electronic Bulletin Board Service

SAM Training Workshops

- Includes SAM448 Training Workshops at the WSI Fremont facility. For details and scheduling, contact PSD Marketing at (415) 656-5400.

Ordering Information

Part Number	Speed (MHz)	Package Type	Package Drawing	Operating Temperature Range	WSI Manufacturing Procedure
SAM448-20J	20	28 Pin PLDCC	J3	Comm'l	Standard
SAM448-20L	20	28 Pin CLDCC	L2	Comm'l	Standard
SAM448-20LI	20	28 Pin CLDCC	L2	Industrial	Standard
SAM448-20LM	20	28 Pin CLDCC	L2	Military	Standard
SAM448-20LMB	20	28 Pin CLDCC	L2	Military	MIL-STD-883C
SAM448-20S	20	28 Pin Plastic Dip, 0.3″	S2	Comm'l	Standard
SAM448-20T	20	28 Pin CERDIP, 0.3″	T2	Comm'l	Standard
SAM448-20TI	20	28 Pin CERDIP, 0.3″	T2	Industrial	Standard
SAM448-20TM	20	28 Pin CERDIP, 0.3″	T2	Military	Standard
SAM448-20TMB	20	28 Pin CERDIP, 0.3″	T2	Military	MIL-STD-883C
SAM448-25J	25	28 Pin PLDCC	J3	Comm'l	Standard
SAM448 25L	25	28 Pin CLDCC	L2	Comm'l	Standard
SAM448-25S	25	28 Pin Plastic Dip, 0.3″	S2	Comm'l	Standard
SAM448-25T	25	28 Pin CERDIP, 0.3″	T2	Comm'l	Standard
SAM448-30J	30	28 Pin PLDCC	J3	Comm'l	Standard
SAM448-30L	30	28 Pin CLDCC	L2	Comm'l	Standard
SAM448-30S	30	28 Pin Plastic Dip, 0.3″	S2	Comm'l	Standard
SAM448-30T	30	28 Pin CERDIP, 0.3″	T2	Comm'l	Standard

WAFERSCALE INTEGRATION, INC.

Section Index

PSD Development Systems

For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

MAP168

PSD Development System

Description

MAP168-GOLD/MAP168-SILVER is a complete set of IBM-PC-based development tools. They provide the integrated easy-to-use environment to support the MAP168 program development and device programming.

The tools run on an IBM-PC XT, AT or compatible computer running MS-DOS version 3.1 or later.

MAPLE

MAPLE is the MAP168 Locator Editor. It has the following features:

□ Simple Menu Driven Commands for selecting different configurations of the MAP168:
- — Byte wide or word wide operation.
- — Address or Chip Select Input (CSI) Mode.
- — PAD security option.

□ Generating the PAD programming data that maps the 8 segments of EPROM, two segments of SRAM and eight Chip Selects Outputs to the user's address space.

□ Combining all the different files to be programmed into the EPROM segments.

MAPPRO

MAPPRO is the interface software that enables the user to program a MAP168 device on the WS6000 MagicPro™ programmer. The MAPPRO enables the user to load the program into the programmer and to execute the following operations:

□ Help

□ Upload RAM from MAP

□ Load RAM from disk

□ Write RAM to FILE

□ Display MAP data

□ Blank test MAP

□ Verify MAP

□ Program MAP

□ Configuration

□ Quit

WS6000 MagicPro™ Programmer

The WS6000 MagicPro Programmer is an engineering development tool designed to program all WSI programmable products (EPROMs, RPROMs, PAC1000, MAP168, PSD301 and SAM448). It is used within the IBM-PC and compatible environment. The MagicPro consists of a short plug-in board and a Remote Socket Adaptor (RSA). It occupies a short expansion slot in the PC. The RSA has two ZIF-DIP sockets that will support WSI's 24, 28, 32 and 40 pin standard 600 mil or slim 300 mil DIP packages without adaptors. Other packages are supported using adaptors.

WS6003 Socket Adaptor

The WS6003 is a socket adaptor that mounts on the MagicPro RSA and adapts the MAP168 in 44-pin CLDCC, PLDCC or CLLCC packages to the programmer.

WS6011 Socket Adaptor

The WS6011 is a socket adaptor that mounts on the MagicPro RSA and adapts the MAP168 in a 44-pin PGA package to the programmer.

WSI-Support

WSI provides on-going support for users of MAP168-GOLD/MAP168-SILVER. For the first year, software and programmer updates are included at no charge. After that, the user may purchase the WSI-Support agreement to continue to receive the latest software releases.

Ordering Information

Product	Description
MAP168-SILVER	Contains MAP168 Software (MAPLE-MAP and MAPPRO), Software User's Manual, WSI-Support.
MAP168-GOLD	Contains MAP168-SILVER, WS6000 MagicPro Programmer, WSI-Support.
WSI-Support	12-Month Software Update Service, Access to WSI's 24-Hour Electronic Bulletin Board, and Hotline to WSI System Application Experts.

MAP168-GOLD

Contents

❑ MAPLE-MAP Locator editor.

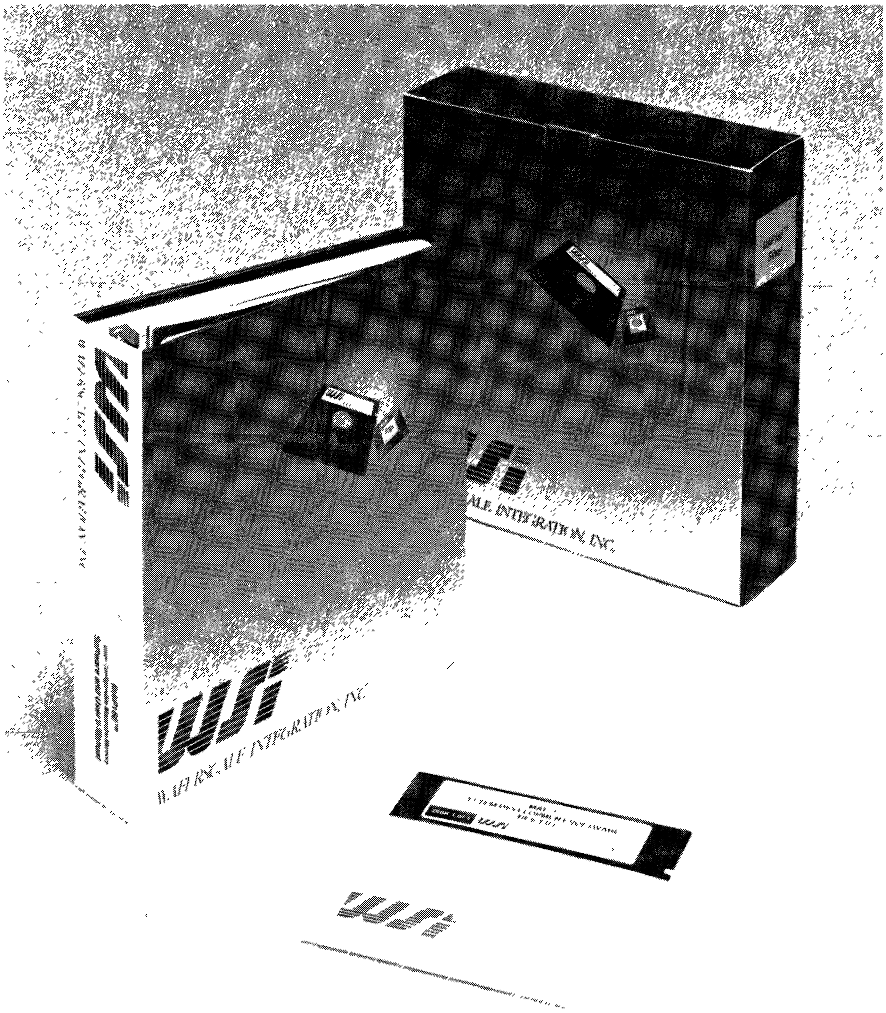
❑ MAPPRO
Interface software to MAP168 device programmer (MagicPro™).

❑ Software user's manual.

❑ WSI-SUPPORT agreement.

❑ WS6000 MagicPro Programmer.

MAP168-SILVER



Contents

□ MAPLE-MAP Locator editor.

□ MAPPRO
Interface software to MAP168 device programmer (MagicPro^{TM}).

□ Software user's manual.

□ WSI-SUPPORT agreement.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

SAM448

PSD Development System

Description

SAM448-GOLD/SAM448-SILVER is a complete set of IBM-PC-based development tools. They provide the integrated easy-to-use environment to support the SAM448 program development and device programming.

The tools run on an IBM-PC XT, AT or compatible computer running MS-DOS version 3.1 or later.

ASMILE

ASMILE is the SAM448 system entry language. It has the following features:

- ❑ State Machine Design Entry.
- ❑ Assembly Design Entry Language.
- ❑ User Definable Macros

SAMSIM

SAMSIM is an interactive functional simulator with Virtual Logic Analyzer Interface:

- ❑ Clock driven functional simulator.
- ❑ Provides trace capabilities on internal states (Registers, Flags, Pins and more).
- ❑ Displays input and output waveforms interactively providing such features as multiple zoom levels, split screens and differential time display.
- ❑ Line disassembler converts the actual code back into the original Assembly source code.
- ❑ On-line HELP available at any level.

SDP

The SAM Design Processor (SDP) takes an assembly file and creates an optimized JEDEC file for the SAM448. The SDP first expands macros that have been defined by the user. It then parses the design, listing any syntax or correction errors in an Error Log file. Next it minimizes the Boolean expressions that define the transition conditions. Finally, it fits the design into the SAM448, generating a JEDEC file.

SAMPRO

SAMPRO is the interface software that enables the user to program a SAM448 device on the WS6000 MagicPro™ programmer. The SAMPRO enables the user to load the program into the programmer and to execute the following operations:

- ❑ Help
- ❑ Upload RAM from SAM
- ❑ Load RAM from disk
- ❑ Write RAM to FILE
- ❑ Display SAM data
- ❑ Blank test SAM
- ❑ Verify SAM
- ❑ Program SAM
- ❑ Configuration
- ❑ Quit

SAMPLUS

SAMPLUS is the interface manager to the SAM448 software tools. SAMPLUS enables the user to access ASMILE, SAMSIM, SDP, SAMPRO, DOS and an editor with a menu driven interface. File specification can be done without extension enabling the user to use the same name throughout the design. A HELP window is available on-line giving information on all the needed steps at each level.

WS6000 MagicPro™ Programmer

MagicPro is an engineering development tool designed to program all WSI programmable products (EPROMs, RPROMs, PAC1000, MAP168, PSD301 and SAM448). It is used within the IBM-PC and compatible environment. The MagicPro consists of a short plug-in board and a Remote Socket Adaptor (RSA). It occupies a short expansion slot in the PC. The RSA has two ZIF-DIP sockets that will support WSI's 24, 28, 32 and 40 pin standard 600 mil or slim 300 mil DIP packages without adaptors. Other packages are supported using adaptors.

WS6008 Socket Adaptor

The WS6008 is a socket adaptor that mounts on the MagicPro RSA and adapts the SAM448 in a 28 pin DIP package to the programmer.

WS6009 Socket Adaptor

The WS6009 is a socket adaptor that mounts on the MagicPro RSA and adapts the SAM448 in a 28-pin PLDCC/CLDCC/CLLCC package to the programmer.

WSI-Support

WSI provides on-going support for users of SAM448-GOLD/SAM448-SILVER. For the first year, software and programmer updates are included at no charge. After that, the user may purchase the WSI-Support agreement to continue to receive the latest software releases.

Ordering Information

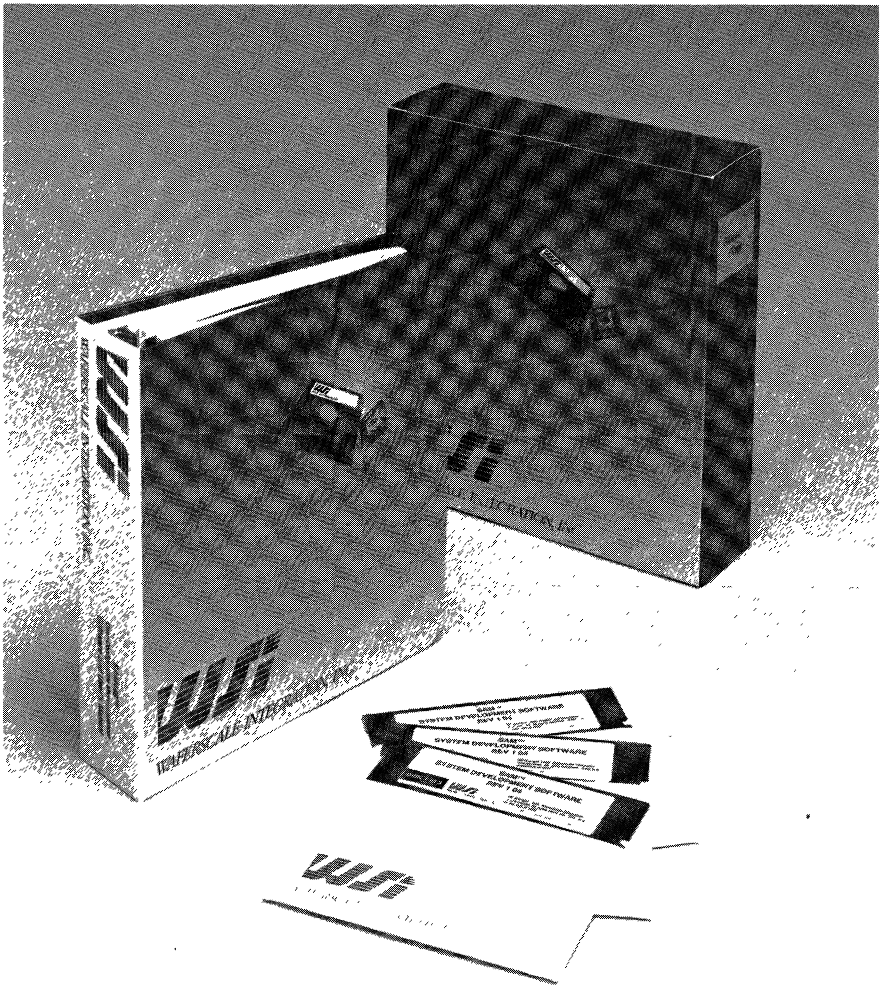
Product	Description
SAM448-SILVER	Contains SAM448 Software (ASMILE, SAMSIM, SDP, SAMPRO and SAMPLUS), Software User's Manual, WSI-Support.
SAM448-GOLD	Contains SAM448-SILVER, WS6000 MagicPro Programmer, WSI-Support.
WSI-Support	12-Month Software Update Service, Access to WSI's 24-Hour Electronic Bulletin Board, and Hotline to WSI System Application Experts.

SAM448-GOLD

Contents

- ❑ ASMILE
  SAM design entry language.
- ❑ SAMSIM
  Interactive Functional simulator with Virtual Logic Analyzer user interface.
- ❑ SDP
  SAM Design Processor Compiles the User's program to fit into the SAM448 Device.
- ❑ SAMPRO
  Interface software to SAM448 device programmer (MagicPro™).
- ❑ SAMPLUS
  Interface manager to SAM448 development tools.
- ❑ Software user's manual.
- ❑ WSI-SUPPORT agreement.
- ❑ WS6000 MagicPro Programmer.

SAM448-SILVER


WAFERSCALE INTEGRATION, INC.
SAM™ SYSTEM DEVELOPMENT SOFTWARE REV 1.04

Contents

- ❑ ASMILE
  SAM design entry language.
- ❑ SAMSIM
  Interactive Functional simulator with Virtual Logic Analyzer user interface.
- ❑ SDP
  SAM Design Processor Compiles the User's program to fit into the SAM448 Device.
- ❑ SAMPRO
  Interface software to SAM448 device programmer (MagicPro).
- ❑ SAMPLUS
  Interface manager to SAM448 development tools
- ❑ Software user's manual.
- ❑ WSI-SUPPORT agreement.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

PAC1000

PSD Development System

Description

PAC1000-GOLD/PAC1000-SILVER is a complete set of IBM-PC-based development tools. They provide the integrated easy-to-use environment to support the PAC1000 program development and device programming.

The tools run on an IBM-PC XT, AT or compatible computer running MS-DOS version 3.1 or later.

PACSEL

PACSEL is the PAC1000 system entry language. It has the following features:

❑ Enables specification of up to three parallel operations:
 — Program control operation
 — CPU operation
 — Out Control operation

 General Syntax:
 Label: Program Control, CPU, Out Control;

❑ Enables mixing of three source language types in one instruction:
 — High Level Language
 — Assembler
 — Microcode

❑ Specific instructions support unique PAC1000 architecture features available in all three source language types.

❑ Links unlimited amounts of modules.

PACSIM

PACSIM is a functional simulator and software debugger. It has the following features:

❑ Clock driven functional simulator.

❑ Provides trace capabilities on internal states (Registers, Flags, Pins and more).

❑ Provides breakpoint capabilities on any internal state of the PAC1000.

❑ Supports batch mode simulation.

❑ Provides waveform analysis.

❑ On-line HELP available at any level.

PACPRO

PACPRO is the interface software that enables the user to program a PAC1000 microcontroller on the WS6000 MagicPro™ programmer. The PACPRO enables the user to load the program into the programmer and to execute the following operations:

❑ Help

❑ Upload RAM from PAC

❑ Load RAM from disk

❑ Write RAM to FILE

❑ Display PAC data

❑ Blank test PAC

❑ Verify PAC

❑ Program PAC

❑ Configuration

❑ Quit

IMPACT

IMPACT is the interface manager to the PAC1000 tools. IMPACT enables the user to access PACSEL, PACSIM, PACPRO, DOS and an editor with a menu driven interface. File specification can be done without extension enabling the user to use the same name throughout the design. A HELP window is available on-line giving information on all the needed steps at each level.

WS6000 MagicPro™ Programmer

MagicPro is an engineering development tool designed to program all WSI programmable products (EPROMs, RPROMs, PAC1000, MAP168, PSD301 and SAM448). It is used within the IBM-PC and compatible environment. The MagicPro consists of a short plug-in board and a Remote Socket Adaptor (RSA). It occupies a short expansion slot in the PC. The RSA has two ZIF-DIP sockets that will support WSI's 24, 28, 32 and 40 pin standard 600 mil or slim 300 mil DIP packages without adaptors. Other packages are supported using adaptors.

WS6010 Socket Adaptor

The WS6010 is a socket adaptor that mounts on the MagicPro RSA and adapts the PAC1000 in an 88-pin CPGA package to the programmer.

WS6013 Socket Adaptor

The WS6013 is a socket adaptor that mounts on the MagicPro RSA and adapts the PAC1000 in a 100-pin QFP package to the programmer.

WSI-Support

WSI provides on-going support for users of PAC1000-GOLD/PAC1000-SILVER. For the first year, software and programmer updates are included at no charge. After that, the user may purchase the WSI-Support agreement to continue to receive the latest software releases.

Ordering Information

Product	Description
PAC1000-SILVER	Contains PAC1000 Software (PACSEL, PACSIM, PACPRO, and IMPACT), Software User's Manual, WSI-Support.
PAC1000-GOLD	Contains PAC1000-SILVER, WS6000 MagicPro Programmer, WSI-Support.
WSI-Support	12-Month Software Update Service, Access to WSI's 24-Hour Electronic Bulletin Board, and Hotline to WSI System Application Experts.

PAC1000-GOLD

Contents

❑ PACSEL
System design entry language and program linker.

❑ PACSIM
Functional simulator and software debugger.

❑ PA,CPRO
Interface software to PAC1000 device programmer (MagicPro™).

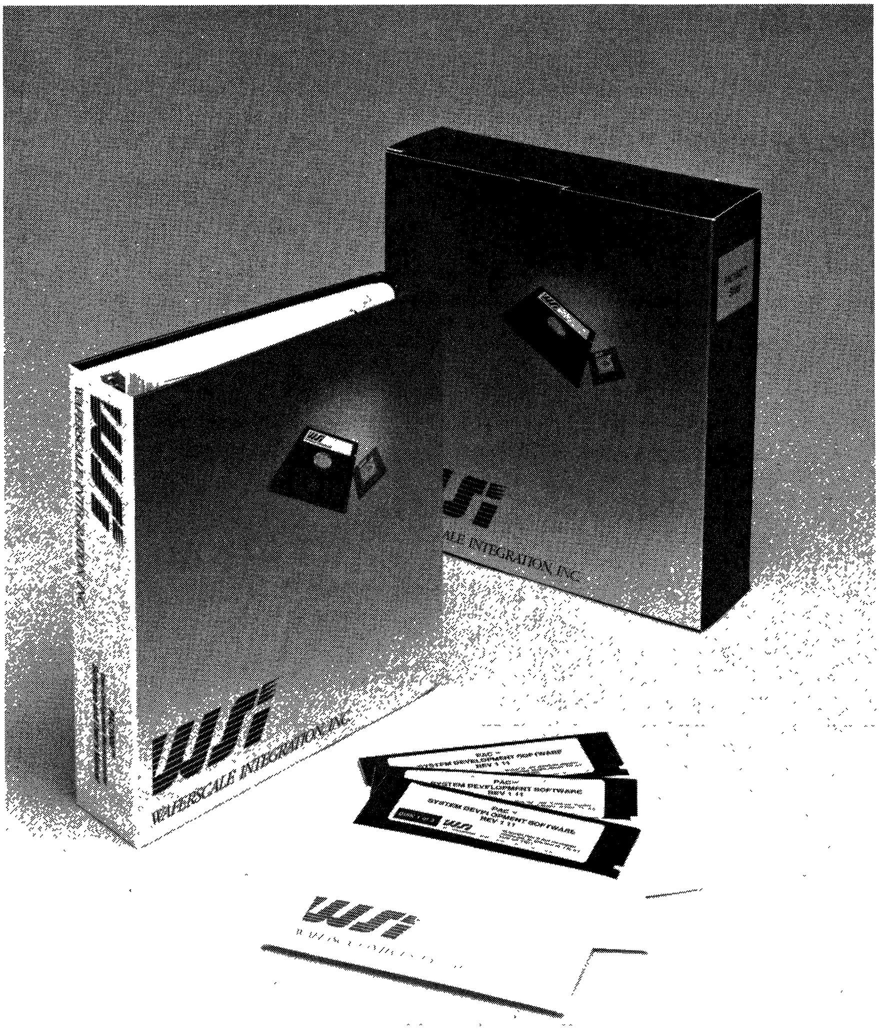
❑ IMPACT
Interface manager for PAC1000 microcontroller development tools.

❑ Software user's manual.

❑ WSI-SUPPORT agreement.

❑ WS6000 MagicPro Programmer.

PAC1000-SILVER


WAFERSCALE INTEGRATION, INC.
PAC™ SYSTEM DEVELOPMENT SOFTWARE REV 1.11

Contents

- ☐ PACSEL
  System design entry language and program linker.
- ☐ PACSIM
  Functional simulator and software debugger.
- ☐ PACPRO
  Interface software to PAC1000 device programmer (MagicPro™).
- ☐ IMPACT
  Interface manager for PAC1000 microcontroller development tools.
- ☐ Software user's manual.
- ☐ WSI-SUPPORT agreement.

WAFERSCALE INTEGRATION, INC.

MAGICPRO™ MEMORY AND PSD PROGRAMMER

KEY FEATURES

- Programs All WSI CMOS Memory and PSD Products and All Future Programmable Products
- Programs 24, 28, 32 and 40 Pin Standard 600 Mil or Slim 300 Mil Dip Packages Without Adaptors
- Programs LCC, PGA and QFP Packaged Product by Using Adaptors
- Easy-to-Use Menu-Driven Software
- Compatible with IBM PC/XT/AT Family of Computers (and True Plug-Compatible)

GENERAL DESCRIPTION

MAGICPRO™ is an engineering development tool designed to program existing WSI EPROMs, RPROMs, Programmable System Devices, and future WSI programmable products. It is used within the IBM-PC® and compatible computers. The MAGICPRO™ is meant to bridge the gap between the introduction of a new WSI programmable product and the availability of programming support from programmer manufacturers (e.g., Data I/O, etc.). The MAGICPRO™ programmer and accompanying software enable quick programming of newly released WSI programmable products, thus accelerating the system design process.

The MAGICPRO™ plug-in board is integrated easily into the IBM-PC®. It occupies a short expansion slot and its software requires only 256K bytes of computer memory. The two external ZIF-Dip sockets in the Remote Socket Adaptor (RSA) support 24, 28, 32 and 40 pin standard 600 mil or slim 300 mil Dip packages without adaptors. LCC, PGA and QFP packages are supported using adaptors.

Many features of the MagicPro™ Programmer show its capabilities in supporting WSI's future products. Some of these are:

- — 24 to 40 pin JEDEC Dip pinouts
- — 1 Meg. address space (20 address lines)
- — 16 data I/O lines

The MagicPro™ menu driven software makes using different features of the MagicPro™ an easy task. Software updates are done via floppy disk which eliminates the need for adding a new memory device for system upgrading. Please call 800-TEAM-WSI for information regarding programming WSI products not listed herein. The MagicPro™ reads Intel Hex format for use with assemblers and compilers.

MagicPro™ *COMMANDS*

- — Help
- — Upload RAM from device
- — Load RAM from disk
- — Write RAM to disk
- — Display RAM data
- — Edit RAM
- — Move/copy RAM
- — Fill RAM
- — Blank test device
- — Verify device
- — Program device
- — Select device
- — Configuration
- — Quit MagicPro™

WSI PRODUCTS

WS57C191/191B/291/291B	2K × 8	RPROM
WS57C43/43B	4K × 8	RPROM
WS57C49/49B	8K × 8	RPROM
WS57C51/51B	16K × 8	RPROM
WS27C64F/L	8K × 8	EPROM
WS57C64F	8K × 8	EPROM
WS57C65	4K × 16	EPROM
WS57C66	4K × 16	EPROM
	(Mux I/O, 28 Pin DIP)	
WS27C128F/L	16K × 8	EPROM
WS57C128F	16K × 8	EPROM
WS27C256F/L	32K × 8	EPROM
WS57C256F	32K × 8	EPROM
WS57C257F	16K × 16	EPROM
WS27C512F/L	64K × 8	EPROM
WS27C010L	128K × 8	EPROM
MAP168		
PAC1000		
SAM448		
PSD301		

TECHNICAL INFORMATION

- Size: IBM-PC® short length card
- Port Address Location: 100H to 1FFH—default 140H (If a conflict exists with this address space, the address location can be changed in software and with the switches on the plug-in board.)
- System Memory Requirements: 256K bytes of RAM
- Power: +5 Volts, 0.03 Amp.; +12 Volts, 0.04 Amp.
- Remote Socket Adaptor (RSA): The RSA contains two ZIF-Dip sockets that are used to program and read WSI programmable products. The 32 pin ZIF-Dip socket supports 24, 28 and 32 pin standard 600 mil or slim 300 mil Dip packaged product. The 40 pin ZIF-Dip socket supports all 40 pin Dip packages. Adaptor sockets are available for LCC, PGA and QFP packages.

ORDERING INFORMATION

The WS6000 MAGICPRO™ System contains:

- MAGICPRO™ IBM-PC® plug-in programmer board
- MAGICPRO™ Remote Socket Adaptor and cable
- MAGICPRO™ Operating System Floppy Disk and Operating Manual

MAGICPRO™ Adaptors include:

- WS6001 28 Pin CLLCC Package adaptor for memory.
- WS6003 44 Pin PLDCC/CLDCC/CLLCC package adaptor for MAP168.
- WS6008 28 Pin 0.3″ wide DIP adaptor for SAM448.
- WS6009 28 Pin PLDCC/CLDCC/CLLCC package adaptor for SAM448.
- WS6010 88 Pin PGA package adaptor for PAC1000.
- WS6011 44 Pin PGA package adaptor for MAP168.
- WS6012 32 Pin CLDCC package adaptor for memory.
- WS6013 100 Pin QFP package adaptor for PAC1000.
- WS6014 44 Pin CLDCC/PLDCC package adaptor for MAP168 and PSD301.
- WS6015 44 Pin PGA package adaptor for MAP168 and PSD301.

WAFERSCALE INTEGRATION, INC.

Section Index

PSD Applications

For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

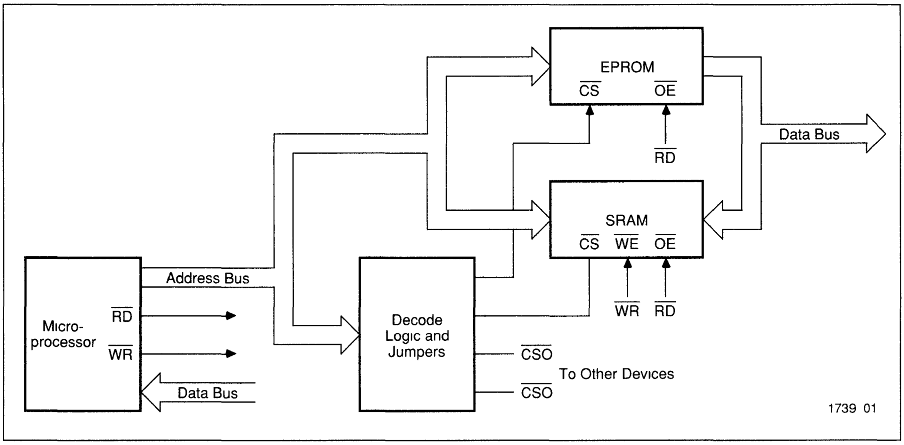
Application Note 002

Introduction to the MAP168 User-Configurable Mappable Memory Subsystem

Memory Structure

Memory configurations in microprocessor and microcontroller systems have similar structure, irrespective of the application. (see Figure 1.) They share basic components, such as an EPROM (for program storage), and an SRAM (for data storage). In addition, a decoder circuit is required to select blocks of memory from the address inputs applied by the processor. A common implementation of address decoding originally used MSI building blocks, such as 74xx138 devices. Memory-configuration changes and expansions in a fixed-logic solution required jumpers on the printed circuit board. More recently, decoders based on PAL® devices have provided a more compact and flexible solution. PAL devices allow configuration changes to be implemented by insertion of a programmed device and avoid jumper changes.

Figure 1. Memory Subsystem Using Standard Devices



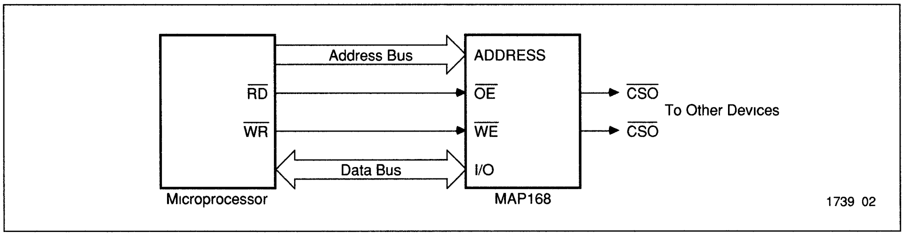
Both solutions involve compromises that affect system performance, board space, power and cost. Since the decoder is in the memory access path, the total memory access time is the sum of the decoder delay and the access time of the memory itself. For example, a 40ns total access time can be achieved with a 12ns decoder and a 25ns memory. This allows 3ns for on-board interconnect delay. Memory products in the 25ns range are expensive and therefore such a performance entails additional cost. To be able to integrate the programmable address decoder with system memory, EPROM and static RAM would offer a more flexible approach. The resulting device would provide board-space economy, higher performance and less overall power consumption without the cost of a multichip solution.

Memory Structure (Con't)

The WSI-MAP family of user-configurable mappable memory subsystem products has been developed to significantly enhance system performance by integrating high density EPROM for program store, high density SRAM for data store and high performance logic in the form of a Programmable Address Decoder (PAD) on one chip. (See Figure 2.) The first of these devices, with 128K bits of EPROM and 32K bits of SRAM, is the MAP168 device. These devices are ideally suited for a number of common design applications:

- ❑ High-speed Digital Signal Processor applications (modems, analog data filtering or analysis)
- ❑ Expanding memory systems for microprocessors and microcontrollers
- ❑ Space- and power-sensitive applications (plug-in cards, avionics, portable systems)

Figure 2. Memory Subsystem Using the MAP168 Device



Features of the MAP168 Device

The MAP168 device offers significant design advantages through integration, performance and user-configurability. It integrates both volatile and non-volatile memory on the same chip, along with a flexible decoding system. The memory is structured as a series of blocks to achieve a highly configurable circuit for general purpose applications. The device operates in one of several modes, one of which is for normal operation and the rest are for device configuration. At the heart of all MAP168 device is a Programmable Address Decoder (PAD), which is programmed during the PAD programming mode through the circuit's address and I/O pins. The PAD offers the following features:

- ❑ Flexible EPROM/SRAM location within the address space
- ❑ Memory array power-down when not being accessed
- ❑ Security protection of memory configuration data to inhibit copying
- ❑ Integrated external device mapping through Chip Select Outputs

Memory Architecture And Technology

The memory in the MAP168 device consists of non-volatile EPROM and volatile SRAM. (See Figure 3.) The EPROM is subdivided into 8 blocks and the SRAM into 2 blocks. The blocks may be configured in either a 2Kx8 or a 1Kx16 format, allowing optimal interaction with both 8- and 16-bit systems. These memory blocks can be considered as separate memories with dedicated internal chip selects. The PAD selects the appropriate block, decoded from the incoming address provided at the device inputs. This architecture enables the product to be configured and compatible with virtually any system address map. Complicated address maps of microcontroller systems can be fully realized by programming blocks of EPROM and SRAM in the memory-mapping scheme of the system.

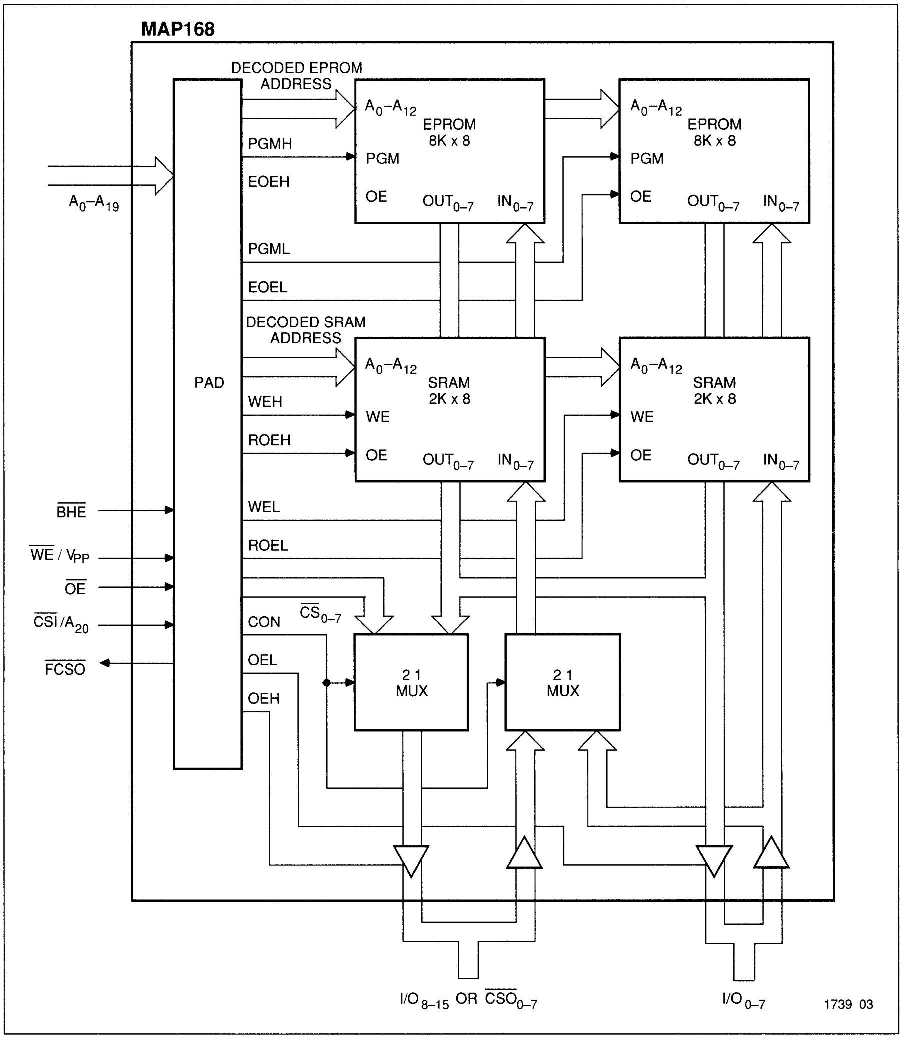
Features of the MAP168 Device (Con't)

In addition to having fine control of memory allocation, software updates which require changes in the address map boundaries can be easily accomplished by simply reprogramming the PAD at the same time as the EPROM code. This means only one part need be sent to the end-product customer to accommodate field software changes. This becomes a user-transparent method that requires no change of PC board jumpers.

The EPROM is based on WSI's patented split-gate EPROM technology for high density and very high speed. It is also used in the reconfigurable PAD section, permitting both fast decode and reconfiguration of the same device. The MAP168 device contains a 128K-bit UV erasable EPROM which can be organized as 16Kx8 (byte-wide) or as 8Kx16 (word-wide).

The SRAM is based on the industry standard full CMOS 6-transistor cell. The advantages of this cell are high speed, very low stand-by power, high noise immunity and good data retention when disturbed by alpha particles. In the MAP168 device, the SRAM contains 32K bits which can be configured as 4Kx8 in the byte mode or 2Kx16 in the word-wide mode.

Figure 3. Internal Architecture



Features of the MAP168 Device (Con't)

PAD Logic Implementation

The PAD uses the same non-volatile EPROM cells as the EPROM array. (See Figure 4.) It can be erased and configured at the same time as the EPROM. After UV erase or with new parts, the EPROM cells in the MAP168 device normally connect between the address inputs and the select outputs. The EPROM cells are disconnected by selective programming.

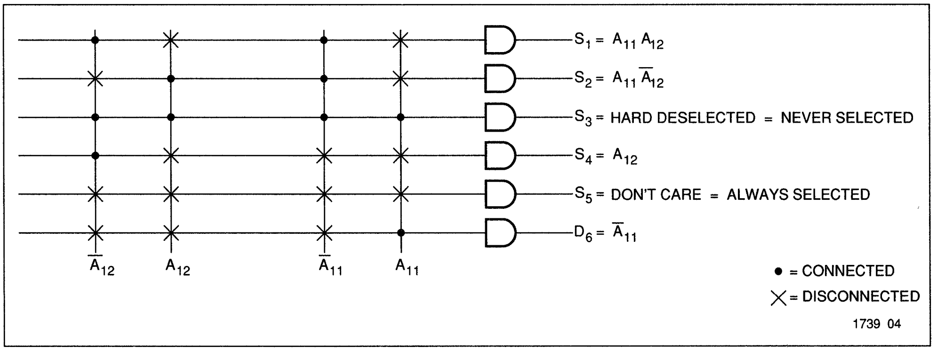
The PAD performs as an address comparator. When the address configuration previously programmed into the PAD is detected, the internal chip-select signal to the memory block selected by that address is enabled. If no block is selected by the address, neither the EPROM nor the SRAM arrays are enabled and other devices may drive the data bus. Independent of internal block selection, external chip-select decoding (known as $\overline{\text{CSO}}$s) are programmable in the same block resolution as the internal memory.

Actual implementation of the PAD is similar to that of a PAL device. (See Figure 5.) In the erased state, all the block decode addresses are connected to the AND plane. There is only one output per AND gate and there is no OR plane. Each AND gate output either selects a block of internal memory or a number of blocks of external memory for the external $\overline{\text{CSO}}$s. Only addresses A_{11}–A_{20} are used as block decode address. Lower-order address lines are used only for addressing within the internal memory arrays.

EPROM select outputs ES_0–ES_7 (ES outputs) select 1 of the 8 available EPROM blocks. SRAM select outputs RS_0–RS_1 (RS outputs) select one of the 2 available SRAM blocks. Because only one EPROM or SRAM block can be active at a particular time, only one line from either ES_0–ES_7 or RS_0–RS_1 is allowed to be active at one time. The $\overline{\text{CSO}}$s are independent of the ES and RS outputs and therefore any one address can be programmed to select one or more of the $\overline{\text{CSO}}$s, even simultaneous to the selection of one of the ES or RS outputs. This is particularly useful for I/O control or address decode for wait state generation.

Programming the decoder is similar to programming a PAL device that has only one product term (AND gate) per output. To enable an output S_1 as shown in Figure 4, fuse locations A_{11} and A_{12} are left intact while $\overline{A}_{11}$ and $\overline{A}_{12}$ are programmed. Conversely, if A_{11} and A_{12} are programmed while their complements are left intact, then the select S function is active when $\overline{A}_{11} = \overline{A}_{12} = 0$. If all fuse locations are programmed on a product term, the inputs are pulled HIGH and no select output can take place. If all fuse locations are left intact, the S output is permanently LOW, always selected.

Figure 4. PAD Programming Examples

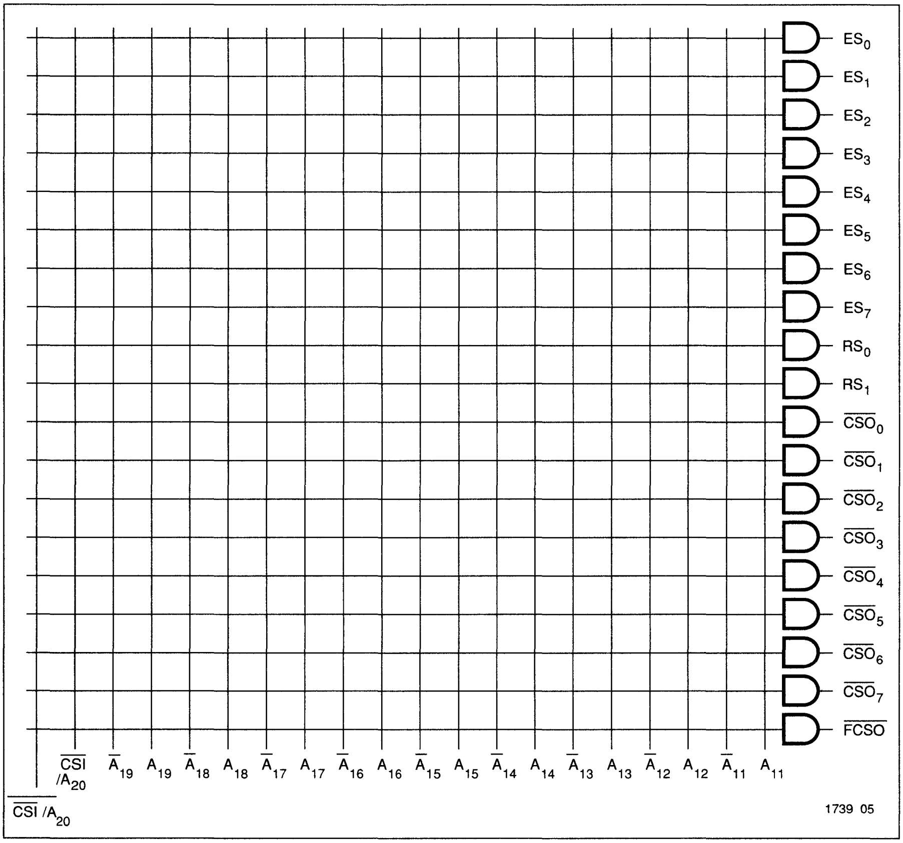

$S_1 = A_{11} A_{12}$
$S_2 = A_{11} \overline{A}_{12}$
S_3 = HARD DESELECTED = NEVER SELECTED
$S_4 = A_{12}$
S_5 = DON'T CARE = ALWAYS SELECTED
$D_6 = \overline{A}_{11}$
$\overline{A}_{12}$
A_{12}
$\overline{A}_{11}$
A_{11}
● = CONNECTED
X = DISCONNECTED
1739 04

Features of the MAP168 Device (Con't)

Device Array Power-Down

Power dissipation on the chip is minimized through logic in the PAD. It selectively powers up the EPROM or SRAM arrays only when they are being accessed. If the EPROM is selected through the decoder, it will draw power while the SRAM stays powered down and vice versa. When neither the EPROM or the SRAM is selected, both are powered down. Note that data integrity in a "powered down" SRAM is maintained. A Chip Select Input ($\overline{CSI}$) to the device is provided for a very low-power quiescent mode. With $\overline{CSI}$=1, the EPROM and SRAM are powered down but the PAD is powered up, independent of the incoming address signals. The $\overline{CSI}$ input pin can be connected to a system power-down signal. If such a signal is unavailable, addressing a location in memory that does not select either the EPROM or the SRAM also reduces power drain. In this case, only the PAD is powered up and draws a small fraction of the active power.

Figure 5. PAD Array Architecture



Features Of The MAP168 Device *(Con't)*

The $\overline{\text{CSI}}/A_{20}$ input is actually a dual function pin. It can be an address (MSB) input, or it can be programmed to be a chip select input as well. As a chip select input, it will enable the EPROM and SRAM memory when active (LOW). If the address option A_{20} is chosen the chip is always enabled.

Address Map Security

Upon entering the PAD programming mode, the contents of the PAD are fully accessible through the I/O pins. After programming is completed, it is possible to render the PADs programmed configuration invisible by programming the security (SEC) bit. This disables external access to the PAD and ensures that the PAD configuration can not be copied. To further aid in securing data in the MAP product, it is suggested that memory blocks that are addressed in a linear block placement be programmed in the PAD as chip selects from product terms that are randomly placed.

Chip Select Outputs

The MAP168 device can be user-configured for 8-bit or 16-bit systems. In the former case, eight unused data lines ($\overline{\text{CSO}}_{0-7}$) are available as chip select outputs, driven by the address decoder section of the PAD. This provides the ability to integrate external devices into the address map with no hardware overhead. Unlike the internal memory blocks, a $\overline{\text{CSO}}$ can be active for more than one address combination or block. Also, groups of blocks may overlap both each other and the internal memory. By deselecting both the true and the complement it is possible to make an address line "don't care".

An external memory can therefore be selected with only one $\overline{\text{CSO}}$. It is possible to enable another external 128K byte memory by programming a single $\overline{\text{CSO}}$ to be active for that entire address range.

A $\overline{\text{CSO}}$ can be programmed to function as a configuration bit which is always deselected (e.g., $\overline{\text{CSO}}_0$=1) or always selected (e.g., $\overline{\text{CSO}}_0$=0) by programming the addresses with "hard deselect" or with the "don't care" patterns, respectively. This is similar in function to a PC-board wire jumper. If unused $\overline{\text{CSO}}$s are programmed with all addresses "don't care", then switching is eliminated and power consumption reduced for those lines.

Since the PAD is always powered up when the device is selected ($\overline{\text{CSI}}$=0), $\overline{\text{CSO}}$s are always active and their state is a direct function of the PAD configuration and current address line inputs.

System Applications

The MAP168 device is designed to reduce memory access time and board area utilization in high performance digital signal processor, microcontroller and microprocessor systems. These systems typically have the following requirements:

- □ 16-bit data path
- □ 64K to 1 Meg address space
- □ Fast memory access time (100ns to 40ns)
- □ Decoding for I/O and memory
- □ Printed circuit board area limitations
- □ Multiple types of memory, including EPROMs and SRAMs for program and data store.

The DSP System Architecture shown in Figure 6 illustrates a typical system based upon a 40MHz TMS320C25 digital signal processor. Such a system allows only 40ns for memory access time. The access time must be broken down into decoding time and memory-access time. The fastest decoders available today require approximately 10ns to complete their decode function. Due to this decoding time, memory access time for both the EPROM and SRAM must be 30ns or less. The WSI-MAP family of products performs decoding on-chip with no speed penalty. As a result, the performance of a 40ns MAP168 device in the above example is equivalent to a 10ns decoder and a 30ns EPROM and SRAM memory. In addition, the package equivalent of two fast EPROMs, two fast SRAMs and at least one decoder are combined into one MAP168 chip resulting in at least a 5-to-1 component count reduction.

System Applications (Con't)

High-Speed, Word-Oriented Application

The MAP168 device is especially suited for high-speed word-only microprocessors. The TMS320C20/25 DSP family is an example of such a microprocessor. Interfacing the MAP168 device to a TMS320C25 operating at 40MHz with no wait states is illustrated in Figure 7. The TMS320C25 has two pins for selecting Program Memory (PS) and Data Memory (DS). These functions are connected to the higher order address of the MAP168 device. PS is connected to A_{18} and DS is connected to A_{17}. Usually PS will select the EPROM and DS will select the SRAM. The PAD permits partitioning of the MAP168 memory to accommodate virtually any system address map. Figure 8 shows two possibilities.

Figure 6. DSP System Architecture

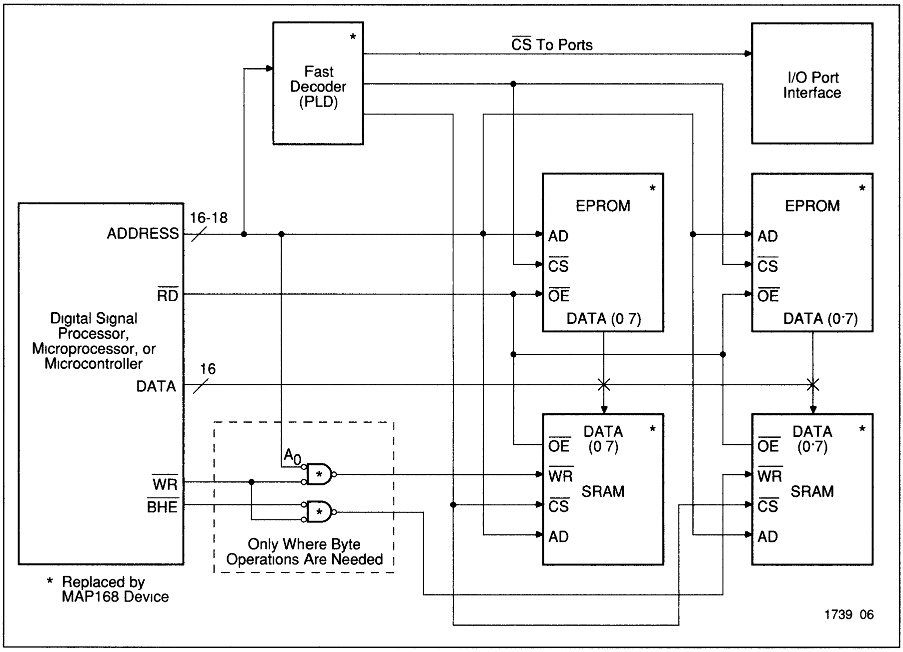
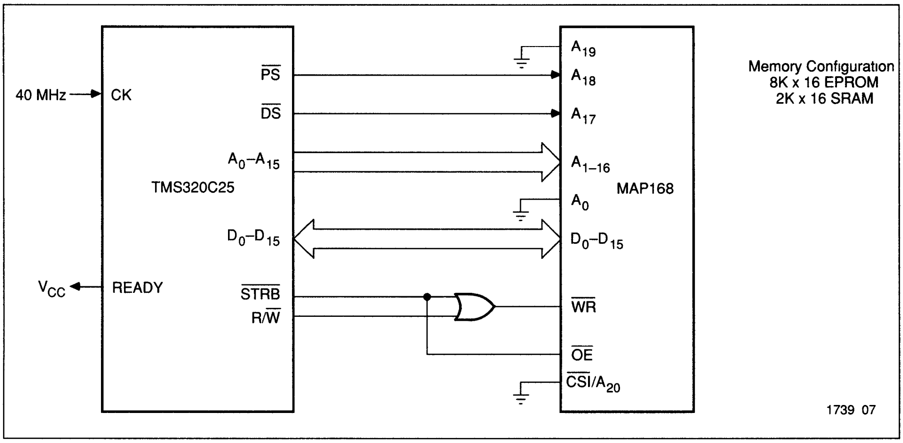


Figure 7. TMS320C25 Interfacing x16 Configuration



System Applications (Con't)

When in a word-wide (x16) configuration, the total memory available on the MAP168 device is 8Kx16 of EPROM and 2Kx16 of SRAM. The implementation shown in Figure 7 replaces at least five circuits:

- ❑ One high-speed decoder (10ns)
- ❑ Two 8Kx8 EPROMs (30ns)
- ❑ Two 2Kx8 SRAMs (30ns)

If the system was previously implemented using a boot EPROM, the MAP168 device replaces ten circuits:

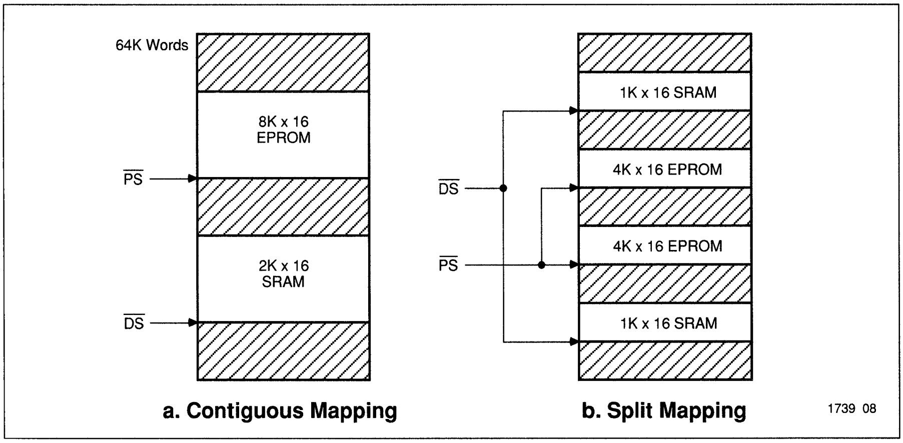
- ❑ One high-speed decoder (10ns)
- ❑ Two 8Kx8 EPROMs (30ns)
- ❑ Two 2Kx8 SRAMs (30ns)
- ❑ Two 8Kx8 slow EPROMs
- ❑ Three ICs for Wait-State generation

For expanded memory requirements in a word-wide (x16) configuration, two MAP168 devices can be interfaced directly with a TMS320C25, as shown in Figure 9. The two MAP168 devices provide the total system memory. Key features of this system are:

- ❑ 40ns access time
- ❑ 16Kx16 EPROM
- ❑ 4Kx16 EPROM
- ❑ 16 general purpose programmable chip selects

The general-purpose programmable chip select outputs can be mapped to any location in the address space via the PAD. These chip selects can be used to access I/O ports, select additional memory or control other system functions.

Figure 8. Memory Mapping With MAP



a. Contiguous Mapping

b. Split Mapping

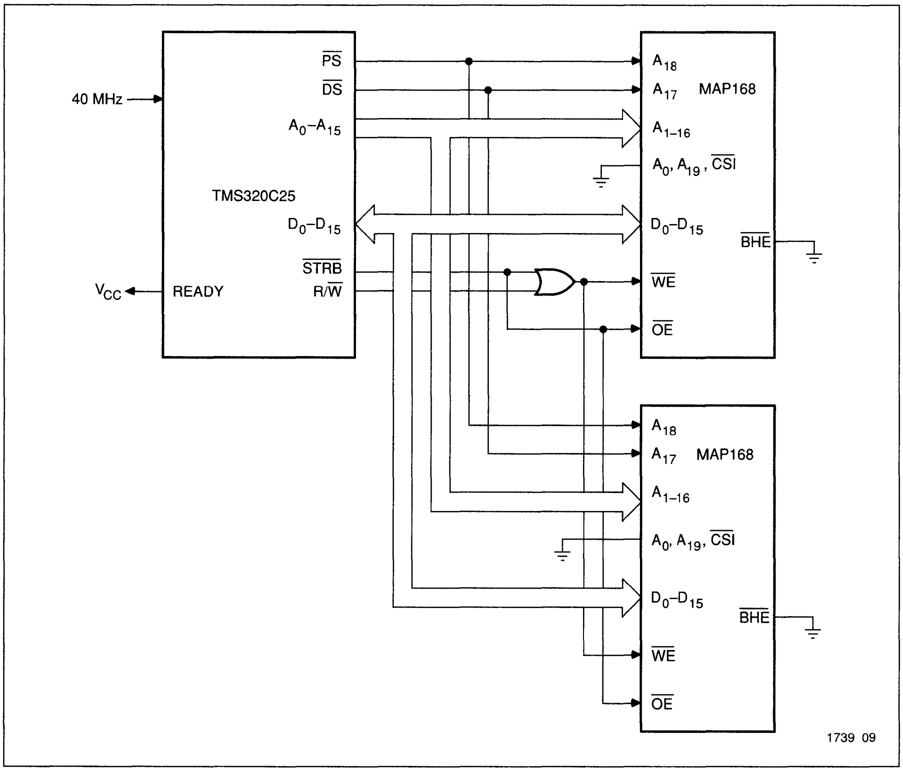
Microcontroller Application

The MAP168 device has two basic configurations. They are a word-wide (x16) configuration with byte operation capability and a byte-wide (x8) configuration with 8 chip select outputs.

The 128K address space (during byte operations in the word-wide mode) makes the MAP168 device especially suited for microcontroller applications. Figure 10 illustrates a simple interconnection of the MAP168 device to a microcontroller. The HPC16040 operating without wait states requires a memory access time of 65ns or better. This makes the MAP168 device a good fit, since it offers an access time of 40ns, leaving a 25ns margin.

The MAP168 device can be configured in a byte-wide (x8) mode and can also be doubled-up with a second device.

Figure 9. DSP with Expanded Memory



Microcontroller Application (Con't)

Embedded Controller Application

An embedded controller is an intelligent section of logic, usually based around a processor, dedicated to a particular task and is not accessible for software alteration by the user. Such applications are generally complex and are becoming more common in system design. Typically, embedded controllers are high performance systems designed under severe space/power constraints. On the other hand, they have a limited ability to be upgraded and limited program memory. This makes them ideal candidates for the WSI-MAP device implementation. The MAP168 device has the following key features which are useful in such an application:

- 1M address space decoding
- 40ns access time
- Byte operations in word-wide mode (BHE)
- One output chip select when in the word-wide mode ($\overline{\text{FCSO}}$)
- Nine output chip selects when in the byte-wide mode
- Programmable Address Decoder (PAD)

A popular processor for embedded applications, due largely to its extensive software library and development support, wide availability of compatible peripherals and low cost from volume production is the 80186 from Intel. Figure 11 shows how a MAP168 device can be interfaced to an 80186.

The UCS (Upper Chip Select) is connected to $\overline{\text{CSI}}/A_{20}$ on the MAP168 device. The PAD is programmed to locate a 1Kx16 EPROM slot in the upper memory address space for a reset subroutine. The rest of the memory can be located as required by the user. Figure 12 shows one possibility.

WSI-MAP Family Development Support

WSI provides the development environment needed to program the WSI-MAP family products. A menu-driven software package known as MAPLE is available under the WISPER top-level software. It operates on the popular IBM-PC® as a platform and includes extensive documentation on installation and operation. It generates configuration files for use by the programming tools. These programming tools include the MagicPro™ programmer hardware and the MAPPRO software. They enable the user to program the PAD and the EPROM. For additional information, consult your nearest WSI sales representative.

Figure 10. Microcontroller Interfacing x16 Configuration

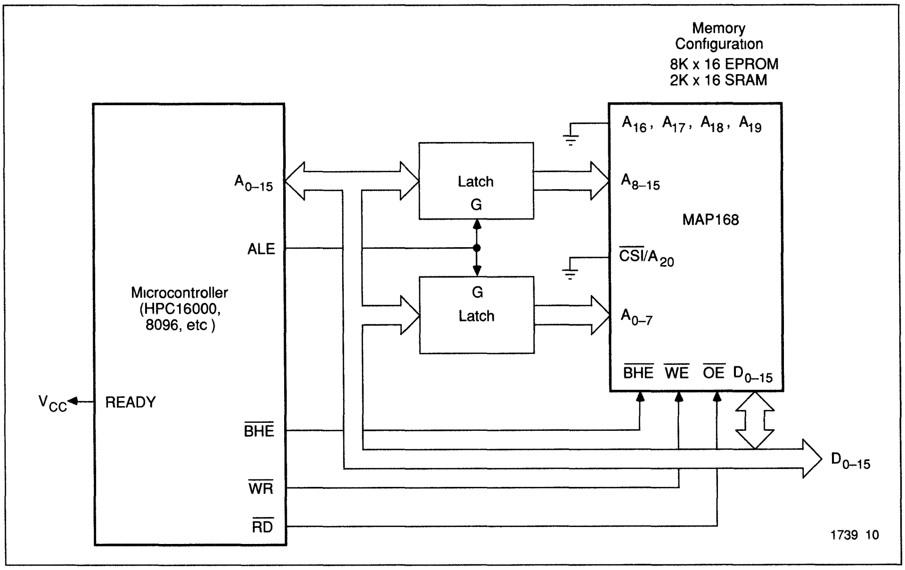


Figure 11. Interfacing To An 80186 x16 Configuration

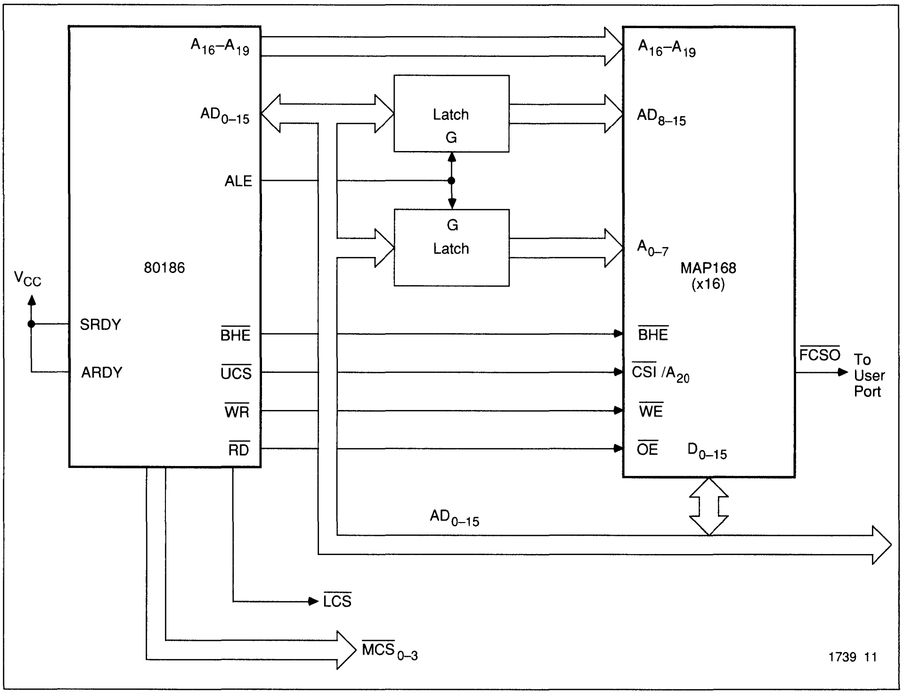
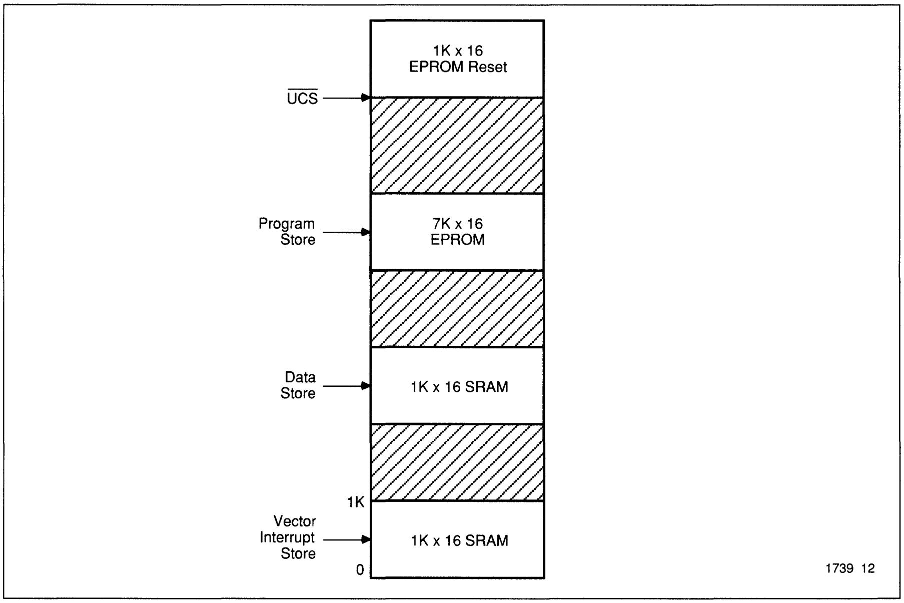


Figure 12. Optional Memory Mapping For An 80186



WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Note 010

PAC1000 Introduction

By Chris Jay and David Fong

Abstract

The PAC1000 user configurable high performance microcontroller, from WaferScale Integration, is the first of a generation of devices intended for applications in high end embedded control. Understanding the device architecture and using its support tools require some practical experience before a full system design is attempted. This application note is intended to introduce the device and its architecture along with the support software tools to the systems designer. Finally to develop some simple applications leveled at common problems found in system design.

Introduction

The PAC1000 has many applications in digital systems where high speed processing, interface or control is required. The two roles of the device are in a standalone mode where the PAC1000 is programmed to control data flow to or from other systems, or as a high speed peripheral working with a host microprocessor. Frequently, many systems designers cannot find the ideal solution to their requirements in a standard chip. The designer may look at creating the required function from discrete logic, a combination of a number of PAL/EPLD devices, Programmable Gate Array (PGA) products or standard gate array. In each alternative, the designer is trying to reduce the chip count of the system solution and hence increase its reliability and reduce assembly costs.

The discrete TTL or CMOS logic solution to a systems design is considered by some to be an old fashioned approach but still popular with many digital design engineers. However, designs using this technology can quickly escalate in chip count as the development progresses and once a system is designed it is very difficult to modify because the finished printed circuit board contains devices that cannot be re-programmed or altered in any way. Also, a revision or system upgrade will require a new printed circuit board design.

The PAL/EPLD solution reduces the chip count over a solution that uses discrete logic but still many devices are used because the PAL/EPLD products are not very register intensive. Small subsystems such as FIFO or a STACK require a number of PAL/EPLD devices and additionally require some additional chips. An alternative solution would be to use additional dedicated chips like FIFO, ALU and SRAM, leaving the PLD/EPLD devices to handle the glue, interface and small state machine functions. The Programmable Gate Array brings the system down to a possible acceptable level but system logic still has to be defined and routed in the logic cells and a number of PGA devices have to be designed such that they all work together. Nevertheless, in the case of the programmable solution, subsystems such as STACK, ALU, REGISTER FILES etc., might still need to be configured in the gates and registers of these devices. This can cause an escalation in the quantity of these chips used in the final system, because PLDs and PGAs are not good vehicles for integration at the subsystem to system level. In a gate array design the turn-around time is longer than the programmable solution, and because the device is not re-programmable there is a high level of risk in going to a gate array solution. Also, the high 'up front' Non Recurring Engineering charges NRE can rule out the use of gate array.

The Programmable Standalone Controller offers the most likely solution to the problem facing the systems designer. Very often both the PAC1000 is used with programmable logic devices to effect an overall solution. For example in some modes of operation PLDs are used for address decoders to select and gate the host interface control lines such as CSB, RDB, and WRB. By bringing the package count of the system down to its lowest

Introduction (Cont.)

level the design cycle time reduces, so minimizing the overall time to market of the final product. The reason for this is that the PAC1000 already contains the subsystems necessary for a fully functional system design, and being programmable, it can be adapted to perform most functions required from systems devices.

The PAC1000 comprises elements such as FIFO, ALU, register files, STACK, microcode store, loop and breakpoint counters, special registers and interface logic all interconnected by a general purpose internal bus structure. The instructions that control data flow are contained in the EPROM section of the microcontrol store. These instructions are entered into the system by the designer as assembly or high level language code. There also exists a microcode entry level for those designers who are used to microprogrammable designs. Designing with the PAC's software support tools is very similar to writing code for microprocessors. The end result is an assembled listing which can be simulated prior to programming into the PAC1000 device's on chip EPROM. The difference between microprocessors, conventional microcontrollers and the PAC1000 device is found its ability to execute instructions in parallel, and to offer the designer a flexible architecture. Microcontrollers and microprocessors function on single operations of execution, but the PAC1000 executes three instructions in parallel during the current clock cycle. In this way the PAC1000 device needs fewer EPROM locations to store the code which performs a given function. In addition high functional speeds can be obtained because the device can execute those instructions at the clock rate of the system.

PAC1000 Device Architecture

The PAC1000 device architecture can be divided into three subsystems, see Figure 1a; a CPU section that is similar to those found in microprocessors, a host interface, and a programmable instruction control unit. The instruction register can be clearly identified with its three output sections of control, output and CPU Operation Definition. Figure 1b illustrates a more detailed diagram of the system than Figure 1a, clearly identifying the sub structures of the three subsystems. The different sections of the PAC1000 are interconnected to each other by internal buses and convey data and instructions to and from each other. Communication to and from the outside world is achieved through various input and output registers, and a Command/Data FIFO.

The Control Unit

The control unit is constructed around a 1K deep 64-bit wide EPROM, see Figure 1b. The 64-bit wide instructions are programmed in the EPROM section and are accessed and executed on each clock cycle. The input RESET causes the PAC1000 to access and execute the first instruction at location 000H of EPROM. On each execution cycle, the Instruction Register shown in Figure 1a will contain three control operatives, a next address instruction to the control section, an output instruction and CPU instruction. The other inputs to the control unit include interrupts and condition codes. There are four external and four internal interrupts that can be enabled under programmed control. These can generate a branch to an interrupt service routine that results from a rising edge applied to the external interrupt input. For interrupts INTO, INT1, INT2, and INT3 there are four locations 008H, 009H OOAH and 00BH respectively. These are the vectored addresses at which processing will continue in the presence of one of these active interrupts. At the interrupt location a jump to an interrupt service routine should be inserted. For example, the occurrence of INTO will divert processing to location 008H, that location may contain a JMP 100H, where 100H is the address where the service routine for INTO should reside. The internally generated interrupts are INT4, INT5, INT6 and INT7 which divert processing to locations OCH, ODH, OEH and OFH respectively. Details of their allocated function is given in the PAC1000 data sheet. In addition there are eight condition code inputs CC[7:0], shown alongside the INT[3:0] inputs in Figure 1b. These inputs can be tested individually under program control. The combination of Next Instruction Definition, Interrupt and Condition Code

The Control Unit (Cont.)

input direct the flow of the program and hence the execution of instructions contained in the EPROM section. The CASE logic is used in the controller section to enable CASE statements to be executed on condition code groups. The eight condition code inputs may be divided into two four bit groups. Case group zero CG0 comprises CC0, CC1, CC2 and CC3. Case group 1 CG1 comprises CC4, CC5, CC6 and CC7. A further two case groups CG2 and CG3 test flag registers (see Table 1). These condition code inputs may be tested individually or tested in a group. When tested in a four bit group, a one-of-sixteen branch will occur, as specified by the CASE instruction.

The current status of the PAC1000 is kept in the sixteen bit status register. STAT0–STAT11 give twelve status bits with four extra bit locations for future development. Table 2 shows the assignment of each register.

Figure 1a. PAC1000 Microcontroller Single Cycle Control Architecture

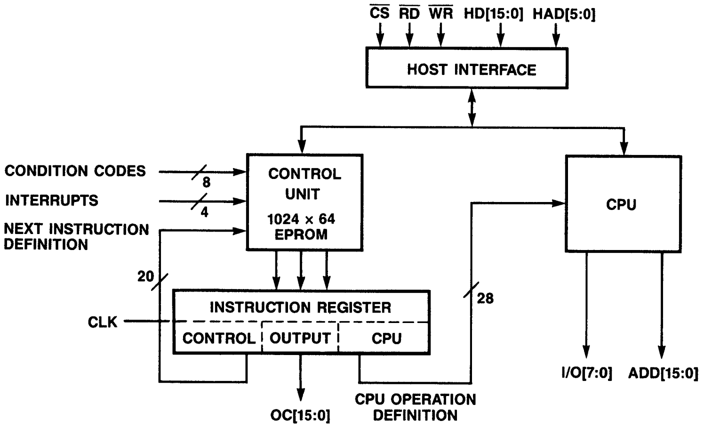


Figure 1b. PAC1000 Microcontroller Block Diagram

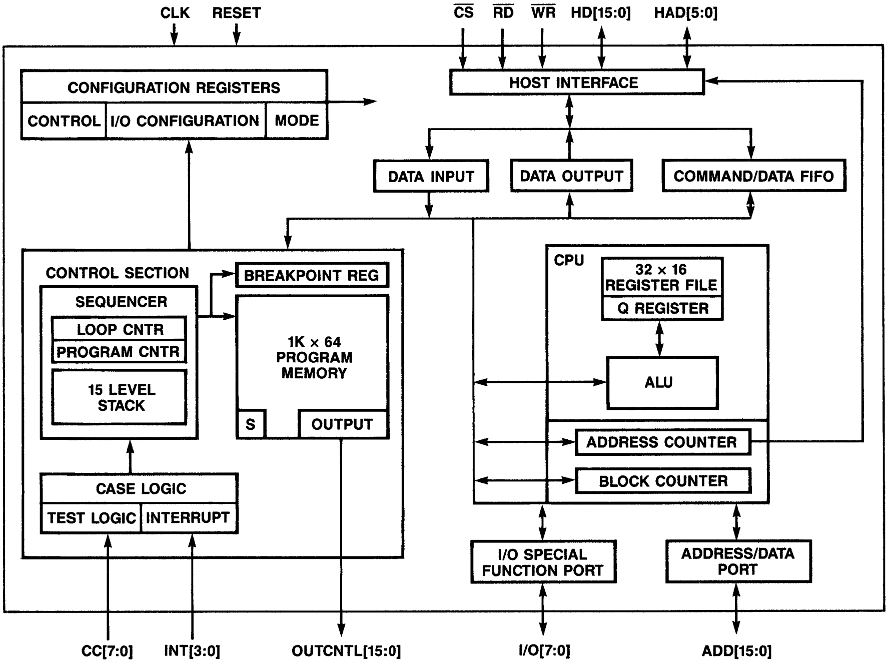


Table 1. CASE Group Assignments

Condition Code	CASE
CC0, CC1, CC2, CC3	CASE Group 0
CC4, CC5, CC6, CC7	CASE Group 1
S, O, Z, CY.	CASE Group 2
INTR, BCZ, FIOR, FICD.	CASE Group 3
FIXP, ACO, STKF, FIIR, DOR, INTR	N/A

Table 2. Status Register

0	0	WSI Reserved	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

S11 — Security Bit, High is Active Security On, Low is No Security.
S10 — Scan Mode, High is Active On, Low is No Scan Mode.
S9 — FIXP FIFO Exception Occurs When a Command is Written, a Low Means No Exception.
S8 — FIIR FIFO Input Ready When There is at Least One Location Vacant.
S7 — CY Set High When the Result of a CPU Operation Generated a Carry.
S6 — Z Set High When the Result of a CPU Operation is Zero.
S5 — O Set When an Overflow Has Occurred During a Two's Complement Operation.
S4 — S Sign Bit Set to One When the Result is a Negative Number.
S3 — Stack Full Flag. Set When the Stack is Full.
S2 — Breakpoint Flag is Set When the Address in the Breakpoint Register is Equal to the Address in the Program Counter.
S1 — BCZ is Set When the Block Counter Reaches Zero.
S0 — ACO Address Counter All Ones Flag is Set When the Address Counter Reaches the Maximum Count.

The Control Unit (Cont.)

A single internal counter is provided for loop control, this is part of the control section, and is shown in Figure 1b. If a FORLOOP is executed the loop counter is loaded and the instructions within that loop are executed until the counter has decremented to zero. The loading of this counter is transparent to the designer in the respect that the FORLOOP instruction automatically performs loading and counting.

A fifteen level stack is incorporated to hold the return address of the main program when a subroutine call or interrupt service routine is being executed. The address of the next sequential instruction to be executed is pushed onto the stack. The stack is also used for LOOP NESTING. There is only one loop counter in the PAC1000 but nested FORLOOP instructions are possible because the current contents of the loop counter is saved in the stack when the next subsequent loop in the next is entered. When leaving the loop the stack is popped to return the old count back into the loop counter thus preserving its original contents. When the stack becomes full a status flag STKF is set in the sixteen bit status register and an interrupt level 7 is generated.

To enable a debugging facility a register called the breakpoint register is included in the microcode section. When the contents of the program counter is equal to that of the breakpoint register an interrupt level six is generated. For debugging purposes a level six interrupt service routine should be written to perform diagnostic tests within the system.

Host Interface

The host interface section has been designed to easily integrate into a CPU based system. When the PAC1000 is used in the peripheral mode, the flow of data or commands to its internal registers may be achieved through an internal FIFO. Standard microprocessor signals of chip select CSB, read RDB and write WRB (active LOW CS,

Host Interface (Cont.)

RD and WR) are accompanied by a sixteen bit Host Data and a six bit Host Address bus. Table 3 gives the conditions governing the mode setting for both standalone and peripheral mode. The logic condition of HDSEL0 and HDSEL1 in the control register will determine the mode of the PAC1000 operation. Bit positions in this register can be set or reset under program control.

A detailed block diagram of the PAC1000 is given in Figure 2 which illustrates the internal structure of the control section, processor section and interface. Data flow from the host processor data inputs HD0–HD15 to the internal 16-bit bus can be achieved through the FIFO section. The FIFO is eight locations deep and twenty-two bits wide. To transfer data words to the registers in the CPU section the host processor uses the chip select, write and HAD inputs. The address of the register is set up on the five HAD lines (this selects one of 32 registers) then the write and chip select lines are driven LOW. The data on the HD lines plus the register address is loaded into the FIFO. An additional bit called the FICD bit is loaded through HAD5 at the same time as address HAD[0–4] and the host data lines HD[0–15]. This is the FIFO Command/Data bit and must be LOW to signify that the sixteen bit word on HD[0–15] is data. If it is set HIGH, the least significant ten bits of that data will be used as an address pointer to the microcoded EPROM. In this way the host system can direct PAC1000 processing to a defined microcoded address. This is a powerful feature that enables dynamic context switching of PAC1000 under supervision of the host processor. The FIFO exception flag FIXP will be set if the information residing in the FIFO was misdirected (if it were treated as a control word when the FICD flag labeled it as data or if the opposite condition prevailed).

Using the FIFO is the only method in which the host can communicate with the PAC1000 using the active LOW chip select CSB and the write input WRB. The DOR and DIR are Data Output and Data Input registers and are available to convey data to and from the internal sixteen bit bus but do not respond to CSB and WRB. The DIR would be used in a synchronous system because, when it is enabled by setting the DIREN flag (see Table 4), data is latched on the rising edge of each clock signal. The data contents of the DOR register may be directed to the host data outputs if all inputs CSB, WRB and RDB are inactive and HDSEL0 and HDSEL1 are 1 and 0 respectively, see Table 3. The use of the DIR and DOR register is intended more for synchronous communication whereas the FIFO is intended primarily for asynchronous systems or synchronous peripheral interface. The flags FIIR and FIOR are the FIFO Input Ready and FIFO Output Ready respectively, these flags can be tested so no overwriting of data will occur. Figure 3 shows the I/O Port and Special Functions. The FIIR register can be directed to the output I/O_7 through a multiplexer so it can be tested externally by the host system.

Table 3. Host Interface Function Table

HDSEL0	HDSEL1	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	HAD5	HAD[0–4]	HD[15–0]	OPERATION
0	0	0	1	0	0	Register Address	Data	Write Data to FIFO
0	0	0	1	0	1	X	Command	Write Command to FIFO
0	0	0	0	1	0	00100	X	Reset FIFO
0	0	0	0	1	0	00011	X	Reset Status Register
0	0	0	0	1	0	00010	X	Read Program Counter
0	0	0	0	1	0	00001	X	Read Status Register
0	0	0	0	1	0	00000	X	Read Data Output Register
1	0	1	1	1	X	X	X	Data Output Register
0	1	1	1	1	X	X	X	Status Register
1	1	1	1	1	X	X	X	Program Counter

Figure 2. PAC1000 Detailed Block Diagram

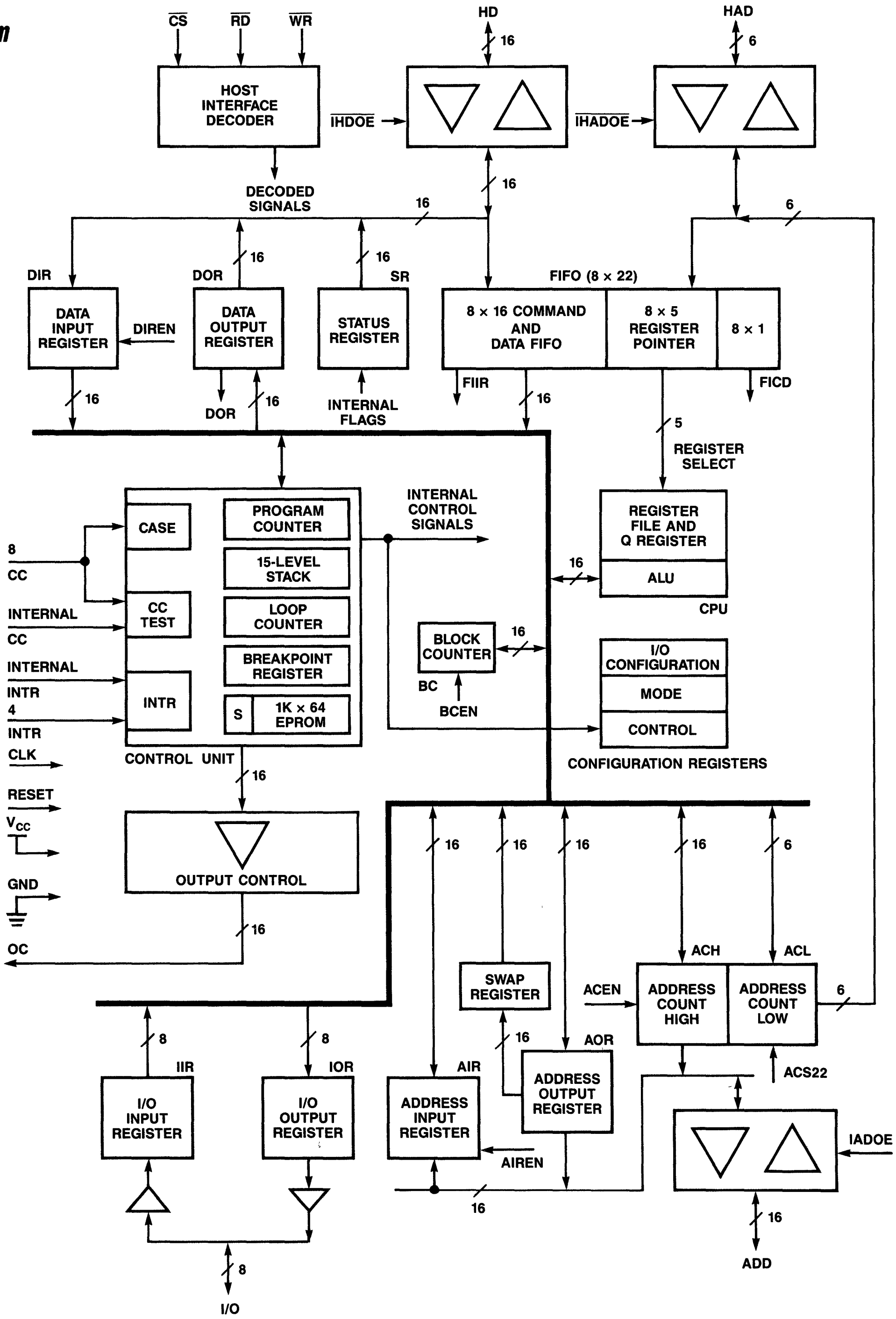


Figure 3. I/O Port and Special Functions

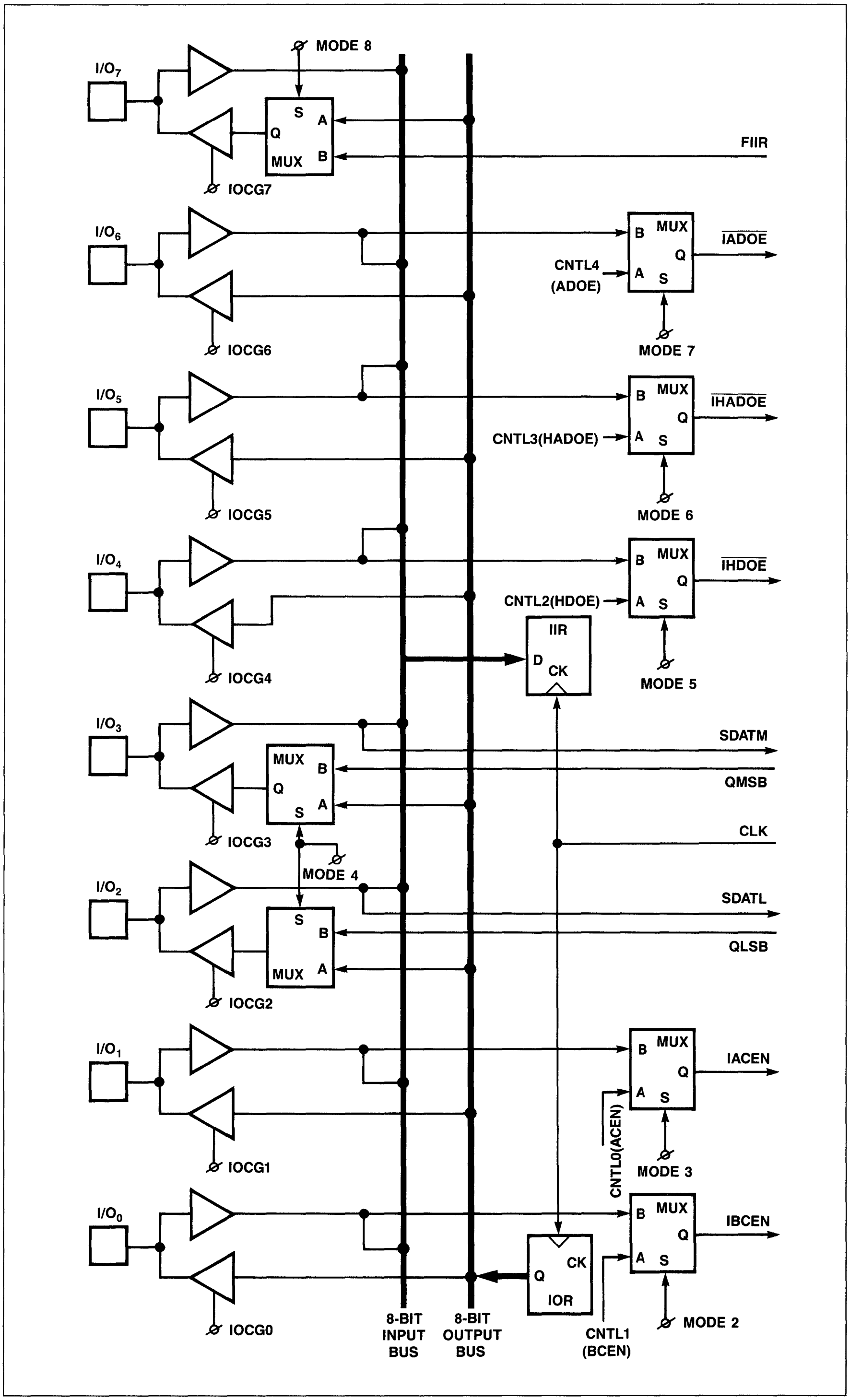


Table 4. Control Register

CTRL9	CTRL8	CTRL7	CTRL6	CTRL5	CTRL4	CTRL3	CTRL2	CTRL1	CTRL0
ASEL	AIREN	DIREN	HDSEL1	HDSEL0	ADOE	HADOE	HDOE	BCEN	ACEN

ASEL — Selects Which Source Will Write to the Address Bus
1 = Address Counter. 0 = Address Output Register.

AIREN — Enables/Disables Writing to the Address Input Register by the Address Bus.
1 = Enabled. 0 = Disabled.

DIREN — Enables/Disables Writing to the Data Input Register.
1 = Enabled. 0 = Disabled.

HDSEL1
HDSEL0 — Decoded to Select Which Source Will be Connected to the Host Data Bus (See Table 3.).

ADOE — Selects Direction of the Address Bus
1 = Output. 0 = Input.

HADOE — Selects Direction of Host Address Bus (HAD).
1 = Output. 0 = Input.

HDOE — Selects Direction of Host Data Bus for Next Clock Cycle.
1 = Output. 0 = Input.

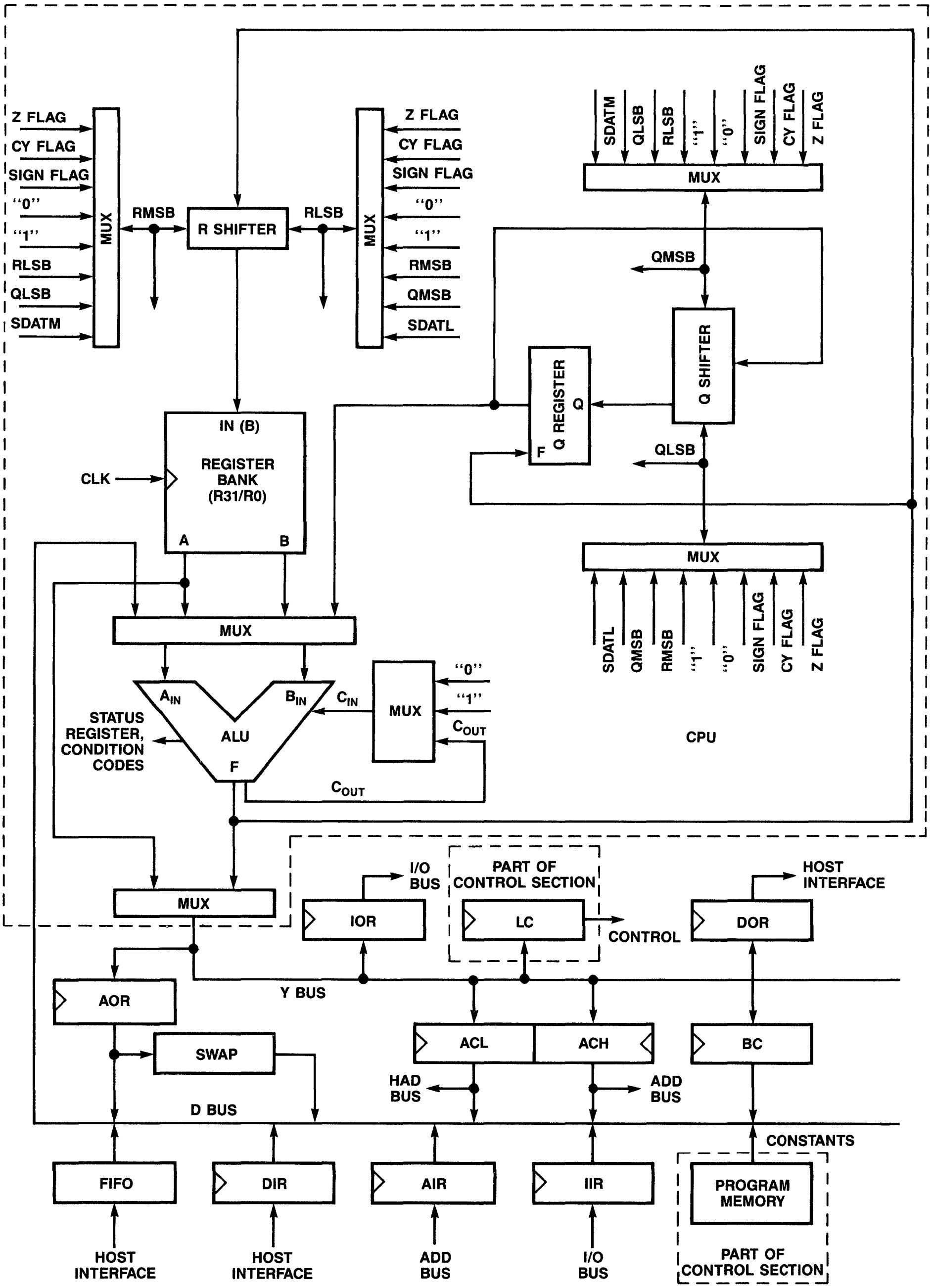
BCEN — Enables/Disables Block Counter Before Next Clock Edge.
1 = Enabled. 0 = Disabled.

ACEN — Enables/Disables Address Counter Before Next Clock Edge.
1 = Enabled. 0 = Disabled.

Central Processing Unit

The section that deals with data processing is the central processing unit. This comprises a sixteen bit wide ALU with a 32 × 16 bit register file. One other special purpose register Q and an R shifter circuit make up this section. The Q register is sixteen bits wide and can be used for data shifting. Figure 4 shows the ALU and register structure of the CPU section. The ALU is in the path of the register outputs such that arithmetic and logic functions may be executed on the contents of any one of the 32 general registers. The output of the ALU passes data back to the selected register through the R shifter. In this logic circuit, data may be shifted either left or right, one position, before being written back into the register file. The output of the ALU can also drive data to registers such as the DOR register. A multiplexer can select either the ALU or the R0–R31 register output. The loop counter LC can be loaded from this multiplexer enabling the contents of a register to determine how many program loops are to be executed. This loop counter can be loaded from the EPROM to give a fixed number of loops or from a register at program 'run time.' In this event, the number of times a loop is executed can be made programmable. Other registers on this bus are AOR, Address Output Register, the IOR, Input Output Register, the ACL and ACH low and high address counters and the BC Block Counter. The ACL counter has a six bit resolution and the ACH counter has sixteen. When enabled by ACEN, the ACH counter will increment on the rising edge of each clock cycle. The default value is for a sixteen bit count. To enable a twenty-two bit count where the ACL takes on the six least significant of the twenty-two bits. The ACS22 flag must be set, to enable the clocking of these counters. This is transparent to the software because once enabled the counters will clock at the system clock rate. However, they can be turned on and off from the microcoded instruction of enable SET ACEN, or disable RESET ACEN, also counting can be influenced by register loading.

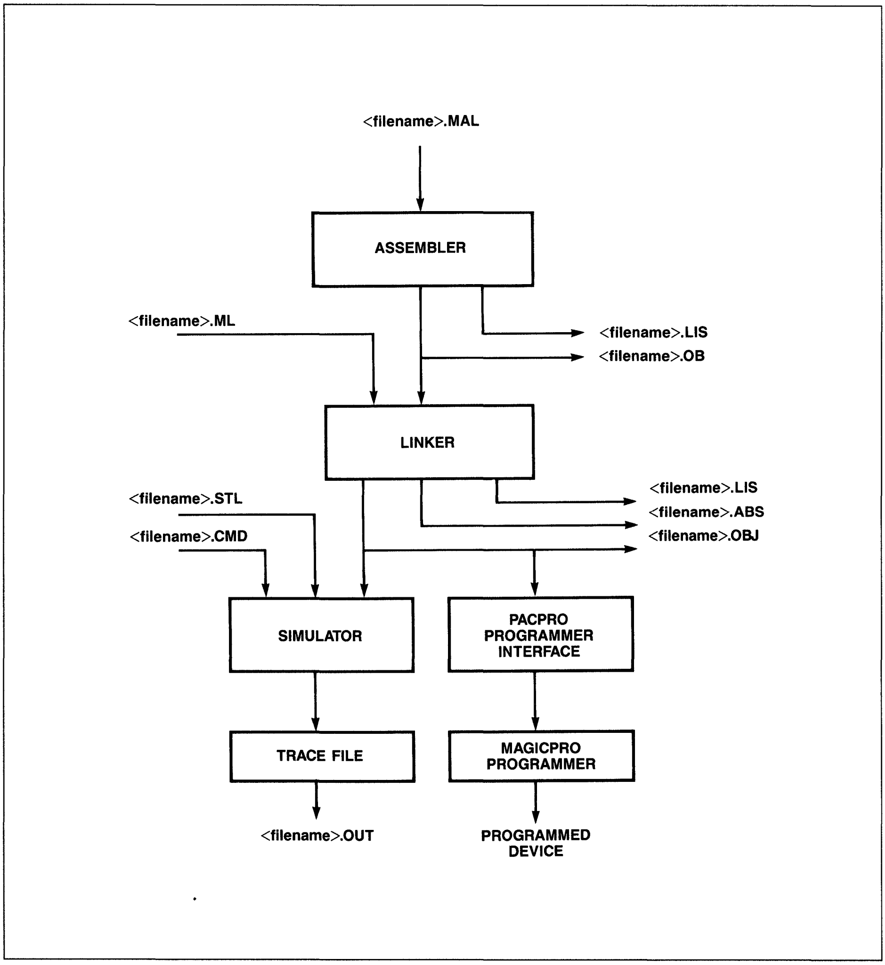
Figure 4. PAC1000 ALU and Registers Structure



Support Software

The PAC1000 device is supported with development software that can run in an IBM PC/XT or AT computer. The main tools that the designer will use are the assembler, the linker and the simulator. These support programs are run from a WSI menu called WISPER that has been designed to make software development a simple process. The designer can select the assembler from the menu and assemble his source program. After assembly the program must be linked. The linker program is designed for those system designers who build their software up from a number of modules. Figure 5 illustrates the flow from original source code entry through the linker to a simulated output. The linker will take these modules and combine them into one object program. On completion of assembly and linking the program may be checked by the simulator. The use of the simulator removes the need for EPROM programming and in-circuit testing during the design cycle and gives the designer a fairly high level of confidence that the program will function as intended. The simulator will take the bit pattern format that was generated during assembly and apply a command and stimulus file to the program. The result will be a series of waveforms that appear on the screen of the PC and is similar to that of a logic analyzer display. A table of vectors is also generated for those signals that are traced from the command file. These vectors can be printed out for analysis and verification.

Figure 5. Program Flow From Assembly Input to Simulated Output



Microcoded EPROM Section

A further aid to the design entry is the ability to mix high level, assembler and microcode mneumonics so designers can use the entry level that they feel the most comfortable with. Most of the applications example gıven below are written in a high level 'C' like language but some assembler instructions are also incorporated.

In systems applications such as Direct Memory Access (DMA), it is required to output the contents of a counter to address memory and then increment it. This is implemented in the PAC1000 high level language syntax as:

```
AOR := R0     ; /*CONTENTS OF R0
                 GOES INTO THE AOR*/

R0 := ++R0  ; /*REGISTER R0 IS
                 INCREMENTED BY ONE*/
```

For efficiency these two instructions may be combined into one line of code, which is executed in one clock cycle:

```
AOR := R0 := ++ R0 ;
  /*COMBINING THE TWO OPERATIONS*/
```

The contents of R0 will be passed to the Address Output Register and will be incremented by the ALU.

Where AOR is the address output register and R0 is one of the thirty-two, 16-bit general purpose registers. The '/*' symbol delimits the comment field boundary.

With a PAL/EPLD/PGA approach the designer would be required to spend much valuable time configuring a loadable binary counter, with a 3-State output capability.

In applications such as digitizer/plotter systems, x,y coordinates have to be quickly summed or subtracted many times to register cursor movements and position. This requires repetitive arithmetic operations. In this application vector addition on two or more sixteen bit words can be defined as two instructions:

```
R0 := R0 + R1 ;
AOR := R0 ;
```

Combining these instructions together:

```
AOR := R0 := R0 + R1 ;
```

With conventional programmable logic an ALU function would have to be designed or a dedicated custom chip used with the programmable logic part used as the data I/O controller. The key point of this issue is that complex logic functions are simply written as a few single lines of statements. Moreover, a combination of functions may be grouped in a single line. These include a microcontrol directive such as a branch, call to subroutine or JUMP on condition, an ALU function such as increment or add, and an output control command. There are sixteen output control lines which can be driven active on each clock cycle. The composite of the three commands are:

```
LABEL:   JMPNC CC7 LABEL ,
         R0 := R0 + 1 , OUT 'HOLD' ;
```

The function of this line of code would be to wait until the condition code input of CC7 went active before the next instruction is executed. At the same time the contents of R0 would be incremented and the output control lines would be driven with a sixteen bit code called HOLD. An equates option 'equ' is used to define uniquely a sixteen bit pattern called HOLD. The assembler encodes an equate statement to allow meaningful words to be used in output control statements. Some examples of this are:

```
HOLD equ H'FFFF' ;
  /* HOLD IS SET AS HEX FFFF */

ENBL equ H'EFFF' ;
  /* ENBL IS SET AS HEX EFFF */
```

The equates directive should be declared at the start of the program before any actual code is written.

Applications Programs

The depth of the microcontrol store is 1K of 64-bit wide words. One 64-bit instruction is executed on each clock cycle. The instruction word is subdivided into three commands: an output control command, a command to the processor section and a next address command to the microcoded memory. Figure 1a shows the Instruction Register with its contents of control, output and CPU commands. The control unit will also respond to condition code inputs and interrupts. An example of output control and response to condition codes is in a handshake loop. The output stimulus can be to set one of the control outputs OC[15:0] and wait for a response to a condition code input CC[7:0]. Under program control a conditional JUMP to a location could result if the bit tested were set. Otherwise linear programming could continue.

The first applications program below demonstrates the use of condition code zero CC0 to test for a start condition. When the input is LOW, the program loops continually testing CC0. When the host raises CC0, the program performs a double precision addition. The sum is available at the data output register DOR.

```
segment pacdes01 :

/* PROGRAM TO PERFORM DOUBLE PRECISION ADDITION ON THE REGISTER*/
/* CONTENTS OF R1.RO: R3.R2 THE CARRY OF THE LEAST SIGNIFICANT */
/* WORD ADDITION IS CONTAINED IN THE CP REGISTER AND IS USED IN*/
/* THE SECOND HALF OF THE 32 BIT ADDITION.                     */
/*                                                             */
/*              PIN FUNCTIONAL DESIGNATIONS.                   */
/*                     INPUTS.                                 */
/*                                                             */
/*              CCO - ACTIVE HIGH - START 32-BIT ADDITION      */
/*              /CS - ACTIVE LOW  - PAC1000 CHIP SELECT        */
/*              /RD - ACTIVE LOW  - READ A REGISTER FROM HOST  */
/*              HAD[5:0] - INPUTS TO SELECT DOR REGISTER FROM  */
/*              HOST INTERFACE                                 */
/*                                                             */

HOLD:     JMPNC  CCO HOLD :         /*WAIT FOR START CONDITION  */

           RO := H'F830' :          /*LOAD REGISTERS WITH DATA  */
           R1 := H'982F' ;          /*RO AND R2 CONTAIN THE     */
           R2 := H'A309' :          /*LEAST SIGNIFICANT WORD OF */
           R3 := H'4500' :          /*THE 32 BIT LONG WORD AND  */
                                    /*R1 AND R3 CONTAIN THE MOST*/
                                    /*SIGNIFICANT WORD          */

                 R5 := R1 ;
                 R4 := RO :
                 DOR := RO := RO + R2 : /*LOAD DOR REGISTER*/
                 R1 := R1 + R3 + CP :

LOOP1:   JMPNC DOR LOOP1 :          /* WAIT FOR HOST TO READ DOR */

                 DOR := R1 ;        /* LOAD MOST SIG WORD INTO DOR */

LOOP2:   JMPNC DOR LOOP2 :          /* WAIT FOR HOST TO READ DATA */

FIN:     JMP HOLD :                 /*END OF THE CYCLE*/

end :
```

Applications Programs (Cont.)

The program adds the contents of R0 and R2, then R1 and R3 and the CARRY bit. In the next design example, double precision subtraction is performed and this time the CY flag will hold the borrow bit. This design example is more practical than the example above because instead of performing arithmetic on fixed values the register file may be loaded from a source. The configuration of the PAC1000 is in the peripheral mode and data is loaded into the FIFO. CC0 is monitored and, when active, is a signal to the PAC1000 that data has been loaded. The FIFO is unloaded into the registers by the series of instructions:

```
FOR 3   ; /*EXECUTE THE LOOP
           FOUR TIMES*/

RDFIFO  ; /*UNPACK DATA FROM
           THE FIFO*/

ENDFOR ; /*END THE FORLOOP */
```

This section of the program performs a read operation on the FIFO four times. In any FORLOOP N, where N is an integer value, the number of times the loop is executed is N + 1 times.

```
segment pacdes02 ;

/*PROGRAM TO PERFORM DOUBLE PRECISION SUBTRACTION ON REGISTER   */
/*CONTENTS R1, RO ; R3. R2 THE BORROW FLAG IS CONTAINED IN THE */
/*CP REGISTER DURING THE SECOND HALF OF A 32-BIT SUBTRACTION    */
/*                                                              */
/*      PIN FUNCTIONAL DESIGNATIONS                             */
/*              INPUTS                                          */
/*      CCO - ACTIVE HIGH - START PROGRAM                       */
/*      /CS - ACTIVE LOW  - PAC1000 CHIP SELECT                 */
/*      /WR - ACTIVE LOW  - FIFO WRITE                          */
/*      /RD - ACTIVE LOW  - READ A REGISTER FROM HOST INTERFACE*/
/*      HAD[5:0] - INPUTS TO SELECT A REGISTER FROM THE HOST    */
/*                          INTERFACE                           */
/*      HD[15:0] - DATA INPUTS TO FIFO THROUGH HOST INTERFACE   */
/*                                                              */
HOLD:   JMPNC   CCO HOLD : /*WAIT FOR START CONDITION EMPTY     */
        FOR 3   :          /*THE FOUR LOCATIONS OF THE FIFO     */
        RDFIFO ;           /*LOADED THROUGH THE HOST INTERFACE */
        ENDFOR ;           /*SECTION OF THE PAC1000          */
                R5 := R1 :              /*SAVE R1 CONTENTS IN R5*/
                R4 := RO ;              /*SAVE RO CONTENTS IN R4*/
                DOR := RO := RO - R2 ;  /*SUBTRACT LSW PROPAGATE*/
                R1 := R1 - R3 - CP ;    /*THE BORROW INTO THE CP*/

                DOR := RO :              /*LOAD DOR WITH MSW    */

LOOP1:  JMPNC DOR LOOP1 :

                DOR := R1 :              /*LOAD DOR WITH MSW    */

LOOP2:  JMPNC DOR LOOP2 ;

        JMP HOLD :                       /*END OF PROGRAM       */
end;
```

Applications Programs (Cont.)

The next program shows a multiply routine. Although there is no dedicated multiplier in the PAC1000, multiplication can be achieved by shifting and adding. The MUL instruction is a MACRO command that is expanded when assembled into a loop of shift and add instructions. The RDFIFO instruction is used to pass the data from the host to the PAC, which is configured as a peripheral. In the example the contents of R0 and R1 are multiplied and the product is available in registers R1 and R2, where R2 contains the most significant word and R1 the least significant.

```
segment pacdes03 ;


HOLD:    JMPNC   CCO HOLD ;     /*WAIT FOR START CONDITION EMPTY*/
         FOR 1  ;               /*THE TWO LOCATIONS OF THE FIFO */
         RDFIFO ;               /*LOADED THROUGH THE HOST INTER-*/
         ENDFOR ;               /*-FACE SECTION OF THE PAC1000  */
         MUL R2 R1 RO ;
                 DOR := R2 ;      /*REGISTER. THE PRODUCT IN THE */
LOOP1:   JMPNC DOR LOOP1 ;        /*DATA OUTPUT REGISTER         */
                 DOR := R1 ;              /*                     */
SELF:    JMP HOLD ;                       /*END OF PROGRAM       */
end:
```

In the following example, the contents of registers R2 and R1 is divided by the contents of register R0. The most significant word of the 32-bit long word is held in register R2 and the least significant 16 bits are stored in R1. The result of the divide operation leaves the quotient in the Q register and any remainder in register R2.

```
segment pacdes04 ;

HOLD:    JMPNC   CCO HOLD ;     /*WAIT FOR START CONDITION EMPTY*/
         FOR 1  ;               /*THE TWO LOCATIONS OF THE FIFO */
         RDFIFO ;               /*LOADED THROUGH THE HOST INTER-*/
         ENDFOR ;               /*-FACE SECTION OF THE PAC1000  */
         DIV R2 R1 RO ;
                 DOR := Q  ;              /*OUTPUT THE REMAINDER*/
LOOP1:   JMPNC DOR LOOP1 ;
                 DOR := R2 ;              /*OUTPUT THE QUOTIENT. */
SELF:    JMP SELF ;                       /*END OF PROGRAM       */
end;
```

The files generated so far can be entered into the assembler and two files <filename>.LIS and <filename>.OB may be generated as shown in Figure 5. The latter object file must be linked before the final object file is available for programming into the PAC1000's EPROM. The link program <filename>.ML performs this function and is shown below.

```
load pacdes04 ;
place pacdes04 ;
end ;
```

This design example only used one program but many sub-modules may be linked together to form a single executable program. It is possible to simulate the design after linking. The necessary inputs to the simulator are the <filename>.OBJ, <filename>.STL and <filename>.CMD. The latter two files are the input stimulus file and the input command file (see Figure 5). The stimulus file is used to drive inputs such as address, data and condition codes. The command file lists which signals should be traced for observation. Examples of the stimulus file and command file are given below.

The command file shown below will instruct the simulator to set an output trace on the Current value of the Program Counter, CPC. The Condition Code zero input, the write, and the chip select lines are also traced. The simulator also enables a trace to be invoked on registers as well as input

Applications Programs (Cont.)

or output pins. The Q register is traced along with host data, loop counter, and registers R0, R1, and R2. The final trace is set on the host data output register. At the end of the stimulus file, the run instruction informs the simulator to run the driving signals for 140 cycles. The final instruction invokes a View Trace Waveform instruction, so the waveforms may be observed on the PC screen.

```
OPEN STIMULUS PACDES04
SET TRACE CPC
SET TRACE CC0
SET TRACE WRB
SET TRACE CSB
SET TRACE RDB
SET TRACE Q
SET TRACE HD
SET TRACE LC
SET TRACE R0
SET TRACE R1
SET TRACE R2
SET TRACE HDOR
OPEN TRACE PACDES04
RUN 140
V T W
```

The stimulus file is used to apply active signals to inputs of the design. At specific time points conditions are established. For example the statement:

.S CC0 0@1 1 @40

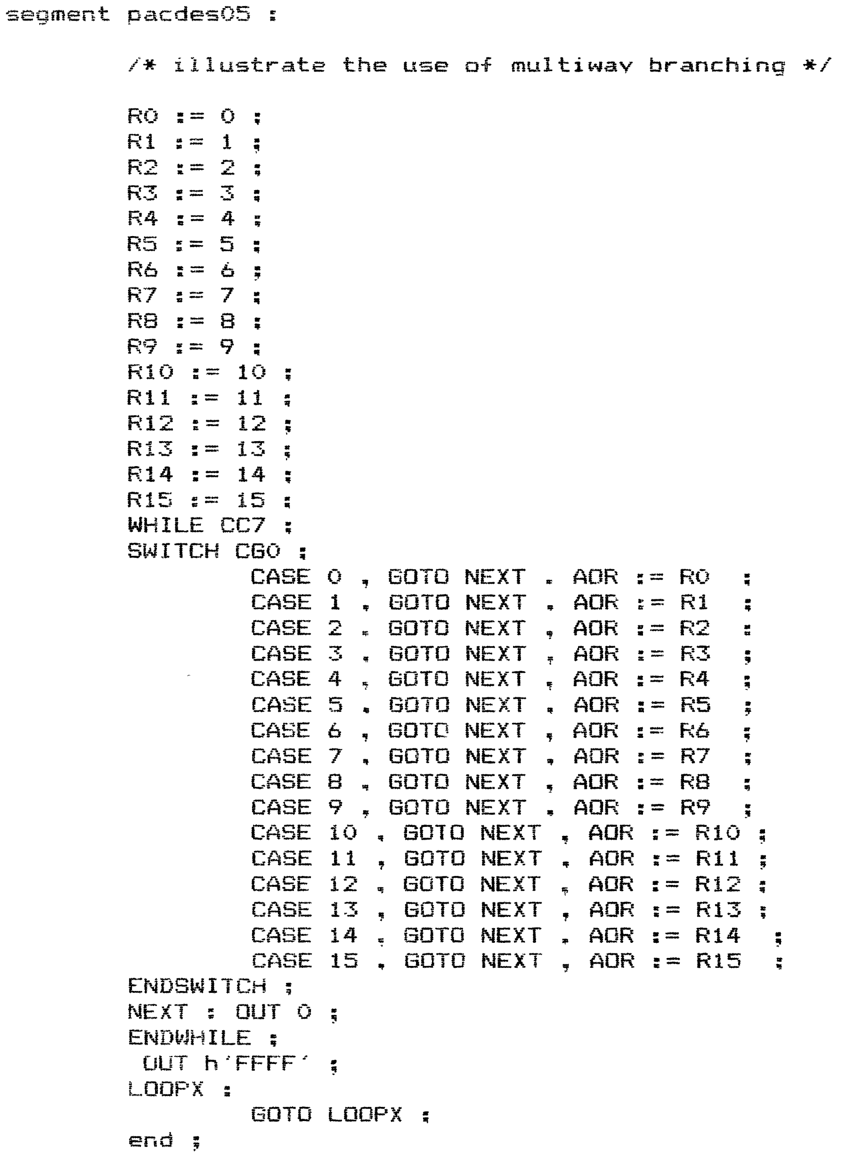
means that the input condition code zero CC0 should become a logic state LOW at time point one and a logic HIGH condition 40 cycles later. A three-state condition can be applied by typing the letter Z in place of logic '1' or '0.' The stimulus file is completed to drive all inputs and applied to the simulator during run time.

```
.S RESETB 0 @ 1 1 @ 2 ;
.S CC0 0@1 1@40 ;
.S WRB 1@1 0@2 1@8 0@12 1 @19 ;
.S CSB 1@1 0@2 1@9 0@11 1@18 0@120 1@129 0@131 1@139 ;
.S RDB 1@1 0@121 1@129 0@131 1@139 ;
.S HAD0 0@1 1@10 0@24 ;
.S HAD1 0@1 ;
.S HAD2 0@1 ;
.S HAD3 0@1 ;
.S HAD4 0@1 ;
.S HAD5 0@1 ;
# WRITE A 7 INTO R0 AND 31 INTO R1
.S HD0 0@1 1@10 Z@70 ;
.S HD1 1@1 Z@70 ;
.S HD2 0@1 1@10 Z@70 ;
.S HD3 0@1 1@10 Z@70 ;
.S HD4 0@1 1@10 Z@70 ;
.S HD5 0@1 Z@70 ;
.S HD6 0@1 Z@70 ;
.S HD7 0@1 Z@70 ;
.S HD8 0@1 Z@70 ;
.S HD9 0@1 Z@70 ;
.S HD10 0@1 Z@70 ;
.S HD11 0@1 Z@70 ;
.S HD12 0@1 Z@70 ;
.S HD13 0@1 Z@70 ;
.S HD14 0@1 Z@70 ;
.S HD15 0@1 Z@70 ;
```

The comment field is denoted by a '#' sign.

Case Statement Logic

The ability of the PAC1000 to perform case statement logic has already been discussed but the following program excerpt illustrates how to encode the case statement. The program will execute when condition code 7 is active high, then case group CG0 is tested for one of sixteen possible states. CG0 comprises CC0, CC1, CC2 and CC3. Sixteen registers are initialized and the output code is driven with zero. When CC7 goes HIGH the CG0 input is tested and the register contents that are equal to the state of the CG0 input is transferred to the AOR outputs.



```
segment pacdes05 :

        /* illustrate the use of multiway branching */

        R0 := 0 ;
        R1 := 1 ;
        R2 := 2 ;
        R3 := 3 ;
        R4 := 4 ;
        R5 := 5 ;
        R6 := 6 ;
        R7 := 7 ;
        R8 := 8 ;
        R9 := 9 ;
        R10 := 10 ;
        R11 := 11 ;
        R12 := 12 ;
        R13 := 13 ;
        R14 := 14 ;
        R15 := 15 ;
        WHILE CC7 ;
        SWITCH CG0 ;
                CASE 0 , GOTO NEXT , AOR := R0  ;
                CASE 1 , GOTO NEXT , AOR := R1  ;
                CASE 2 , GOTO NEXT , AOR := R2  ;
                CASE 3 , GOTO NEXT , AOR := R3  ;
                CASE 4 , GOTO NEXT , AOR := R4  ;
                CASE 5 , GOTO NEXT , AOR := R5  ;
                CASE 6 , GOTO NEXT , AOR := R6  ;
                CASE 7 , GOTO NEXT , AOR := R7  ;
                CASE 8 , GOTO NEXT , AOR := R8  ;
                CASE 9 , GOTO NEXT , AOR := R9  ;
                CASE 10 , GOTO NEXT , AOR := R10 ;
                CASE 11 , GOTO NEXT , AOR := R11 ;
                CASE 12 , GOTO NEXT , AOR := R12 ;
                CASE 13 , GOTO NEXT , AOR := R13 ;
                CASE 14 , GOTO NEXT , AOR := R14  ;
                CASE 15 , GOTO NEXT , AOR := R15  ;
        ENDSWITCH ;
        NEXT : OUT 0 ;
        ENDWHILE ;
         OUT h'FFFF' ;
        LOOPX :
                GOTO LOOPX ;
        end ;
```

Simple DMA Controller for Memory to Memory Transfer

The software designs discussed so far have been based on arithmetic functions but an important feature of how to use the FIFO in the host interface section of the PAC1000 for the communication of data will enable the reader to develop ideas into more complex programs. The FIFO Output Ready flag is used in a loop to read the data into the registers. The output codes are used to create signals to control read, write, latch, output enable and bus acknowledge signals. A summary of these signals is given in Table 5 each time an instruction is executed. These signals are generated to accompany the memory addresses which control the DMA cycle.

Figure 6a shows a generic system solution where the PAC1000 sits on the address and data bus of a microprocessor and memory interface. The PAC1000 is mapped into the system with a PLD decoder and an external latch is used to catch data on read and write cycles. It is possible to use the internal DIR and DOR for this purpose but a faster solution would use an external component. Also, if the bus were greater than sixteen bits, an external latch would be required. This mode where data does not enter the PAC1000 device is called the 'fly by' mode.

Figure 6b shows the timing waveform derived from the program simulation. Active LOW WRB and CSB inputs to ADD1, ADD2 and ADD3 will write to the registers. The Source Address Register R0, the Destination Address Register R1 and the transfer counter R2 are all loaded through the FIFO. At time point 1, the registers become loaded. At time 2, CC7 is set HIGH to indicate transfer can commence. The response from the PAC1000 is an active LOW output from output control OC14 to inform the microprocessor that DMA activity is taking place. This occurs at time point 3. OC14 stays LOW during DMA activity but goes HIGH after the transfer is complete (at time point 4). Three transfers have taken place and the microprocessor is free to regain control of the bus.

```
segment pacdes06;

 /*THE PROGRAM ILLUSTRATES A SIMPLE DMA DESIGN WHICH  */
 /*READS THE DATA FROM SUCCESSIVE MEMORY LOCATIONS    */
 /*ADDRESSED BY THE CURRENT CONTENTS OF R0 THEN WRITES*/
 /*THAT DATA TO LOCATIONS ADDRESSED BY THE CONTENTS   */
 /*OF R1. BOTH REGISTERS ARE INCREMENTED AFTER THE    */
 /*READ/WRITE CYCLE. R2 IS A TRANSFER COUNTER THAT IS */
 /*DECREMENTED AFTER EACH TRANSFER. WHEN R2 IS ZERO   */
 /*ALL TRANSFER ACTIVITY CEASES AND A NEW DEVICE WAITS*/
 /*FOR A NEW DMA CYCLE.                               */

 /*             PIN FUNCTIONAL DESIGNATIONS.          */
 /*                    OUTPUTS.                       */

 /*     OC15 - LATCH ENABLE...........ACTIVE LOW.     */
 /*     OC14 - BUS TAKEN..............ACTIVE LOW.     */
 /*     OC13 - WRITE ENABLE...........ACTIVE LOW.     */
 /*     OC12 - READ ENABLE............ACTIVE LOW.     */
 /*     OC11 - LATCH OUTPUT ENABLE....ACTIVE LOW.     */
 /*     AOR  - 16 BIT ADDRESS OUTPUT..ACTIVE TRUE.    */

 /*                    INPUTS.                        */

 /*     CC7 - ACTIVE HIGH - INITIATE DMA ACTIVITY.   */
 /*     HD  - ACTIVE TRUE - 16 DATA INPUTS.          */
 /*     HAD - ACTIVE TRUE - REGISTER ADDRESS INPUTS  */
 /*     /CS - ACTIVE LOW  - PAC1000 SELECT           */
 /*     /WR - ACTIVE LOW  - WRITE TO PAC1000 FIFO    */
 /*     /RD - ACTIVE LOW  - READ NOT USED            */

 /*                LIST OF EQUATES.                  */

 READ     equ     H'AFFF'; /*ACTIVE LOW READ.TRANSFER  */
                           /*ENABLE.AND BUS BUSY       */
```

Figure 6a. PAC1000 as a Simple DMA Controller

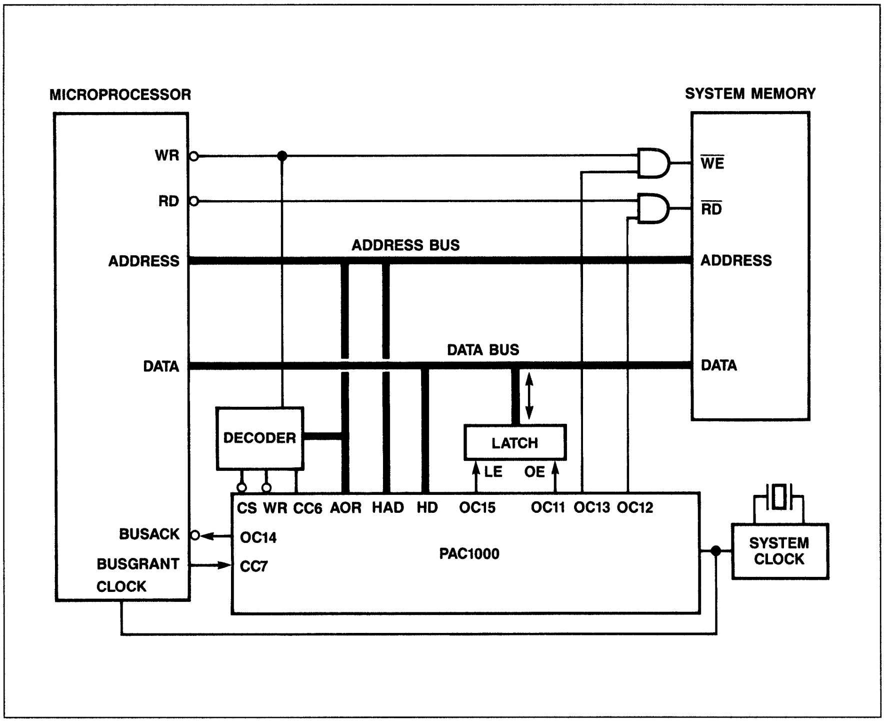


Figure 6b. System Waveforms

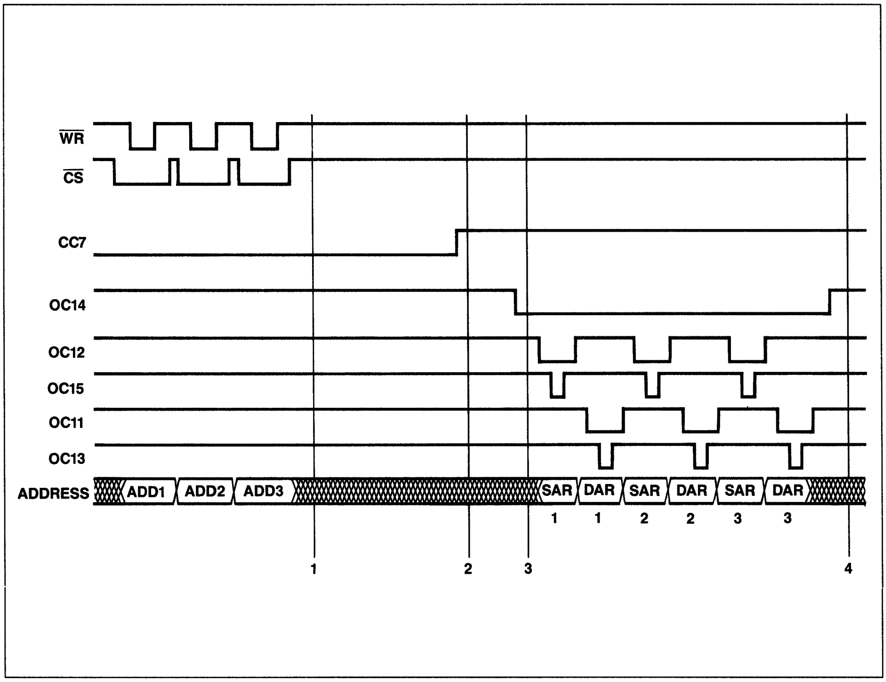


Table 5. Output Condition Assignment Codes for the DMA Controller Application

	OC15	OC14	OC13	OC12	OC11	OC10–OC0
INIT	1	1	1	1	1	All High
READ	1	0	1	0	1	All High
OENBL	1	0	1	1	0	All High
WRITE	1	0	1	0	0	All High
ENBLE	1	0	1	1	1	All High
LATCH	0	0	1	0	1	All High

OC15 = Active Low Latch Command
OC14 = Active Low DMA in Progress
OC13 = Active Low Write Signal
OC12 = Active Low Read Signal
OC11 = Active Low Output Enable

```
LATCH    equ      H'2FFF';      /*ACTIVE LOW READ.TRANSFER        */
                                /*ENABLE.LATCH ENABLE.AND         */
                                /* BUS BUSY                       */
OENBL    equ      H'B7FF':      /*ACTIVE LOW TRANSFER ENABLE      */
                                /*OUTPUT ENABLE.AND BUS           */
                                /* BUSY                           */
WRITE    equ      H'97FF';      /*ACTIVE LOW WRITE,TRANSFER       */
                                /*OUTPUT ENABLE.AND BUS           */
                                /* BUSY                           */
INIT     equ      H'FFFF':      /*INITALIZE ALL OUTPUTS HIGH      */
ENBLE    equ      H'BFFF';      /*ACTIVE LOW ENABLE TRANSFER      */
                                /*SIGNAL,AND BUS BUSY             */

/*                    PROGRAM START                               */

START:   OUT INIT;              /*INITALIZE OUTPUT CODES TO CCO-15*/
LOOP1:   RESET ADOE ;           /*SET THE ADDRESS BUFFERS INPUTS  */
         FOR 2 ;                /*SET READ FIFO LOOP TO 3         */
HOLDO:   JMPNC FIOR HOLDO ;     /*WAIT FOR ACTIVE FIOR FLAG       */
         RDFIFO ;               /*READ FIFO INTO THE REGISTER FILE*/
         ENDFOR ;               /*ALL THREE WORDS READ END LOOP   */

HOLD1:   JMPNC CC7 HOLD1 ;      /*ACTIVE CC7 BUSACK SIGNAL INPUT  */
         SET ADOE :             /*SET ADDRESS BUFFER AS OUTPUT    */
                                /*FOR DMA CYCLES                  */

         FOR , R2 := R2 , OUT ENBLE ;/*START DATA TRANSFERS       */
         AOR := RO ;                 /*OUTPUT SOURCE ADDRESS      */
         RO := ++ RO . OUT READ ;    /*OUTPUT ACTIVE READ         */
         OUT LATCH ;               /*AND LATCH DATA ON READ       */
         OUT READ :                /*HOLD READ LINE ACTIVE        */
         AOR := R1;                /*OUTPUT DESTINATION ADDRESS   */
         R1 := ++ Ri , OUT OENBL ; /*ENABLE LATCH OUTPUT          */
         OUT WRITE ;               /*PERFORM WRITE CYCLE          */
         OUT OENBL ;               /*DISABLE WRITE BEFORE OE      */
         OUT ENBLE ;               /*END OF SINGLE TRANSFER       */
         ENDFOR ;                  /*END OF TRANSFER CYCLE        */

HALT:    GOTO LOOP1 . OUT INIT ;   /*RETURN TO PROGRAM START      */
end;
```

FIFO DRAM Controller

The next PAC1000 design example illustrates how to use the device as a FIFO DRAM Controller. See Figure 7a for device implementation.

If the DRAMs are 64K devices, only the least significant byte of the AOR register need be used (that is ADD0–ADD7). The system could easily be upgraded to handle 256K or 1M bit DRAMs by wiring in address bits A8 and A9 but additional PAC1000 software would need to be written to accommodate the FIFO status counter. About 45 lines of code are used to enable the PAC1000 to handle REFRESH, READ and WRITE activity. The design uses the output control lines to provide RAS, CAS and WRITE signals to the DRAM and additional signals to give busy status during read, write and refresh activity. The whole system responds to input condition codes CC0 and CC1 as RQWRITE request to write and RQREAD request to read respectively. During active read, write and refresh cycles, three signals BUSYWR, BUSYRD and BUSYRFSH which go active LOW an additional composite signal which goes LOW when the FIFO is in any of these conditions. The system design also incorporates an UP/DOWN status counter which increments on write activity and decrements on read activity. This counter is tested to provide information to the outside world that the FIFO is full, empty or neither full or empty. The FULL, EMPTY and ACTIVE flags can be read from the IO0 and IO1 and give information to the outside world about the status of the FIFO.

The waveforms associated with read, write and refresh activity are shown in Figures 7b, 7c and 7d respectively. These waveforms were created from the PACDES08.OUT vector tables generated from the simulator. Table 6 illustrates the assignment of the output conditions which drive the various functions RAS, CAS, RFSH WR etc.,

It is recommended that high current buffer circuits be used to interface the outputs of the PAC1000 to the inputs of the memory chips used in both the DMA and FIFO applications.

```
 segment pacdes08 :

/*LIST OF EQUATES.*/
/*CONDITTION CODE OUTPUTS*/

RASW     equ      H'55FF' ;         /*WRITE RAS OUTPUT   */
RASR     equ      H'79FF' ;         /*READ RAS OUTPUT    */
RFSH     equ      H'7CFF' ;         /*REFRESH OUTPUT     */
CASW     equ      H'15FF' ;         /*WRITE CAS OUTPUT   */
CASR     equ      H'39FF' ;         /*READ CAS OUTPUT    */
ENDWR    equ      H'35FF' ;         /*END OF WRITE OUFUT*/
INIT     equ      H'FFFF' ;

ZERO     equ      H'0000' ;         /*ZERO COUNT*/
FULL     equ      H'FD'   ;         /*FULL FLAG */
EMPTY    equ      H'FE'   ;         /*EMPTY FLAG*/
ACTVE    equ      H'FF'   ;         /*ACTIVE    */
MAX      equ      H'FFFF' ;         /*MAX COUNT */

RQWRITE equ       CC0 ;             /*REQUEST TO WRITE*/
RQREAD   equ      CC1 ;             /*REQUEST TO READ */

                          /*PROGRAM START*/

START:   OUT INIT ;              /*INITALIZE OUTPUT CODES*/
                                 /*INITALIZE REGISTERS   */
R0  := H'0000'            ;      /*ROW ADDRESS WRITE     */
R1  := H'0000'            ;      /*COLUMN ADDRESS WRITE  */
R2  := H'0000'            ;      /*ROW ADDRESS READ      */
R3  := H'0000'            ;      /*COLUMN ADDRESS READ   */
```

FIFO DRAM Controller (Cont.)

```
                R4 := H'FFFF'        ;    /*REFRESH COUNTER      */
                R5 := H'0000'        :    /*STATUS COUNTER       */

                OUTPUT IO0 IO1 ;          /*SER IO0 AND IO1 TO   */
                SET ADOE , OUT INIT ;     /*OUTPUT, ADOE INPUT   */
                IOR := EMPTY  ;           /*FIFO IS EMPTY        */
                GOTO TEST ;               /*TEST REQUEST TO      */
                                          /*READ/WRITE           */

LOOP:           AOR := R4 ;               /*OUTPUT REFRESH CTR   */
                OUT RFSH ;                /*PERFORM REFRESH      */
                R4 := ++ R4 . OUT INIT :  /*INCREMENT RFSH CTR   */
                                          /*CLEAR OUTPUT         */

TEST:           IF RQWRITE:               /*IF REQUEST TO WRITE */
                 AOR := RO , OUT INIT ;   /*OUTPUT WRITE ADDR    */
                 R5 := ++ R5 ;            /*INCREMENT STATUS     */
                 OUT RASW ;               /*OUTPUT RAS WRITE     */
                 AOR := R1 ;              /*OUTPUT CAS ADDR      */
                 R1 := ++ R1 ;            /*INCREMENT CAS ADDR   */
                 OUT CASW ;               /*OUTPUT CAS ADDR      */
                 OUT ENDWR ;              /*END WRITE CYCLE      */
                 OUT INIT ;               /*FINISH WRITE CYCLE   */
                ENDIF ;

                 IF R1 == 256 ;           /*TEST FOR 256 COLUMNS*/
                  RO := ++ RO ;           /*INCREMENT ROW        */
                 ENDIF ;                  /*IF 256               */

                 IF RQREAD:               /*IF REQUEST TO READ   */
                 AOR := R2 . OUT INIT :   /*OUT ROW READ ADDRESS*/
                 R5 := -- R5 ;            /*DECREMENT STATUS     */
                 OUT RASR ;               /*OUTPUT RAS READ      */
                 AOR := R3 ;              /*OUTPUT CAS ADDRESS   */
                 R3 := ++ R3 . OUT CASR ;/*INCREMENT CAS ADD    */
                 OUT CASR ;               /*STRETCH CAS          */
                 OUT INIT ;               /*FINISH READ CYCLE    */
                ENDIF ;

                IF R3 == 256 :            /*TEST FOR 256 COLUMNS*/
                 R2 := ++ R2 ;            /*INCREMENT ROW        */
                ENDIF ;                   /*IF EQUAL TO 256      */

                R6 := R5 :                /*SAVE STATUS COUNTER */
                R6 := MAX - R5 ;          /*TEST FOR MAX COUNT   */
                IF Z ;                    /*IF MAXIMUM           */
                IOR := FULL ;             /*SET OUTPUT FULL FLAG*/
                GOTO LOOP ;               /*GOTO REFRESH LOOP    */
                ENDIF ;                   /*END TEST             */

                R6 := R5 ;                /*SAVE STATUS COUNTER */
                R6 := ZERO - R6 ;         /*TEST FOR ZERO COUNT */
                IF Z ;                    /*IF ZERO              */
                IOR := EMPTY ;            /*SET EMPRY FLAG       */
                GOTO START;               /*RESTART PROGRAM      */
                ENDIF ;                   /*ELSE                 */

                IOR := ACTVE ;            /*THE SYSTEM IS NOT    */
                GOTO LOOP ;               /*FULL OR EMPTY        */
end;
```

Table 6. Output Condition Assignment Codes for the PAC FIFO DRAM Controller Design

	OC15	OC14	OC13	OC11	OC10	OC9	OC8	OC12, OC7–OC0
INIT	1	1	1	1	1	1	1	All High
RASW	0	1	0	1	0	1	0	All High
CASW	0	0	0	0	1	0	1	All High
ENDW	0	0	1	0	1	0	1	All High
RASR	0	1	1	1	0	0	1	All High
CASR	0	0	1	1	0	0	1	All High
RFSR	0	1	1	1	1	0	0	All High

OC15 = Active Low RAS
OC14 = Active Low CAS
OC13 = Active Low Write
OC11 = Active Low BUSYWR
OC10 = Active Low BUSYRD
OC9 = Active Low Busy
OC8 = Active Low BUSYRFSH

Figure 7a. Using a PAC as a FIFO DRAM Controller

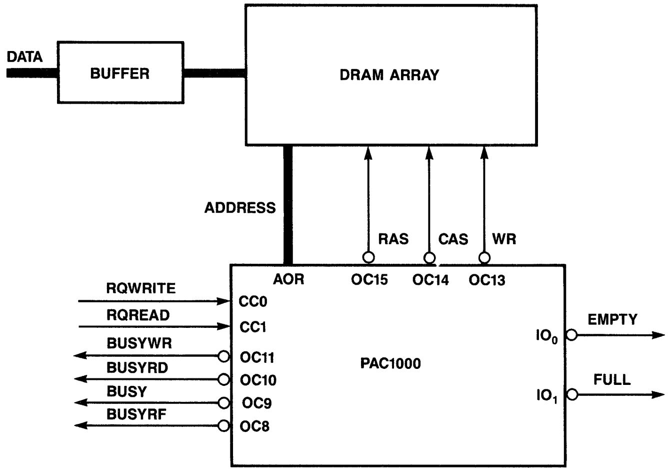
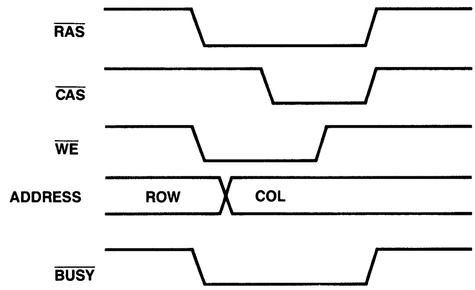
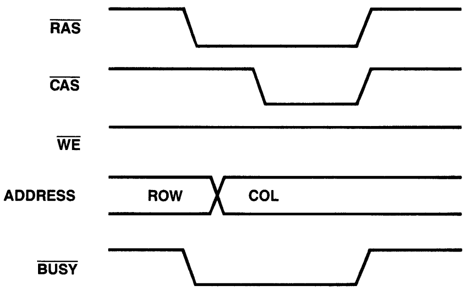


Figure 7b.



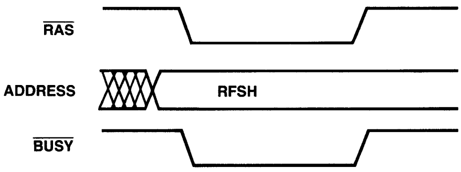
In response to a request to read one early write cycle will take place. $\overline{\text{RAS}}$ will latch in the row address and the WE line goes low. The column address is set up followed by the falling edge of $\overline{\text{CAS}}$. The WE input is taken inactive followed by $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$. During the whole cycle the busy signal is active.

Figure 7c.



In response to a request to read one read cycle will take place. The $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ signals latch in the row and column addresses respectively but the WE input is inactive throughout the cycle. The $\overline{\text{BUSY}}$ signal is active throughout the whole cycle.

Figure 7d.



To refresh the memory the PAC will output a refresh count to be strobed into the DRAMs by an active low $\overline{\text{RAS}}$ transition.

Programmable UART

The PAC1000 contains no UART for serial data but parallel to serial conversion is possible through the Q register and I/O Port 2 and 3. The following program illustrates the designer how to create a UART function in the PAC1000 with about 40 lines of instructions. The PAC1000 device will receive data in parallel from the host system. The FIFO is used to interface to the host and transfer data into the registers. The program will take the seven bits of ASCII code and calculate the parity, then add a parity bit. The result is serialized and framing bits are applied. The data, one parity bit, one start bit and two stop bits are serially clocked out of the Q register into Port 3. The handshake signals of Data Terminal Ready and Data Set Ready are built into the program.

```
segment pacdes09 ;

/*THIS PROGRAM ILLUSTRATES THE PARALLEL TO SERIAL       */
/*CHANNEL CONVERSION OF THE PAC1000 TO THE PERIPHERAL   */
/*BUS OF THE SYSTEM                                     */
/*                                                      */
/*              PIN FUNCTIONAL DESIGNATIONS.            */
/*                      OUTPUTS.                        */
/*                                                      */
/*      OC12 - DTR - DATA TERMINAL READY....ACTIVE LOW. */
/*      OC13 - RHD - RECEIVED HOST DATA.....ACTIVE LOW. */
/*      OC14 - DONE.........................ACTIVE LOW. */
/*      OC15 - ABORT........................ACTIVE LOW. */
/*      IO3 - TxD - TRANSMITTED DATA........ACTIVE LOW. */
/*                                                      */
/*                      INPUTS.                         */
/*                                                      */
/*      CC0 - DSR - DATA SET READY.........ACTIVE HIGH. */
/*      CC1 - START TRANSMITTING...........ACTIVE HIGH. */
/*      HD  - ACTIVE TRUE - 16 DATA INPUTS.             */
/*      HAD - ACTIVE TRUE - REGISTER ADDRESS INPUTS     */
/*      /CS - ACTIVE LOW  - PAC1000 SELECT              */
/*      /WR - ACTIVE LOW  - WRITE TO PAC1000 FIFO       */
/*                                                      */

INIT   equ  H'FFFF'; /*INITALIZE ALL OUTPUTS HIGH       */
RHD    equ  H'DFFF'; /*ACKNOWLEDGE RECEIVING HOST DATA  */
DTR    equ  H'EFFF'; /*DATA TERMINAL READY              */
DONE   equ  H'BFFF';
ABORT  equ  H'8FFF'; /*TELL HOST THAT DATA WAS CORRUPTED*/


/* R21 - H'0060' - MASK REGISTER FOR EVEN PARITY        */
/* R20 - H'00E0' - MASK REGISTER FOR ODD PARITY         */
/* R19 - H'0002' - CONSTANT TO DIVIDE THE 32-BIT VALUE  */
/*                 IN RX R16                            */
/* R18 - H'0000' - COUNTER OF THE NUMBER OF ONES IN THE */
/*                 DATA                                 */
/* R17 - H'FFFF' - A CONSTANT TO MASK WITH DATA         */
/* R16 - H'0000' - A CONSTANT TO MASK WITH DATA         */
/* R8  -           WORKING REGISTER FROM R0             */
/* R0  -           ORIGINAL DATA FROM HOST SYSTEM       */
/* Q   -           REGISTER TO SHIFT OUT DATA TO THE    */
/*                 SERIAL PORT                          */
```

Programmable UART (Cont.)

```
begn:    R21 := H'0060' , OUT INIT ;   /*SET OC[15:0] HIGH*/
                 R20 := H'00E0' ;
                 R19 := H'0002' ;
                 R18 := H'0000' ;
                 R17 := H'FFFF' ;
                 R16 := R18 ;
                 Q := R18 ; /*  INITIALIZE Q TO ZERO'S    */


/*               WAIT FOR HOST TELLS PAC1000              */
/*               TO START TRANSMITTING DATA               */

stndby:  JMPNC CC1 stndby ;
         JMPC FICD abort ;
         RDFIFO , OUT RHD ; /* READ FIFO DATA INTO RO     */
                            /*TELL HOST THAT DATA WAS     */
                            /*READ CORRECTLY              */

/*********************************************************/
/*               FORMAT OF DATA RECEIVED                  */
/*       FIFODA[15:0]                                     */
/*       15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0  */
/*       0  0  0  0  0  0  0  0  0 D1 D2 D3 D4 D5 D6 D7  */
/*********************************************************/

/*       SWAP THE HIGHER AND LOWER BYTES                  */

                                /* SET OC TO NON-         */
         AOR := RO , OUT INIT ; /* FUNCTIONING MODE       */
         R8 := SWPV ;           /* MODE SWAP TO SHIFT     */
         RO := SWPV ;           /* LATER SWAP NOW         */

/*       SHIFT DATA                                       */

                 FOR 7 ;
                   R8:= R8 << 0 ;
                   IF S ;
                     R18 := ++ R18 ;   /*INCREMENT COUNTER*/
                   ENDIF ;
                 ENDFOR ;

/*        CHECK FOR EVEN/ODD PARITY                       */

         DIV R16 R18 R19 ; /* DIVIDE R18 R16 BY 2         */
         OR Q 0 ;          /* CHECK IF REMAINDER IS ZERO */
         IF Z ;            /* IF Z=1 THEN JUMP TO PARITY */
                           /* (EVEN PARITY)               */
                           /* IF Z = 0 THEN (ODD PARITY) */
         OR RO R21 ;   /* MERGE MASK BITS FOR EVEN PARITY */
         ELSE ;
         OR RO R20 ; /*  MERGE MASK BITS FOR ODD PARITY  */
         ENDIF ;     /* RO IS NOW FORMATTED CORRECTLY FOR*/
                     /* SERIAL SHIFTING                   */
         Q := RO ;   /* LOAD RO TO Q TO SHIFT OUT TO IO3 */

/*       CHECK THAT RECEIVING END IS READY                */
```

Programmable UART (Cont.)

```
wait:   IF CCO ;     /*IF RECEIVER READY SET IO3 TO OUTPUT */
        CONFIGURE SIO ; /*AND SET MODE TO SHIFT Q TO IO3 */
        OUTPUT IO3 , OUT DTR ; /*DRIVE DTR TO ZERO THIS  */
                                  /*TELLS THE RECEIVER THAT */
                                  /*THE TRANSMITTER IS READY*/


/* SHIFT OUT THE 1 START BIT,7-BITS OF DATA,1 PARITY AND */
/* 2 STOP BITS . THEREFORE SHIFT 11 TIMES */

        LDLC 10 ;        /*LOAD 10 INTO LOOP COUNTER FOR */
                         /*A SHIFT OF 11 THEN FILL WITH  */
                         /*ZEROS                         */
lp:     LOOPNZ lp , QR8 := Q << 0 R8 << 0 ;
        ELSE:
        JMP wait ;    /*IF RECEIVER IS NOT READY THEN WAIT*/
        ENDIF ;
        OUT DONE ;    /*TELL HOST THAT PAC1000 IS DONE    */
        JMP begn ;    /* START AGAIN FOR NEXT DATA        */

/*      ABORT DATA READ AND TELL HOST ABOUT IT            */

abort:  JMP begn , OUT ABORT ;

end;
```

Summary

The PAC1000 user programmable high performance microcontroller incorporates many features that enable a high speed design to be quickly realized. Its re-programmability has enabled many designers to go to printed circuit board layout early in the design cycle. Moreover, because the system logic is programmable into the on-chip EPROM, modifications can be made at a later time without having to change printed circuit board artwork. In fact over discrete and PAL/EPLD type solutions the printed circuit board artwork is considerably less complex because a greater degree of circuit complexity containing much interconnect has migrated into the instructions encoded in the EPROM section of the chip.

To learn how to use the PAC1000 is a relatively quick process for most systems designers have designed with microprocessors and microcontrollers. This is because they understand the writing of assembly or high level code. With the support of WSI's user friendly software tools, an engineer can be designing with the PAC1000 in less than a week. This contrasts with the many and diverse schematic capture, net translation, placing and routing, annotation and back-annotation packages that support EPLD and PGA devices. These products subject the designer to a multiplicity of software tools that he must become familiar with. This results in generating a long learning curve that can easily be avoided with the PAC1000 and WSI's software support.

The result of using the PAC1000 device and software tools virtually guarantees the fastest route possible from initial conception to the final design of a complex high performance system.

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Note 005

PAC1000 as a High-Speed Four-Channel DMA Controller

By Arye Ziklik and Kiran Buch

Abstract

The objective of this Application Note is to demonstrate the use of the PAC1000 User-configurable Microcontroller in a typical high performance application. The text describes an implementation of a generic four-channel DMA controller that supports transfer rates of up to *20 Mbyte/sec* (10 Mword/sec) in 16-bit data-bus environments.

This Application Note covers the terminology of DMA operations as well as an implementation description. The readers will be able to use this article as a get-started tutorial that shows how to configure the PAC1000 for any specific task.

Introduction

A DMA (Direct Memory Access) controller coordinates fast data transfers between peripheral devices and the system memory. All possible transfer combinations might occur: device to device, device to memory or memory to memory. By taking care of these high-speed transfers, the host computer (typically a Microprocessor) is off-loaded from these time-consuming tasks and can execute other operations concurrently, on its local bus.

We refer to peripherals such as FIFOs, video, communication, graphics or serial channel controllers, latches, ports, etc., as devices in this text. The distinction between memory and device is that a memory needs an explicit address in order to specify a byte or a word, whereas a device requires only strobes (such as: $\overline{RD}$, $\overline{WR}$, $\overline{CS}$) combined sometimes with additional hand-shaking signals for data accessing.

The PAC1000 is a perfect match for most DMA applications. Its unique structure, shown in Figure 1 and Figure 2, allows the user to execute three independent instructions in one cycle. The ability of the PAC1000 to perform three different tasks concurrently (Control, Output and CPU) is fully exploited here, thereby speeding-up DMA transfers.

For example, during DMA operations, the control section checks for the block-count termination, the output control section generates $\overline{RD}$ and $\overline{WR}$ strobes, and the CPU calculates and produces the next address. All these activities occur simultaneously during the same clock cycle(!).

Unlike most other available DMA controllers, the PAC1000 is a user-programmable Microcontroller. It may be easily modified by reprogramming to support various DMA schemes.

Figure 3 illustrates a typical system configuration using the PAC1000 as a DMA controller. The host controls the system bus as well as its local bus (not shown here). It can also access the memories, the devices as well as the PAC1000 via the system bus. It does so by driving the Address, Control and Data buses.

Initially the PAC1000 is in the slave mode, waiting for host messages. Once the host begins a channel initialization phase by writing into the PAC1000's FIFO, a DMA operation will start. In that phase, the host instructs the PAC1000 of the required DMA transfer. The PAC1000 then decodes the transfer type and optimizes it internally to perform at the fastest rate the surrounding hardware allows. At this point the PAC1000 requests the system bus from the bus arbiter. When the bus is granted to the PAC1000, it becomes the Bus Master, driving the address, data and control buses.

If the DMA operation is fully completed, or a higher priority transfer is pending, or the host or active devices abort the transfer, a DMA transfer can be successfully terminated or suspended, respectively. In all of these cases, system control is returned to the host and the PAC1000 re-enters to Slave Mode.

Figure 1. PAC1000 Microcontroller Block Diagram

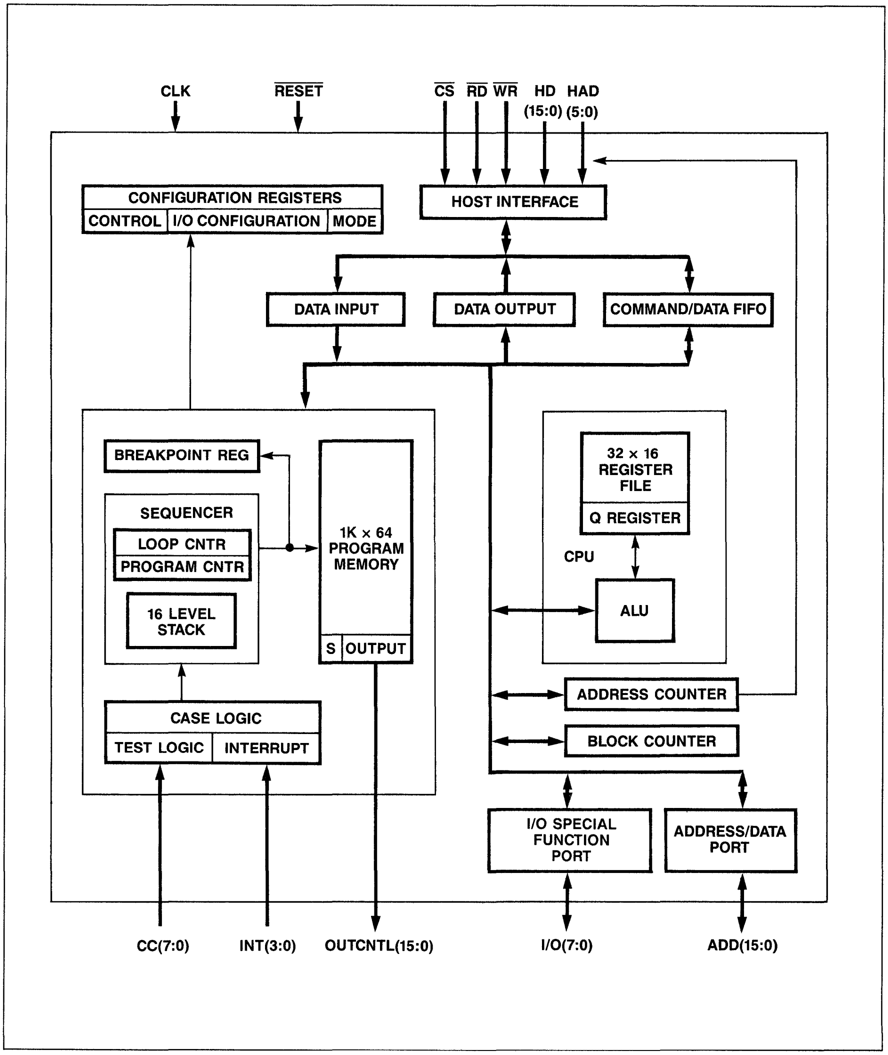
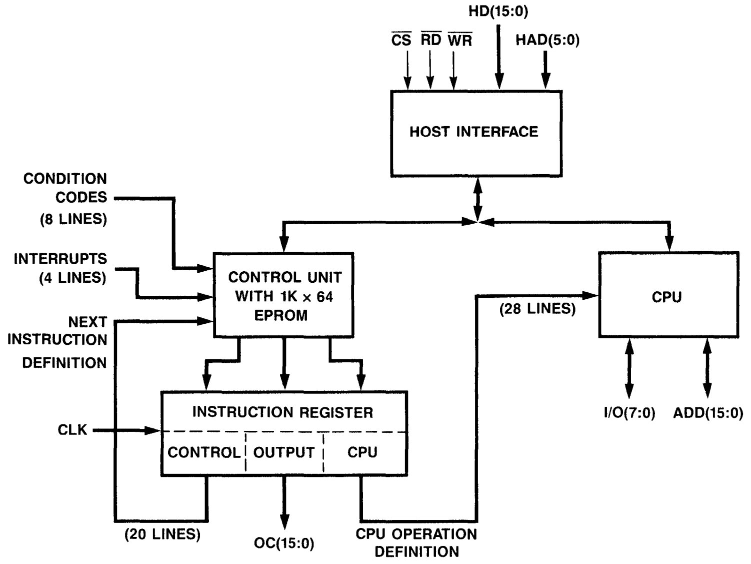


Figure 2. Single Cycle Control Architecture



Important Features:

- One cycle per instruction.
- 20 MHz instruction execution rate.
- Every instruction executes 3 parallel operations (Control, Output, CPU).

Transfer Modes

There are two transfer modes: Fly-by and Dual cycle.

Fly-by is the fastest transfer mode (refer to Figure 4). Transfers can be carried out at a rate of up to 10 Mword/sec provided that the PAC1000 uses a 20-MHz clock. In this application note, Fly-by can only be used between memory and device if they share the same data-bus path (either 8 or 16 bits).

The fly-by operation is initiated by a DMARQ from the device. The PAC1000 explicitly addresses the memory, while sending the $\overline{RD}$ strobe to the source side and the $\overline{WR}$ strobe to the destination side. It also acknowledges the device with the $\overline{DMACK}$ signal that serves as the device's $\overline{CS}$ signal. Data is then directly transferred from the source to the destination in one bus cycle.

Double-cycle is a transfer mode comprised of two bus cycles. It takes place whenever one of the following DMA combinations is specified (refer to Figure 5):

□ Memory to/from device that is not connected to the same part of the data-bus.

□ Memory to Memory transfers (require the generation of two different explicit addresses).

□ Device to Device transfers (with simple additional hardware it might be easily upgraded to support the Fly-by mode, too).

Once the transfer has started, the PAC1000 reads an operand from the source on the first bus-cycle, processes it, and then writes that operand on the second bus cycle into the destination.

The READY signal enables the PAC1000 to synchronize its operations with slow memories or devices (whenever they are explicitly addressed). READY is an active-high signal, derived from the address decoder. It is driven low as long as the addressed memory or device is not ready to finish the current bus-cycle.

Figure 3. System Block Diagram

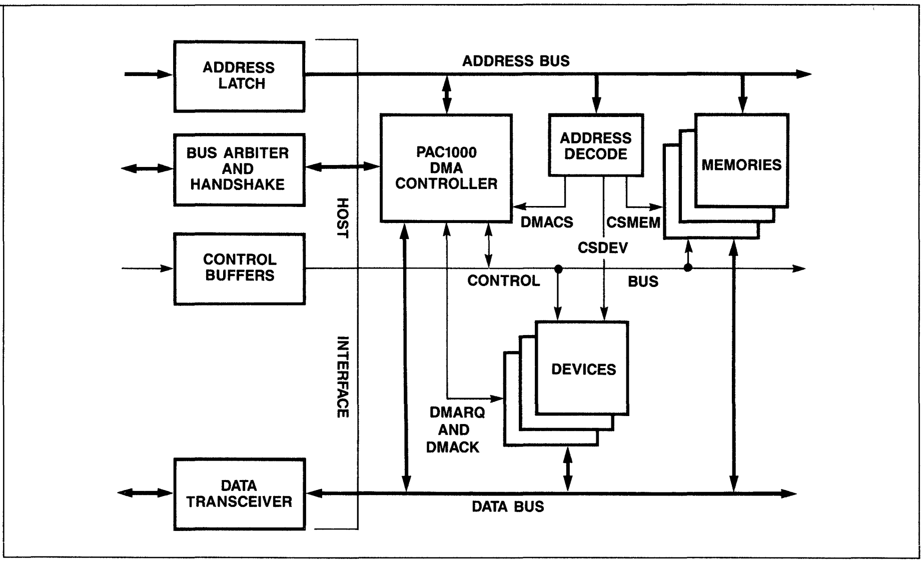


Figure 4. Fly-by DMA Transfer

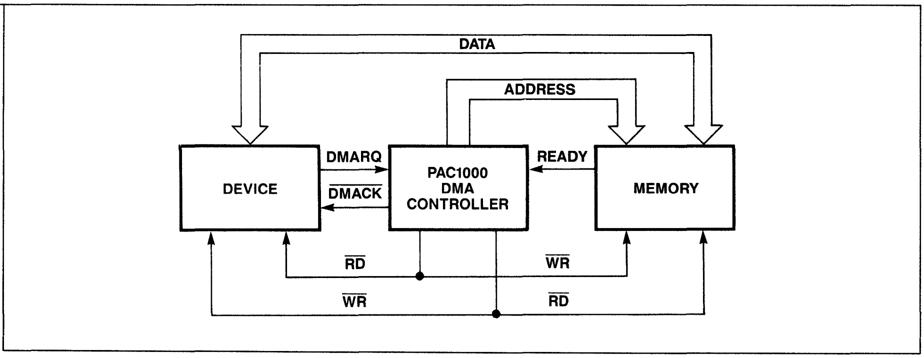


Figure 5. Double Cycle DMA Transfer — Memory to Device

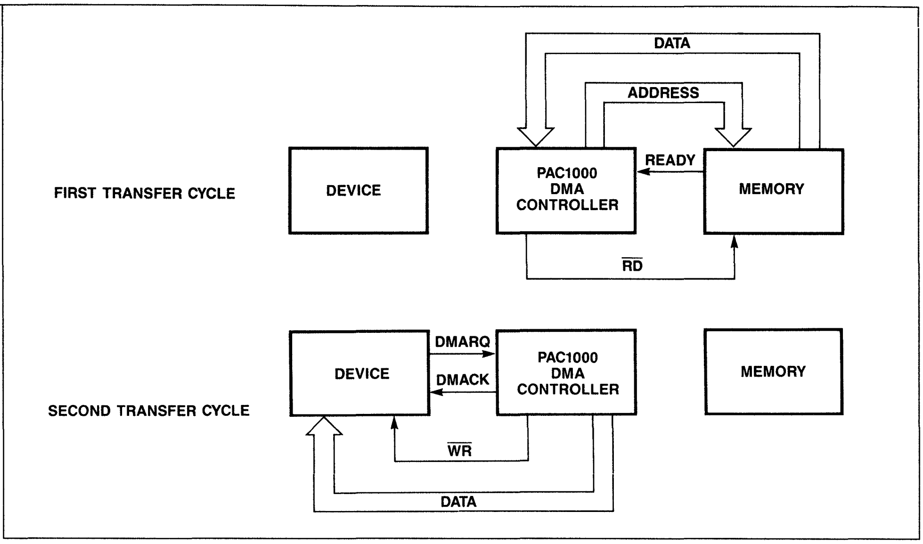


Figure 5. (Cont.) Double Cycle DMA Transfer — Memory to Memory

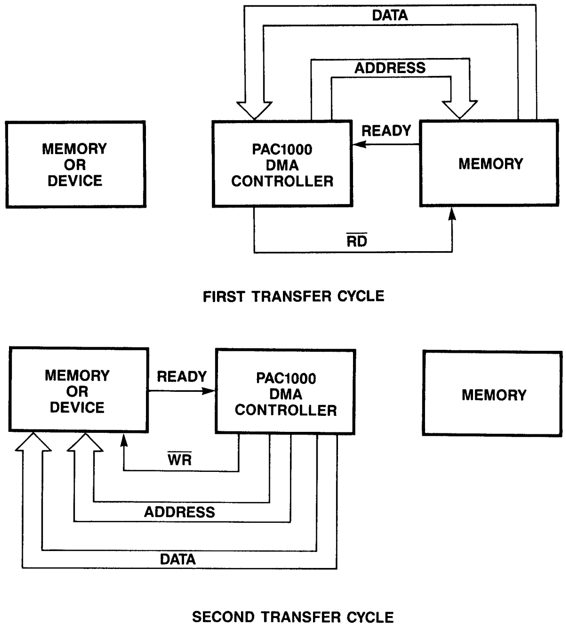
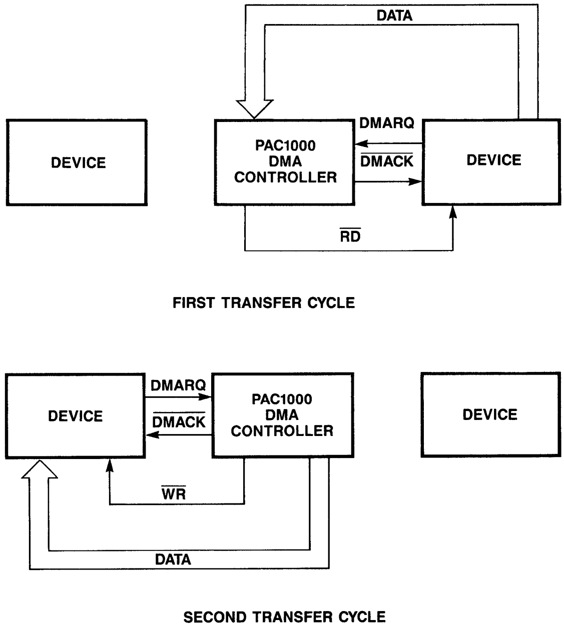


Figure 5. (Cont.) Double Cycle DMA Transfer — Device to Device



Request Modes

Requests may be externally generated by a device or internally created by the auto-request mechanism of the PAC1000, whenever a memory to memory transfer is performed. Auto-requests are always pending so that the PAC1000 can work at its maximum speed, provided that the memories are always ready. Otherwise, the PAC1000 adapts itself to the READY signal.

External requests may be of either the block-type or of the single-operand transfer mode. Block-type transfers are provided for high-speed devices that are capable of meeting the speed rate of the PAC1000. DMARQ is asserted at the beginning of the block transfer and remains so as long as the transfer is in process. Single-operand transfers are used by slow devices. They toggle on and off the DMARQ. Each individual transfer is indicated by an active high DMARQ level. When the transfer is completed, DMARQ is held low until the device is ready for the next transfer cycle, and so on.

Some important observations:

❑ Memory to device (or device to memory) transfers will begin only after an external DMARQ is asserted by the device.

❑ Synchronization with the memory is always achieved via the Ready signal.

Table 1 briefly summarizes the transfer and request options:

Table 1. Summary of Transfer and Request Modes

Transfer Type	DMA Mode	Transfer Mode
Memory to Memory	Two Bus-cycles	Block
Memory to Device or Device to Memory	Fly-by or Two Bus-Cycles	Block or Single Operand
Device to Device	Two Bus-Cycles	Block or Single Operand

Functional Description

General:

Figure 6 contains the circuit diagram. Refer also to Appendix 1 for the Pin Description Table. The PAC1000 is configured in this application as a four-channel DMA controller. This means that it can handle up to four DMA transfers concurrently, on a prioritized basis. Each of the channels can be any one of the above-mentioned DMA transfer types. The maximum transfer rate is accomplished during Fly-by transfers with rates approaching 10 Mword/sec for word transfers or 10 Mbyte/sec for byte transfers. Double-cycle transfer modes achieve a rate of up to 5 Mword/sec (in word transfers) or 5 Mbyte/sec (in byte transfers). The only exception to this is the Memory to Memory transfer mode which is a little bit slower due to the internal creation of two different 24-bit addresses.

The PAC1000 drives 24 address lines and handles a 16-bit data bus, so it is well tuned for most common high-performance buses or Microprocessors. The maximum operand block-size is 64K (in accordance with VMEbus specs, for example).

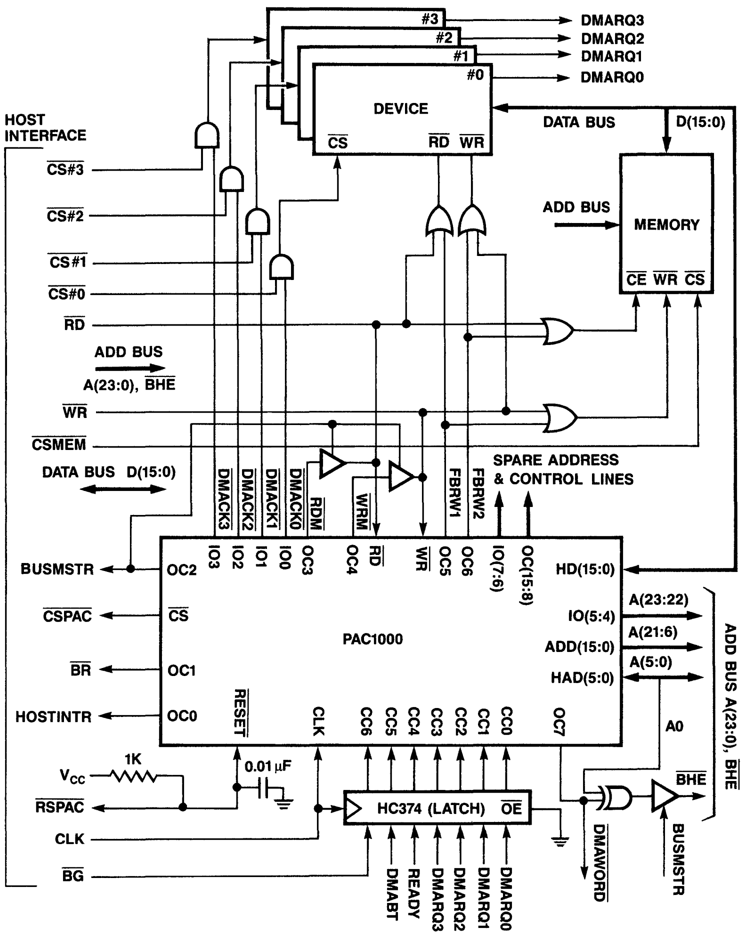
Host-PAC1000 Communication:

DMA specifications are programmed into the PAC1000 by the host, according to the message format of Appendix 2. The host writes eight words into the PAC1000's FIFO. The command message fully specifies one of the four possible channels that can be active at the same time. Word 1 defines the transfer characteristics of the DMA operation: transfer type, data bus width, device numbers (redundant in Memory to Memory operations), channel-priority and transfer mode. Bit 12 in that word serves as a software abort-command bit. When set, it instructs the DMA controller to cease the transfers of the channel specified in that command buffer.

The low-order byte of word 7 is a DMA-transfer identification number. It assigns a serial number to a DMA process. Whenever the PAC1000 sends a status message to the host, that number is also included in order to unambiguously identify the process that has either normally terminated or abnormally aborted (by an external device or due to a PAC1000 exception).

Functional Description (Cont.)

Figure 6. PAC1000 Configured as a Generic High-Speed DMA Controller



Functional Description (Cont.)

Several fields in the command buffer are optional. For instance, in transfers where devices are involved, one can still specify the explicit addresses of the source and the destination even though it has already been defined by the command word's device-number field (Appendix 2 — command word format). This feature allows the programmer to define the device interface with either explicit or implicit address.

Whenever the PAC1000 has to inform the host of an important event, it prepares a status word in its DOR (Data Output Register), enters the slave mode and interrupts the host by raising the HOSTINTR line. The possible messages are:

- □ Reject the Command buffer with the specified identification number because of internal discrepancies or illegal combinations.
- □ Propagate a Hardware DMA abort, generated by the source or the destination of the current transfer.
- □ Signal a PAC1000 exception. The host is capable of reading the PAC1000's SR register in order to find out the cause.
- □ An end-of-count message. This transfer has been normally terminated.

Initial State and Slave Mode:
After a reset (either a power-on reset or a reset through the $\overline{\mathrm{RSPAC}}$ line driven from the host side), the PAC1000 enters its initial state, which is the Slave Mode. Table 2 describes the signal states during the Slave Mode. The PAC1000 monitors its internal FIIR flag (FIFO Input Ready) and when it is not set, the FIFO is full with a new command buffer written by the host. The PAC1000 decodes the message and acts accordingly. If it is a memory to memory transfer, then it immediately requests the bus. When one or two devices participate in a transfer operation, the PAC1000 monitors the corresponding DMARQ lines to determine when to issue a bus request to the arbiter. The PAC1000 requests the bus by lowering $\overline{\mathrm{BR}}$. Then it waits for $\overline{\mathrm{BG}}$ to go low in order to switch to the Master Mode.

Master Mode:
Upon gaining mastership, the PAC1000 drives the HOSTINTR signal low and BUSMSTR high. BUSMSTR remains high (active) as long as the PAC1000 remains master of the system bus, thereby enabling $\overline{\mathrm{RDM}}$ and $\overline{\mathrm{WRM}}$ to $\overline{\mathrm{RD}}$ and $\overline{\mathrm{WR}}$, respectively. $\overline{\mathrm{BR}}$ is set high (= not active). According to the required DMA operation, the PAC1000 drives the appropriate address and data lines, and the $\overline{\mathrm{RDM}}$, $\overline{\mathrm{WRM}}$ and $\overline{\mathrm{DMACK}}$ signals.

DMA transfers may be successfully ended (when the terminal-count expires) or aborted. Abortion can emanate either from an external DMABT signal that is driven by one of the DMA participants, or from an internal exception recognized by the PAC1000. Whenever one of the above events occur, the PAC1000 changes its mode to the Slave mode, writes a status word into the DOR register (discussed previously) and raises the HOSTINTR line to cause the host to read that information through its own Interrupt routine.

Releasing and resuming bus control:
The host is allocated a higher priority than the PAC1000 by the bus arbiter. This is done in order to enable the host to suspend DMA transfers whenever it needs the bus. Each time the host accesses an address that resides within the system bus domain (including the $\overline{\mathrm{CSPAC}}$ address), the bus will be granted. If the PAC1000 is the current master (as reflected by BUSMSTR), the bus arbiter will negate $\overline{\mathrm{BG}}$ (high level). The PAC1000 monitors this line while it is a bus Master and consequently will relinquish the bus and return to the slave mode. The host might use the bus for programming the PAC1000 with a new DMA channel. Upon completion of the host activities over the system bus ($\overline{\mathrm{BG}}$ becomes high), the PAC1000 checks whether DMA transfers are still pending. If this is the case, it will request the bus. When the bus is granted, it will determine whether to continue the suspended transfer or to start a higher priority pending-DMA request. If it starts a higher priority transfer, then the suspended operation will be resumed after the completion of the higher priority transfer.

$\overline{\mathrm{DMAWORD}}$ is set low during word transfers and high during byte transfers. It is used to derive the $\overline{\mathrm{BHE}}$ strobe, as displayed in Figure 6. The most efficient transfer method is the word transfer mode. In order to use it, the specified addresses must be even, otherwise the PAC1000 will perform only in the byte transfer mode regardless of the command word content.

Functional Description (Cont.)

Table 2. Signal States During the Slave Mode

PAC1000 Signal Names	Function	Signal States
ADD(15:0)	A(21:6)	Float
HAD(5:0)	A(5:0)	Input
IO(5:4)	A(23:22)	Float
OC6, OC5	FBRW2, FBRW1	0, 0 — Normal Operation
OC4, OC3	$\overline{\text{RDM}}$, $\overline{\text{WRM}}$	Don't Care
10(3:0)	$\overline{\text{DMACK}}$ (3:0)	1,1,1,1 — Normal Operation
OC2	BUSMSTR	0 — Non-active
OC1	$\overline{\text{BR}}$	1 — Non-active
OC0	HOSTINTR	0 — Non-active
OC7	$\overline{\text{DMAWORD}}$	Don't Care
HD(15:0)	D(15:0)	Input

Hardware Considerations

Figure 6 is the detailed schematic diagram. The host side is beyond the scope of this paper since it is application dependent. In addition to the PAC1000, there are a few standard glue-logic chips used to interface with the memory and the four devices.

Throughout the following description it is assumed that the glue-logic components belong to the HC family. However, since the PAC1000 is a fully TTL compatible device implemented in CMOS technology, the reader can use other glue-logic families like: LSTTL, HCT, etc.

The HC374 latch is gated into the condition code inputs by the PAC1000's clock, thus ensuring that the CC7–CC0 lines will meet the set-up time requirements.

The three-state buffers controlled by BUSMSTR, are part of a HC126 chip. They are used to float the PAC1000's $\overline{\text{BHE}}$, $\overline{\text{RDM}}$, $\overline{\text{WRM}}$ control lines during slave operations, because at that time these signals are driven by the host.

The four AND-Gates amount to one HC08 chip. They enable either the host side (during Slave operations) or the PAC1000 (in the Master Mode) to drive the appropriate device $\overline{\text{CS}}$ signals.

The four OR-Gates comprise together one HC32 chip. They are used during Fly-by operations to avoid $\overline{\text{CE}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ from reaching the selected devices and memories concurrently (for functional explanation, refer to the Pin Description Table, Appendix 1).

Prior to the setting of $\overline{\text{BG}}$ in the active position (low), the arbiter floats the data bus D(15:0), address bus A(23:0) and $\overline{\text{BHE}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ from the host side. As long as BUSMSTR remains high, these lines are driven by the PAC1000.

The six chip select lines from the host side ($\overline{\text{CS\#3}}$ — $\overline{\text{CS\#0}}$, $\overline{\text{CSMEM}}$ and $\overline{\text{CSPAC}}$) are derived from the system address decode block, as illustrated in Figure 6. During the time that the PAC1000 is the bus master, the address decode block (shown in Figure 3) is driven by the PAC1000's address lines. Therefore, the PAC1000 can access memories and devices in the same manner the host does.

The $\overline{\text{DMACK3}}$–$\overline{\text{DMACK0}}$ signals provide the PAC1000 with an alternative chip select generation method to the devices. It is considerably faster than the host's method, since there is no need to generate explicit device addresses inside the PAC1000.

Hardware Considerations (Cont.)

In this application note, it is assumed that the READY signal is produced by the address decoder. However, if a device or memory can generate the READY signal independent of the decoder, the system designer can connect it with a Three-state buffer so that it will drive the READY input whenever it is chip-selected.

The host programmer is free to choose whether to synchronize the PAC1000 with slow devices via single operand transfers or through the READY mechanism. READY is always considered when the PAC1000 generates an explicit address. The selection between single operand transfer and READY is done in the command word (see Appendix 2).

As seen in Figure 6 there are several spare pins, such as output controls, I/Os, interrupts and condition codes. These pins can be used to perform other operations in parallel (unrelated to the DMA controller function), without any performance degradation of the DMA task.

PAC1000 Internal Resources Usage

Using PAC1000 as a 4-channel DMA controller utilizes most of the resources available on the chip, shown in Figure 1.

The Host microprocessor uses the FIFO to program the DMA request in to PAC1000. Internal condition codes are used to monitor FIFO status, CPU operation flags and external condition code inputs are used to monitor situations like bus-grant, DMA requests by the devices, etc. The CPU registers are used to store source and destination addresses, device numbers and other relevant information about the DMA transfers in progress.

To achieve the fastest transfer rate possible with PAC1000, address generation and block size counting are achieved by different methods depending on the type of transfer. For example, for the Device-Memory fly-by transfers, a nested loop is set up using the loop counter and the stack for maintaining block count and ACH and ACL are used as independent registers for address generation. On the other hand, for the memory to memory transfers, Block counter is used for counting and address generation is done by using ACH and ACL as 22-bit counter.

The IOR is used to output chip selects to the devices. The OUTCTL lines are used to generate Read and Write signals and also used for generating hand-shake signals to the host.

The data bus and associated CPU registers are used to read data in and out of PAC1000 for non-fly-by transfers.

Software Considerations

All the algorithms described so far are internally realized by Software. Flowcharts and partial code implementation (of all the important transfer procedures referred to in the flowcharts) can be found in Appendix 3 and Appendix 4, respectively. Both flowcharts and code listings contain sufficient explanations that let the reader understand the subjects they describe. The attached code listings cover all the important DMA transfer procedures (see Appendix 4).

Conclusion

PAC1000 is perfectly suitable for any DMA transfers which require an intelligent processor that can adapt its data handling according to the changing requirements of its interface. The PAC1000 does so by properly exploiting its unique structure of a very high speed sequencer combined with a programmable ALU and user configurable ports. The PAC1000's programmability enables it to handle complex tasks concurrently in a very efficient manner, unlike all other existing DMA controllers that are restricted to perform in a predefined environment.

Appendix 1:

The PAC1000 is configured in this application note as a generic DMA controller. It has a separate 24-bit address (that can be easily expanded) and a 16-bit data bus. It also has a set of control signals to enable operation as a bus master or a bus slave. The following table defines the individual PAC1000 pins. These brief descriptions are provided for reference only. Each signal is further detailed within the sections that describe the associated DMA function. For pin identifications refer to Figure 6.

Pin Descriptions

Symbol	Type	Name and Function
A(23:22)	O	**Address Lines A(23:22):** Output the two most significant address lines during Master operations. Tied to IO(5:4) on the PAC1000. Float in Slave Mode.
A(21:6)	O	**Address Lines A(21:6):** Output the mentioned address lines only in Master Mode. Connected to ADD(15:0) on the PAC1000. Float in Slave operations.
A(5:0)	I/O	**Address Lines A(5:0):** Bidirectional address lines. Input during Slave operations, output in Master mode. Tied to HAD(5:0) on the PAC1000.
FBRW2 FBRW1	O O	**Fly-by Read/Write (2:1):** Enable fly-by DMA operations. In fly-by mode, operands are transferred directly from the source to the destination bypassing the DMA controller. FBRW2 and FBRW1 are tied to OC6 and OC5, respectively. FBRW2 | FBRW1 0 | 0 — Normal operation. 0 | 1 — Enable fly-by from memory to device. 1 | 0 — Enable fly-by from device to memory. 1 | 1 — Illegal.
$\overline{WR}$	I	**Write:** Active as an input, only in Slave Mode. When low, HD(15:0) is written into the PAC1000.
$\overline{RD}$	I	**Read:** Active as an input, only in Slave Mode. When low, HD(15:0) is driven by the PAC1000.
$\overline{WRM}$	O	**Write-Out:** Active as an output, only in Master Mode. Enabled by BUSMSTR signal. Tied to OC4 on the PAC1000.
$\overline{RDM}$	O	**Read-Out:** Active as an output, only in Master Mode. Enabled by BUSMSTR signal. Tied to OC3 on the PAC1000.
$\overline{DMACK(3:0)}$	O	**DMA Acknowledge (3:0):** 4 active low signals. High in Slave Mode. Correspond to the 4 devices shown in Figure 6 respectively. Chip select the active devices during DMA operations. In the PAC1000 they are tied to IO(3:0) lines.
BUSMSTR	O	**Bus-Master:** An active high signal. Asserted whenever the PAC1000 is the current Bus Master. Informs arbiters or hosts not to access the bus before the PAC1000 relinquishes it. Enables OC4 and OC3 into $\overline{WR}$ and $\overline{RD}$, respectively. Connected to OC2 on the PAC1000.
$\overline{CSPAC}$	I	**PAC1000 Chip Select:** This pin is driven low whenever the PAC1000 is addressed in a slave bus read or write cycle.
$\overline{BR}$	O	**Bus Request:** The PAC1000 drives this pin low whenever it requests the bus due to pending DMA requests.

Appendix 1 (Cont.)

Pin Descriptions (Cont.)

Symbol	Type	Name and Function
HOSTINTR	O	**Host Interrupt:** The PAC1000 interrupts the host in order to inform him of one of the following events: PAC1000 exception, Terminal-Count or DMA aborted by a device. The OC0 line is assigned to this signal.
CLK	I	**Clock:** 20 MHz clock input to the PAC1000. It also latches the condition codes to ensure the proper Set-up time.
CC7	I	**DMA Abort:** An active-high input driven by the memories and/or devices currently participating in the DMA process. Whenever it is sensed high, the PAC1000 will generate a HOSTINTR signal towards the host after writing into the DOR register the appropriate status word.
CC6	I	**Bus Grant:** An active-low signal monitored by the PAC1000 to determine when it is in the Master mode or when to relinquish the buses and enter the Slave Mode.
CC4	I	**Ready:** An active-high signal (RDY) that enables the PAC1000 to synchronize its DMA cycles with slow memories or devices in the Master Mode.
CC(3:0)	I	**DMA Requests (3:0):** External DMA requests monitored by the PAC1000. Active-high signals, driven by the four devices.
$\overline{\text{DMAWORD}}$	O	**DMA Word or Byte Transfers:** Determines whether the next DMA cycle will be of word (low) or byte (high) length. Used to derive the $\overline{\text{BHE}}$ (Bus High Enable) signal that enables data lines D15:D8 in the Master Mode. $\overline{\text{BLE}}$ is directly driven by the A0 address line.
$\overline{\text{RSPAC}}$	I	**Reset PAC1000:** This asynchronous input initializes the state of. PAC1000. RESET must be held low for at least two clock cycles.
$\overline{\text{D(15:0)}}$	I/O	**Data-Bus (15:0):** This is the 16-bit data bus. During Master cycles, it is controlled and sometimes also driven by the PAC1000. In Slave mode the host drives it. Tied to HD(15:0) on the PAC1000.

Appendix 2: Host-DMA Message Formats

1) Host to PAC1000 Commands (via the FIFO)

HD(15:0) CONTENT

Word 1: Command word (see paragraph 3).
Word 2: 16 low-order source address lines.
Word 3: 8 high-order source address lines.
Word 4: 16 low-order destination address lines.
Word 5: 8 high-order destination address lines.
Word 6: 16 bit block-count.
Word 7: 8 bit DMA-transfer identification byte.
Word 8: Spare.

HAD(5:0) CONTENT

HAD5	HAD4	HAD3	HAD2	HAD1	HAD0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	0	1	1	1

2) PAC1000 to Host Status Word (via DOR register)

b15	b14	b13	b12	b11	b10	b09	b08	b07	b06	b05	b04	b03	b02	b01	b00

b15,b14,b13,b12,b11,b10,b9,b8: DMA-transfer identification byte.
b7,b6,b5,b4: spare.
b3: Reject or accept the DMA transfer identified by b15 ÷ b8. 1 — reject. 0 — accept.
b2: 1 — PAC1000 aborted. 0 — Normal operation
b1: 1 — DMA terminal-count completed 0 — Normal operation
b0: 1 — PAC1000 exception occurred 0 — Normal operation

Appendix 2 (Cont.)

3) Command Word Format

b15	b14	b13	b12	b11	b10	b09	b08	b07	b06	b05	b04	b03	b02	b01	b00

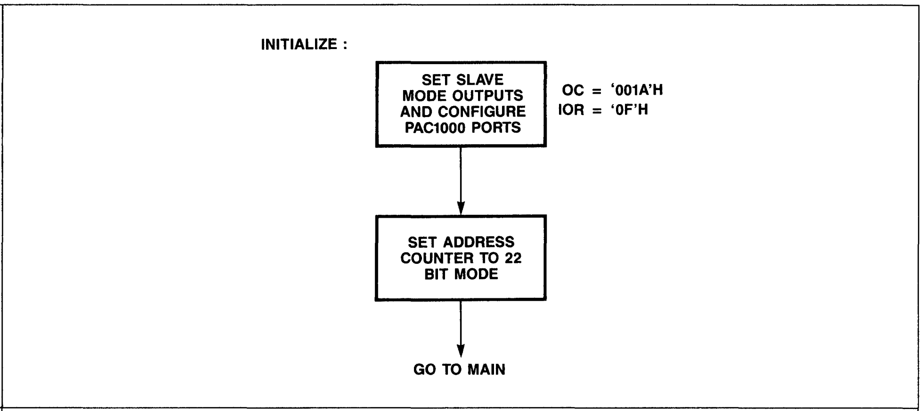
b15,b14:	spare.
b13:	block transfer or single transfer mode. 1 — DMA block operation. 0 — DMA single operand transfer mode.
b12:	DMA abort bit. Quits DMA-transfer specified in word 7. 1 — abort. 0 — nop.
b10,b9:	Priority level of this DMA-transfer. 00 — level 0 (lowest priority level). 01 — level 1 • 02 — level 2 • 03 — level 3 (highest priority level).
b9,b8:	Source Device number for DMA transfer or Abort. 00 — Device #0 01 — Device #1 02 — Device #2 03 — Device #3
b7,b6:	Dest. Device number for DMA transfer or Abort. 00 — Device #0 01 — Device #1 02 — Device #2 03 — Device #3
b5,b4:	Destination data bus definition. 00 — Data bus is D7–D0 (bit bits). 01 — Data bus is D15–D8 (8 bits). 02 — Data bus is D15–D0 (16 bits). 03 — Illegal.
b3,b2:	Source data bus definition. 00 — Data bus is D7–D0 (8 bits). 01 — Data bus is D15–D8 (8 bits). 02 — Data bus is D15–D0 (16 bits). 03 — Illegal.
b1,b0:	DMA transfer mode. 00 — Memory to memory. 01 — Memory to device. 02 — Device to device. 03 — Device to memory.

Appendix 3

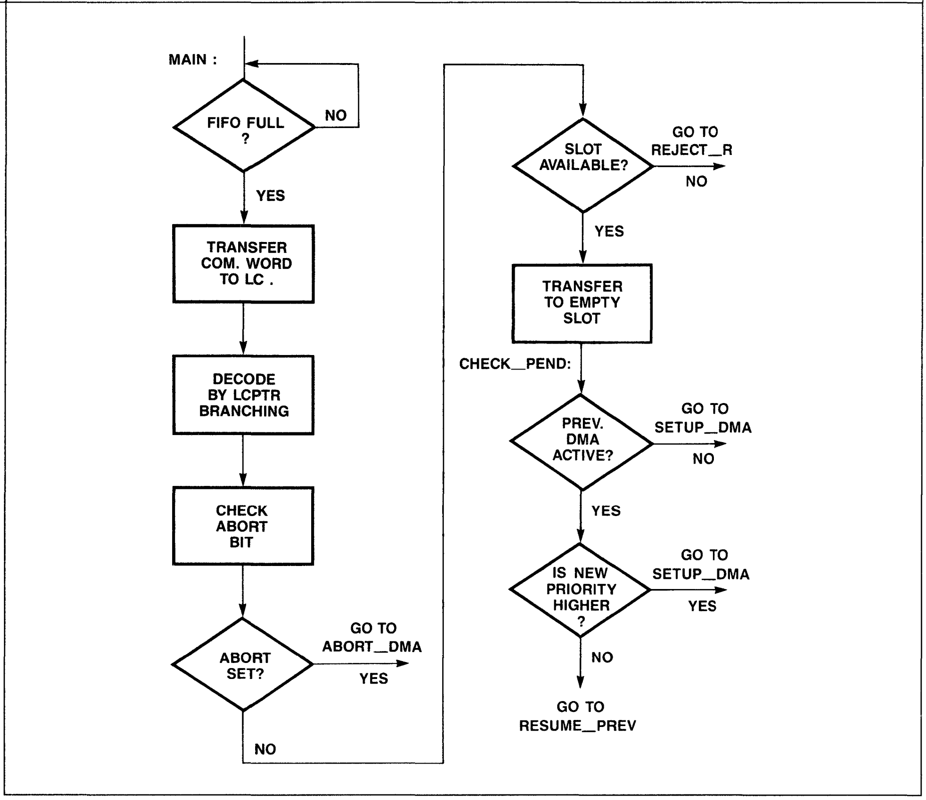
General Note:

Code implementation of labels marked with an asterisk (*) can be found in Appendix 4.

Initialization


INITIALIZE :
SET SLAVE MODE OUTPUTS AND CONFIGURE PAC1000 PORTS
OC = '001A'H
IOR = '0F'H
SET ADDRESS COUNTER TO 22 BIT MODE
GO TO MAIN

Main Loop

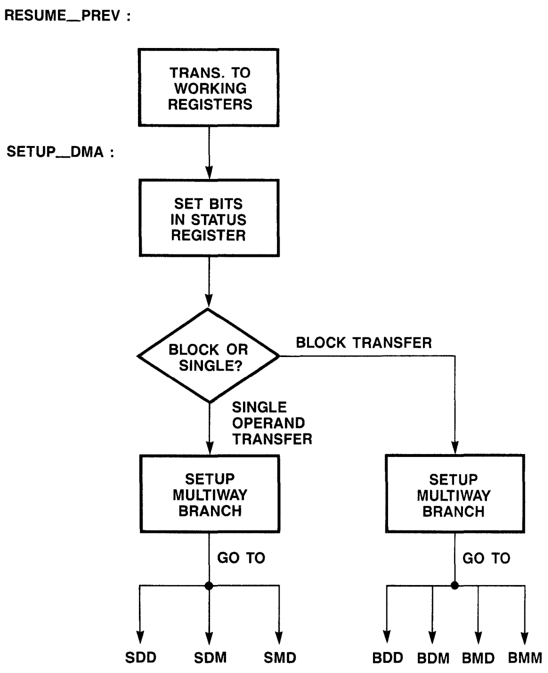


Legend:

1. **Slot:** The PAC1000 can handle up to 4 DMA channels concurrently. Slot means empty register space inside the PAC1000 that is allocated for a pending channel.
2. **LCPTR branching:** A goto instruction of the command section, enabling multi-way branching of the program according to a value loaded into the LC register by the ALU (executed in two cycles).

Appendix 3 (Cont.)

Setting Up the Transfer



Legend:

1. SDD — single operand transfer, device to device.
2. SDM — single operand transfer, device to memory.
3. SMD — single operand transfer, memory to device.
4. BDD — block transfer, device to device.
5. BDM — block transfer, device to memory.
6. BMD — block transfer, memory to device.
7. BMM — block transfer, memory to memory.

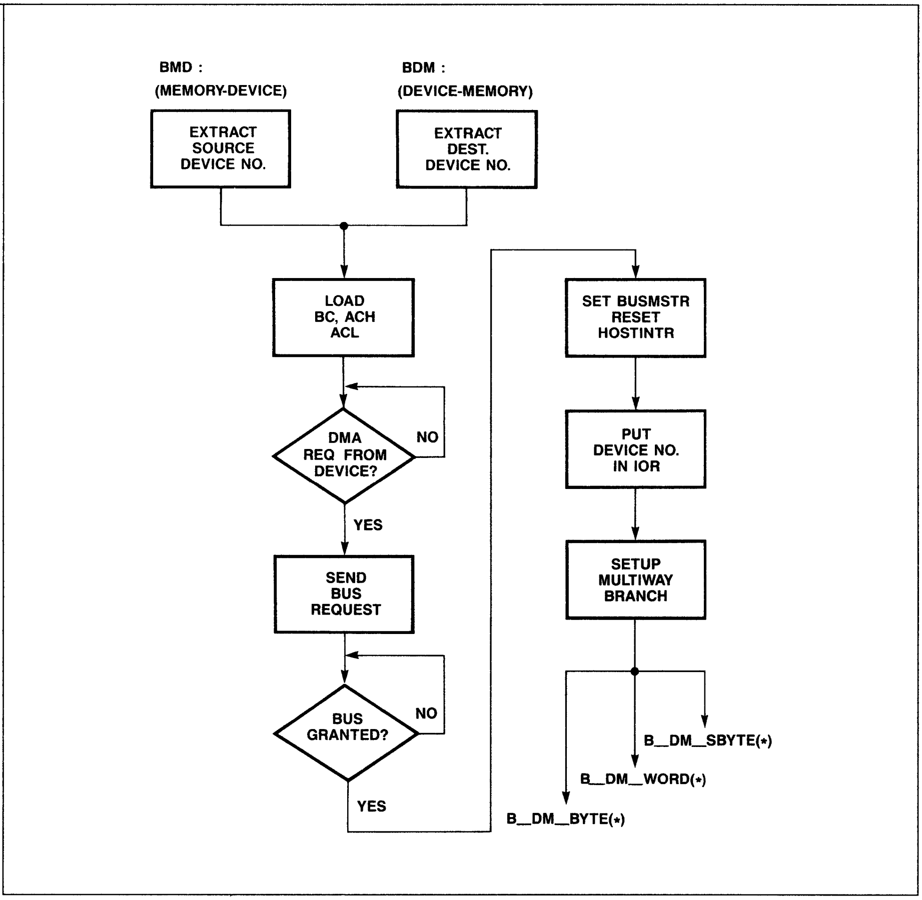
General Remarks:

In a single operand transfer, at least one of the involved devices requests a DMA transfer for each operand. This method is used with slow devices.

Block transfers are used to move data blocks between fast memories and/or devices. A DMA request is set for every block transfer.

Appendix 3 (Cont.)

Device to Memory Block Transfer

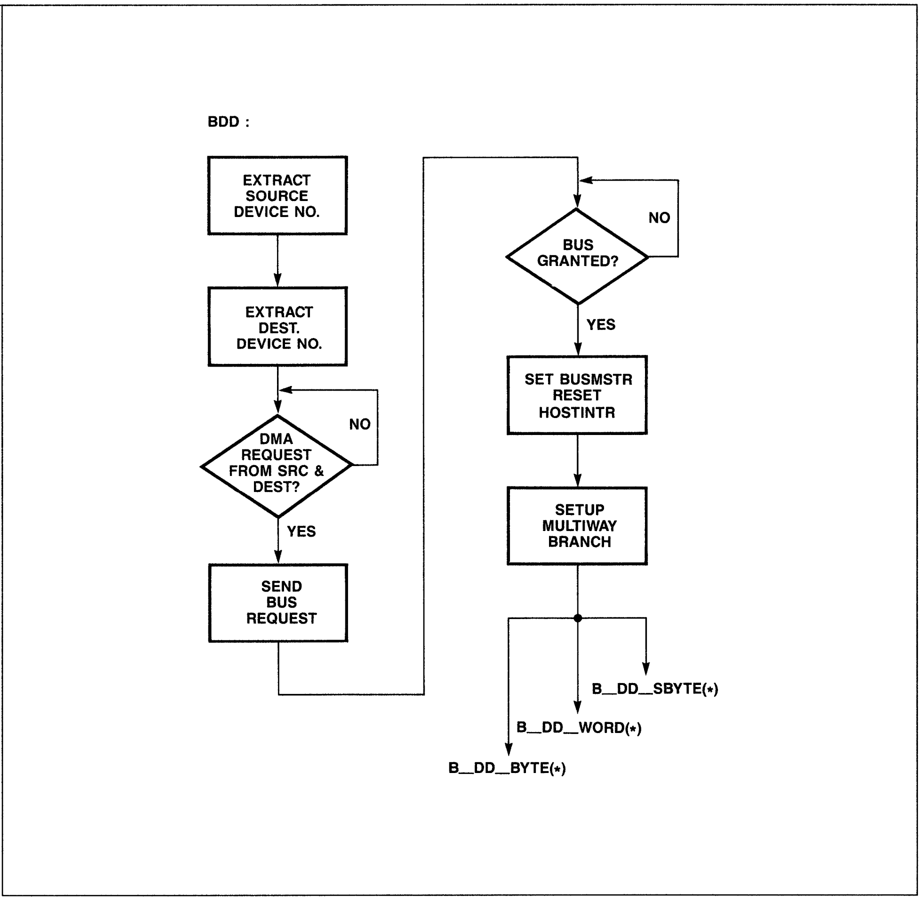


Legend:

1. B__dm__byte: block device to/from memory transfer of bytes.
2. B__dm__word: block device to/from memory transfer of words.
3. B__dm__sbyte: block device to/from memory transfer of swapped bytes. Occurs whenever the transfer is between even and odd addresses.

Appendix 3 (Cont.)

Device to Device Block Transfer

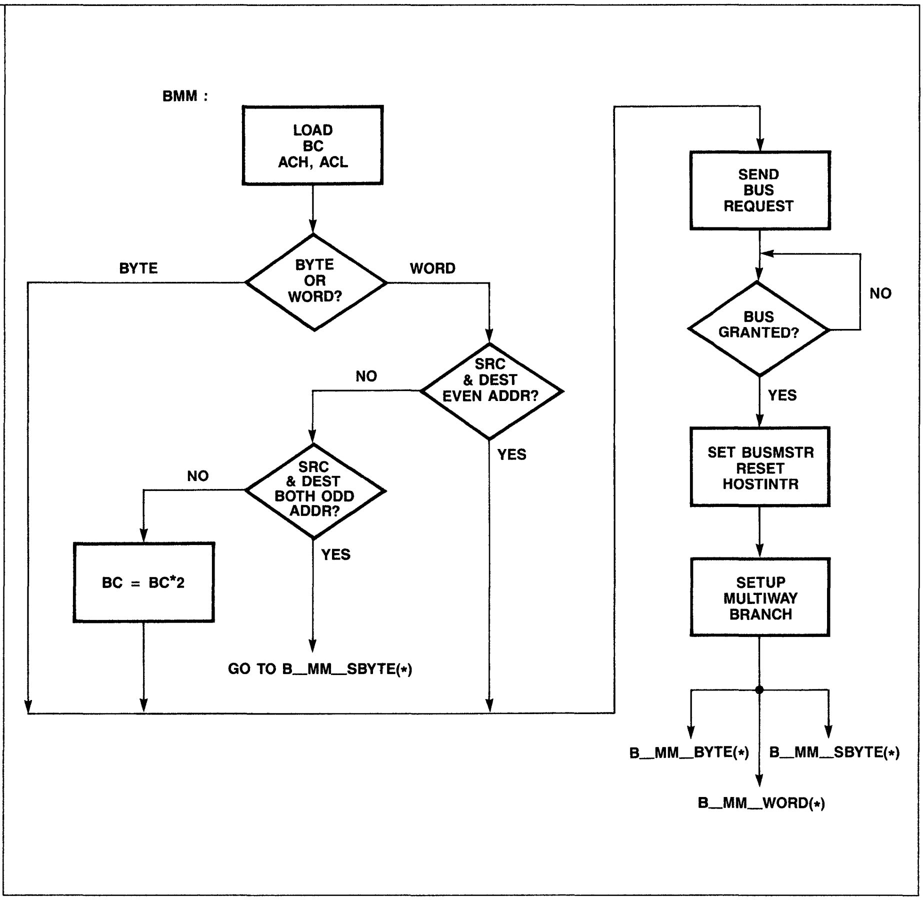


Legend:

1. B__dd__byte: block device to device transfer of bytes.
2. B__dd__word: block device to device transfer of words.
3. B__dd__sbyte: block device to device transfer of swapped bytes. Happens whenever the transfer is between even and odd addresses.

Appendix 3 (Cont.)

Memory to Memory Block Transfer

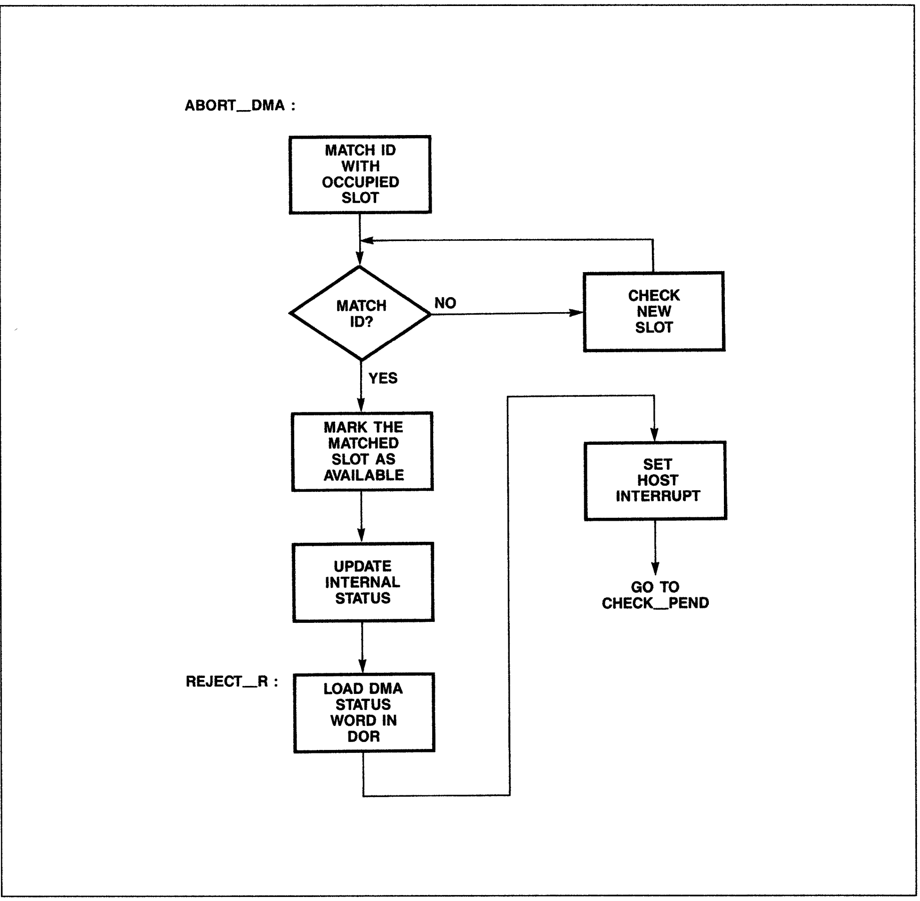


Legend:

1. B_mm_byte: block memory to memory transfer of bytes.
2. B_mm_word: block memory to memory transfer of words.
3. B_mm_sbyte: block memory to memory transfer of swapped bytes. Occurs whenever the transfer is between even and odd addresses.

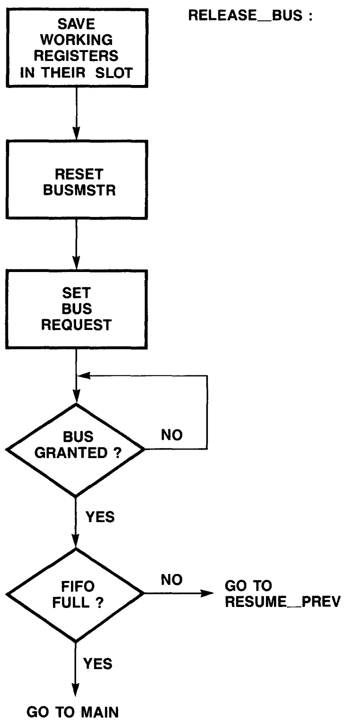
Appendix 3 (Cont.)

Abort DMA Transfer

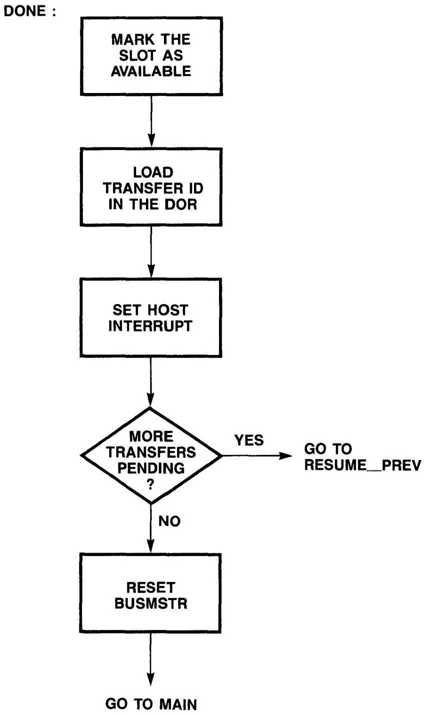


Appendix 3 (Cont.)

Bus Release



End of Transfer



Appendix 4

```
/************************************************************************/
/* device to memory byte transfer in the fly-by mode. The start address */
/* of the memory is loaded in  R3 and R4 and the device number is loaded*/
/* in Q . Assume that the initial protocol has been gone through and    */
/* PAC has  control of the bus.   For simplicity it is assumed that the*/
/* block size is a multiple of 64 and R5*64 = block size.              */
/************************************************************************/
segment b_dm_byte ;
     /* define equates */
     bgn equ CC7                 ;  /* bus grant (active low)     */
     ready equ CC4               ;  /* ready input                */
     b_dm_byte_norm equ h'00de' ;  /* dma active w/o read/write  */
     b_dm_byte_read equ h'00d6' ;  /* read (active low )         */
     b_dm_byte_write equ h'00c6';  /* write (active low )        */
init_b_dm_byte :
     ACH := R3                   ;  /* upper 16 bits address      */
     SET ASEL ADOE HADOE   ,
     ACL := R4                   ;  /* select counter to output ,
                                    enable ADD and HAD output, and
                                    load lower address in ACL    */
     IOR := ~ Q ,
     OUT b_dm_byte_norm          ;  /* select device #       */
     Q := 1                      ;  /* address increment for byte   */
     LDLCD , MOV R5 R5           ;  /* R5 * 64 -> block count      */
/************************************************************************/
/* start of outer transfer loop                                         */
/************************************************************************/
x1:  PLDLC H'3F'                 ;  /* push cnt to stack and load 64
                                    in cnt             */
/************************************************************************/
/* start inner transfer loop                                            */
/************************************************************************/
y1 : JMPNC ready y1 ,
     OUT b_dm_byte_read          ;   /* wait till ready signal high    */
     LOOPNZ y1    ,
     ACL := ACL + Q ,
     OUT b_dm_byte_write         ;   /* strobe the write signal and
                                    set up the next address      */
/************************************************************************/
/* end inner loop                                                       */
/************************************************************************/
     POPLC   ,
     ACH := ++ ACH ,
     OUT b_dm_byte_read          ;  /* pop stack to cnt , increment
                                    upper address bits       */
     JMPC bgn release_bus        ;  /* check if bus grant has been
                                    taken away               */
     LOOPNZ x1                   ;  /* loop back if counter not zero*/
/************************************************************************/
/* end outer loop                                                       */
/************************************************************************/
done :
     ..................
     ..................
release_bus :
     ..................
/************************************************************************/
```

Appendix 4 (Cont.)

```
/************************************************************************/
/* device to memory word transfer in the fly-by mode. The start address */
/* of the memory is loaded in  R3 and R4 and the device number is loaded*/
/* in Q . For simplicity it is assumed that the block size is a multiple*/
/* of 64 and R5*64 = block size.                                        */
/************************************************************************/
segment b_dm_word ;
     /* define equates */
     bgn equ CC7            ; /* bus grant (active low)     */
     ready equ CC4          ; /* ready input                */
     b_dm_word_norm equ h'00de' ;  /* dma active w/o read/write  */
     b_dm_word_read equ h'00d6' ;  /* read (active low )         */
     b_dm_word_write equ h'00c6';  /* write (active low )        */
init_b_dm_word :
     ACH := R3                  ;  /* upper 16 bits address      */
     SET ASEL ADOE HADOE  ,
     ACL := R4                  ;  /* select counter to output ,
                                   enable ADD and HAD output, and
                                   load lower address in ACL    */
     IOR := ~ Q ,
     OUT b_dm_word_norm         ;  /* select device #       */
     Q := 2                     ;  /* address increment for word   */
     LDLCD , MOV R5 R5          ;  /* R5 * 64 -> block size (words)*/
/************************************************************************/
/* start of outer transfer loop                                         */
/************************************************************************/
x1:  PLDLC H'1F'                ;  /* push cnt to stack and load 32
                                   in cnt           */
/************************************************************************/
/* start inner transfer loop                                            */
/************************************************************************/
y1 : JMPNC ready y1 ,
     OUT b_dm_word_read         ;   /* wait till ready signal high    */
     LOOPNZ y1    ,
     ACL := ACL + Q ,
     OUT b_dm_word_write        ;   /* strobe the write signal and
                                   set up the next address      */
/************************************************************************/
/* end inner loop                                                       */
/************************************************************************/
     POPLC   ,
     ACH := ++ ACH ,
     OUT b_dm_word_read         ;  /* pop stack to cnt , increment
                                   upper address bits        */
     JMPC bgn release_bus       ;  /* check if bus grant has been
                                   taken away              */
     LOOPNZ x1                  ;  /* loop back if counter not zero*/
/************************************************************************/
/* end outer loop                                                       */
/************************************************************************/
done :
     ..................
     ..................
release_bus :
     ..................
/************************************************************************/
```

Appendix 4 (Cont.)

```
/**************************************************************************/
/* device to memory byte transfer in the fly-by mode. The start address */
/* of the memory is loaded in  R3 and R4 and the device number is loaded*/
/* in Q . For simplicity it is assumed that the block size is a multiple*/
/* of 64. This code illustrates individual transfer mode(non-block mode)*/
/**************************************************************************/
segment s_dm_byte ;
     /* define equates */
     bgn equ CC7             ; /* bus grant (active low)     */
     ready equ CC4           ; /* ready input                */
     s_dm_byte_norm equ h'00de' ;  /* dma active w/o read/write    */
     s_dm_byte_read equ h'00d6' ;  /* read (active low )          */
     s_dm_byte_write equ h'00c6';  /* write (active low )         */
init_s_dm_byte :
     ACH := R3                  ;  /* upper 16 bits address         */
     SET ASEL ADOE HADOE  ,
     ACL := R4                  ;  /* select counter to output ,
                                   enable ADD and HAD output, and
                                   load lower address in ACL    */
     BC := R5                   ;  /* load block size in to BC   */
     IOR := ~ Q ,
     OUT s_dm_byte_norm         ;  /* select device #       */
     CMP Q H'0001'          ;      /* find out if device #0        */
     JMPC Z dev0            ;
     CMP Q H'0002'          ;      /* if device # 1         */
     JMPC Z dev1            ;
     CMP Q H'0004'          ;      /* if device # 2         */
     JMPC Z dev2            ;
                                   /* else it is device # 3
/**************************************************************************/
/* start  transfer loop for dev#3                                         */
/**************************************************************************/
dev3 :
     JMPC bgn release_bus        ;  /* monitor bus grant             */
     JMPNC CC3 dev3        ,
     OUT s_dm_byte_read         ;   /* branch to check for dma request
                                   from  device3          */
     SET ACEN BCEN ,
     OUT s_dm_byte_write    ;      /* start counter         */
     RESET ACEN BCEN ,
     OUT s_dm_byte_norm     ;       /* stop counter            */
     JMPNC BCZ dev3              ;  /* loop back if not done        */
     JMP done          ;

/**************************************************************************/
/* start  transfer loop for dev#2                                         */
/**************************************************************************/
dev2 :
     JMPC bgn release_bus        ; /* monitor bus grant            */
     JMPNC CC2 dev2        ,
     OUT s_dm_byte_read          ; /* branch to check for dma request
                                   from  device2             */
     SET ACEN BCEN ,
     OUT s_dm_byte_write         ; /* start counter         */
```

Appendix 4 (Cont.)

```
     RESET ACEN BCEN ,
     OUT s_dm_byte_norm            ;  /* stop counter                */
     JMPNC BCZ dev2                ;  /* loop back if not done */
     JMP done                      ;
/************************************************************************/
/* start  transfer loop for dev#1                                      */
/************************************************************************/
dev1 :
     JMPC bgn release_bus           ;    /* monitor bus grant            */
     JMPNC CC1 dev1          ,
     OUT s_dm_byte_read            ;     /* branch to check for dma request
                                           from  device1                 */
     SET ACEN BCEN ,
     OUT s_dm_byte_write           ;    /* start counter       */
     RESET ACEN BCEN ,
     OUT s_dm_byte_norm            ;    /* stop counter              */
     JMPNC BCZ dev1                ;    /* loop back if not done    */
     JMP done              ;
/************************************************************************/
/* start  transfer loop for dev#0                                      */
/************************************************************************/
dev0 :
     JMPC bgn release_bus           ;    /* monitor bus grant            */
     JMPNC CC3 dev0          ,
     OUT s_dm_byte_read            ;     /* branch to check for dma request
                                           from  device3                 */
     SET ACEN BCEN ,
     OUT s_dm_byte_write       ;    /* start counter      */
     RESET ACEN BCEN ,
     OUT s_dm_byte_norm        ;    /* stop counter              */
     JMPNC BCZ dev0            ;    /* loop back if not done    */

/************************************************************************/
done :
     .....................
     .....................
release_bus :
     .....................
/************************************************************************/
```

Appendix 4 (Cont.)

```
/**********************************************************************/
/* code to illustrate device to memory transfer in non fly by mode .    */
/* This is used when data bus is connected d7-d0 to d15-d8  or the      */
/* other way around. Use counter to output addresses.Q contains device  */
/* number and R3 R4 contain destination address.R5 contains block size. */
/**********************************************************************/
segment b_dm_sbyte ;
     /* define equates */
     b_dm_sbyte_norm equ h'009e' ;
     b_dm_sbyte_read equ h'0096' ;
     b_dm_sbyte_write equ h'008e';
     rdy equ CC4            ;
     bgn equ CC7            ;
init_b_dm_sbyte :
     BC := R5 ,
     OUT b_dm_sbyte_norm     ;    /* load block size in bcnt      */
     SET DIREN ASEL HADOE ADOE   ;/* select  counter to output ,
                                    enable had output             */
     ACH := R3          ;
     ACL := R4          ;
/**********************************************************************/
/* start of transfer loop                                               */
/**********************************************************************/
b_dm_sbyte :
     JMPC bgn release_bus        ;
     SET DIREN                   ; /* enable DIR                   */
srdy :
     JMPNC rdy  srdy ,
     OUT b_dm_sbyte_read         ; /* wait till source ready      */
     SET HDOE HDSEL0 ,
     AOR := DIR              ;     /* when src is ready read the data
                                   in , enable HD output , select
                                   DOR to output           */
     DOR := SWPV    ,
     OUT b_dm_sbyte_write         ;    /* put swapped data in DOR      */
     SET ACEN BCEN ,
     OUT b_dm_sbyte_norm     ;     /* start counter , output swapped
                                   data                    */
     RESET ACEN BCEN HDOE        ;
     JMPNC BCZ b_dm_sbyte        ;
/**********************************************************************/
/* end of transfer loop                                                 */
/**********************************************************************/
done :
     ...................
release_bus :
     ...................
/**********************************************************************/
```

Appendix 4 (Cont.)

```
/************************************************************************/
/* code to illustrate memory to memory transfer.Use counter to output   */
/* both addresses.R1,R2 contain source address and R3 R4 contain dest   */
/* address . R5 contains block size.                                     */
/************************************************************************/
segment b_mm_byte ;
     /* define equates */
     b_mm_byte_norm equ h'009e' ;
     b_mm_byte_read equ h'0096' ;
     b_mm_byte_write equ h'008e';
     rdy equ CC4            ;
     bgn equ CC7            ;
init_b_mm_byte :
     BC := R5 ,
     OUT b_mm_byte_norm      ;    /* load block size in bcnt */
     SET ASEL HADOE ADOE         ;    /* select  counter to output ,
                                   enable had output           */
/************************************************************************/
/* start of transfer loop                                               */
/************************************************************************/
b_mm_byte :
     JMPC bgn release_bus ,
     ACH := R1          ;         /* monitor bus grant , source
                                   address in R1        */
     SET DIREN , ACL := R2       ;  /* enable dir, r2 <- low 6 bits */
srdy :
     JMPNC rdy  srdy ,
     OUT b_mm_byte_read      ;      /* wait till source ready     */
     SET ACEN HDOE HDSEL0 ,
     DOR := DIR              ;      /* when src is ready read the data
                                   in , enable HD output , select
                                   DOR to output          */
     RESET ACEN DIREN ,R1 := ACH,
     OUT b_mm_byte_norm      ;      /* stop counter , store it back in
                                   to registers           */
     ADD R2 ACL Q ARDREG ACH R3 ;  /* mov ACL back to r1 and at the
                                   same time load r3 to ach     */
     ACL := R4                  ;  /* ach,acl have dest address    */
drdy :
     JMPNC rdy drdy             ;  /* wait for destination ready   */
     SET ACEN BCEN ,
     OUT b_mm_byte_write         ; /* when dest is ready , write the
                                   data, increment counter , also
                                   enable block counter     */
     RESET ACEN BCEN HDOE ,
     R3 := ACH ,
     OUT b_mm_byte_norm      ;      /* stop counters , set HD to input
                                   save dest address (upper 16) */
     JMPNC BCZ b_mm_byte ,
     R4 := ACL          ;           /* loop back if block counter not
                                   zero , also save lower 6 bits
                                   of dest address        */
/************************************************************************/
/* end of transfer loop                                                 */
/************************************************************************/
done :
     .....................
release_bus :
     .....................
/************************************************************************/
```

Appendix 4 (Cont.)

```
/************************************************************************/
/* code to illustrate memory to memory transfer (word mode).Use counter */
/* to output both addresses.R1,R2 contain source address and R3 R4      */
/* contain destination address . R5 contains block size in words.       */
/************************************************************************/
segment b_mm_word ;
     /* define equates */
     b_mm_word_norm equ h'009e' ;
     b_mm_word_read equu h'0096' ;
     b_mm_word_write equ h'008e';
     rdy equ CC4          ;
     bgn equ CC7          ;
init_b_mm_word :
     BC := R5 ,
     OUT b_mm_word_norm         ;    /* load block size in bcnt  */
     SET ASEL HADOE ADOE        ;    /* select  counter to output ,
                                   enable had output          */
/************************************************************************/
/* start of transfer loop                                               */
/************************************************************************/
b_mm_word :
     JMPC bgn release_bus ,
     ACH := R1                 ;   /* monitor bus grant , source
                                   address in R1       */
     SET DIREN , ACL := R2     ;   /* enable dir,ACL <- low 6 bits */
srdy :    JMPNC rdy  srdy ,
     OUT b_mm_word_read        ;   /* wait till source ready     */
     SET ACEN HDOE HDSEL0 ,
     DOR := DIR                ;   /* when src is ready read the data
                                   in , enable HD output , select
                                   DOR to output          */
     OUT b_mm_word_norm       ;
     RESET ACEN DIREN ,
     ADD R1 ACH Q ARDREG ACH R3 ;  /* stop counter , store ACH in to
                                   R1 and also load ACH with R3 */
     ADD R2 ACL Q ARDREG ACL R4 ;  /* store ACL in R2 and at the same
                                   time put R4 in to ACL     */
drdy :    JMPNC rdy drdy        ;  /* wait for destination ready   */
     SET ACEN BCEN ,
     OUT b_mm_word_write        ;  /* when dest is ready , write the
                                   data, increment counter , also
                                   enable block counter     */
     RESET BCEN HDOE ,
     OUT b_mm_word_norm        ;   /* stop block counter, set HD to
                                   input                  */

     RESET ACEN   ,  R3 := ACH  ;  /* stop add counter ,
                                   save dest address (upper 16) */
     JMPNC BCZ b_mm_word ,
     R4 := ACL                 ;   /* loop back if block counter not
                                   zero , also save lower 6 bits
                                   of dest address         */
/************************************************************************/
/* end of transfer loop                                                 */
/************************************************************************/
done :
     ....................
release_bus :
     ....................
/************************************************************************/
```

Appendix 4 (Cont.)

```
/**************************************************************************/
/* code to illustrate memory to memory transfer  from D7-D0 to D15-D8    */
/* or vice-versa. Use counter to output both addresses .R1 , R2 contain */
/* source address and R3 R4 contain destination address.R5 contains     */
/* block size. Data is read in to AOR and byte-swpped before outputting */
/* through DOR.                                                          */
/**************************************************************************/
segment b_mm_sbyte ;
     /* define equates */
     b_mm_sbyte_norm equ h'009e' ;
     b_mm_sbyte_read equ h'0096' ;
     b_mm_sbyte_write equ h'008e';
     rdy equ CC4             ;
     bgn equ CC7             ;
init_b_mm_sbyte :
     BC := R5,OUT b_mm_sbyte_norm;     /* load block size in bcnt */
     SET ASEL HADOE ADOE          ;     /* select  counter to output ,
                                           enable had output           */
/**************************************************************************/
/* start of transfer loop                                                 */
/**************************************************************************/
b_mm_sbyte :
     JMPC bgn release_bus ,
     ACH := R1           ;            /* monitor bus grant , source
                                      address in R1       */
     SET DIREN , ACL := R2       ;   /* enable dir, r2 <- low 6 bits */
srdy :    JMPNC rdy  srdy ,
     OUT b_mm_sbyte_read          ; /* wait till source ready     */
     SET ACEN HDOE HDSEL0 ,
     AOR := DIR              ;        /* when src is ready read the data
                                      in , enable HD output , select
                                       DOR to output          */
     RESET ACEN DIREN,R1 := ACH ,
     OUT b_mm_sbyte_norm         ;   /* stop counter , store it back in
                                      to registers          */
     ADD R2 ACL Q ARDREG ACH R3 ;   /* mov ACL back to r1 and at the
                                      same time load r3 to ach      */
     ACL := R4                   ;   /* ach,acl have dest address    */
drdy :    JMPNC rdy drdy,DOR := SWPV ; /* wait for destination ready
                                      and write swapped value       */
     SET ACEN BCEN ,
     OUT b_mm_sbyte_write         ; /* when dest is ready , write the
                                      data, increment counter , also
                                      enable block counter     */
     RESET ACEN BCEN HDOE ,
     R3 := ACH ,
     OUT b_mm_sbyte_norm          ; /* stop counters , set HD to input
                                      save dest address (upper 16) */
     JMPNC BCZ b_mm_sbyte ,
     R4 := ACL           ;            /* loop back if block counter not
                                      zero , also save lower 6 bits
                                      of dest address         */
/**************************************************************************/
/* end of transfer loop                                                   */
/**************************************************************************/
done :
     ...................
release_bus :
     ...................
/**************************************************************************/
```

4

Appendix 4 (Cont.)

```
/************************************************************************/
/* code to illustrate device to device transfers in the byte as well as */
/* word mode.  source device is in r1 and dest device is in r3. block   */
/* size is in r5.                                                       */
/************************************************************************/
segment b_dd_bw ;
     /* define equates */
     b_dd_bw_norm equ h'009e'  ;
     b_dd_bw_read equ h'0096'  ;
     b_dd_bw_write equ h'008e' ;
     rdy equ CC4           ;
     bgn equ CC7           ;
init_b_dd_bw :
     SET DIREN , IOR := ~ R1 ,
     OUT b_dd_bw_norm          ;    /* enable DIR and output source
                                    device chip select        */
/************************************************************************/
/* start of transfer loop                                               */
/************************************************************************/
b_dd_byte :
b_dd_word :
b_dd_bw :
     JMPC bgn release_bus ,
     IOR := ~ R3 ,
     OUT b_dd_bw_read      ;  /* read source device and output
                              dest device chip select , also
                              monitor bus grant         */
     SET  HDOE HDSEL0 ,
     DOR := DIR ,
     OUT b_dd_bw_norm      ;  /* enable HD output , select DOR
                              to output                 */

     RESET HDOE ,
     DEC R5 ,
     OUT b_dd_bw_write     ;  /* HD to input , decrement count ,
                              output write strobe       */
     JMPNC Z b_dd_bw ,
     IOR := ~ R1 ,
     OUT b_dd_bw_norm      ;  /* loop back if R5 not zero , also
                              output src device cs         */
/************************************************************************/
/* end of transfer loop                                                 */
/************************************************************************/
done :
     ....................
     ....................
release_bus :
     ....................
/************************************************************************/
```

Appendix 4 (Cont.)

```
/***********************************************************************/
/* code to illustrate device to device transfer in non fly by mode .    */
/* This is used when data bus is connected d7-d0 to d15-d8  or the      */
/* other way around. Source device # is in R1 and dest device # in R3   */
/***********************************************************************/
segment b_dd_sbyte :
     /* define equates */
     b_dd_sbyte_norm equ h'009e' ;
     b_dd_sbyte_read equ h'0096' ;
     b_dd_sbyte_write equ h'008e';
     rdy equ CC4               ;
     bgn equ CC7               ;
init_b_dd_sbyte :
     SET DIREN , IOR := ~ R1 ,
     OUT b_dd_sbyte_norm          ; /* enable DIR and output source
                                    device chip select       */
/***********************************************************************/
/* start of transfer loop                                               */
/***********************************************************************/
b_dd_sbyte :
     JMPC bgn release_bus ,
     IOR := ~ R3 ,
     OUT b_dd_sbyte_read          ; /* read source device and output
                                     dest device chip select , also
                                     monitor bus grant        */
     AOR := DIR               ;     /* read in the data       */
     SET  HDOE HDSEL0 ,
     DOR := SWPV ,
     OUT b_dd_sbyte_write         ; /* enable HD output , select DOR
                                    to output , put swapped data in
                                    DOR                  */
     RESET HDOE ,
     DEC R5 ,
     OUT b_dd_sbyte_norm          ; /* HD to input , decrement count ,
                                    output write strobe       */
     JMPNC Z b_dd_sbyte ,
     IOR := ~ R1                  ; /* loop back if R5 not zero , also
                                    output src device cs        */
/***********************************************************************/
/* end of transfer loop                                                 */
/***********************************************************************/
done :
     ....................
     ....................
release_bus :
     ....................
/***********************************************************************/
```

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Brief 006

PAC1000 as a 16 Bi-Directional Serial Channel Controller

By Arye Ziklik

Introduction

This Application Brief describes a Communications Controller that utilizes the PAC1000 as the board level control element in a 16 bi-directional serial channel board. The aggregate board throughput is around *1 Mbyte/sec.*

Serialization and de-serialization of the data is handled by eight Serial Communication Controllers (SCC). Every SCC has two bi-directional serial channels with individual baud rate generator and digital phase loop mechanism. The SCC can handle all the customary synchronous and asynchronous protocols as well as the popular serial data encoding/decoding schemes. With a 16-MHz clock, the maximum bit rate in every individual channel can be up to 2 Mbps.

PAC1000 — Host Interface

The PAC1000 performs the low level function of moving the data to and from the serial devices and buffer RAM memory. The host interface is a generic 32-bit system. The host processor communicates with the PAC1000 through two interrupt lines, two status signals and a mail-box area that resides in the buffer memory. Prior to accessing the board, the host drives the "system board access" signal. The PAC1000 is interrupted (INT3) and relinquishes control of the board's data and address buses as long as that signal is active (as reflected by CC0). The host reads and/or writes into the buffer memory. After completion of this activity, it updates the mail-box region and then lowers the "system board access" signal. The PAC1000 continuously monitors that signal. After CC0 is negated, the PAC1000 can raise its "PAC1000-board master" signal and start controlling the data/address buses and control signals. Whenever it needs a fast response from the host, the PAC1000 updates the mail-box portion of the shared buffer memory, lowers the "PAC1000-board master" signal and activates the system interrupt.

Buffer Memory Structure

The high speed buffer memory is composed of 64K bytes of static RAM that can be accessed in three ways: by bytes (during SCC transfer operations), by words (when accessed by the PAC1000), or by double words (from the host side). Memory access configuration is determined by the PAC1000 output control signals (OC port).

The buffer memory is divided into three regions:

1) SCC control image register space that includes copies of the SCC registers.

2) Buffer message space where the 32 buffers of the corresponding serial channels are stored.

3) Mail-box area in which the PAC1000 exchanges command and status information with the host. This region also contains the pointers to the 32 channel buffers.

Whenever instructed to do so, the PAC1000 writes the image register content of a channel into the corresponding SCC, thereby initializing that channel for a particular transfer mode. Buffer message sizes are allocated by the host according to the speed of each individual channel. The pointers of the buffers are stored in the mail-box area.

Every transfer takes place between the buffer memory and the selected SCC. The PAC1000 is acting in this design as a 32-channel DMA controller, capable also of communicating with the host processor through their mail-box region. Once the board is properly configured, the only interface of the host system is the reading of data from the receive and mail-box buffers and the placing of new data into the transmit and mail-box buffers. The PAC1000 off-loads the host processor from maintaining the low level control of each channel.

PAC1000 — SCC Devices Interface

The high speed data transfers are achieved due to the very fast response of the PAC1000 to the channel service requests. The SCCs are programmed to request DMA transfers whenever they are either ready to transmit or containing new received characters.

The 16 received character DMA requests are priority encoded and latched. The encoder output is connected to the PAC1000's CC3 pin. The 16 transmit DMA requests are priority encoded and latched, too. Their encoder drives the CC2 input pin. The condition code multiplexer presents to the CC7–CC4 the highest priority encoded-channel-number of the pending receiver request, or the transmitter request, or the highest priority SCC number that is currently requesting an interrupt service via the CC1 pin. The receiver requests have higher priority over the transmitter requests. The lowest service priority is assigned to the SCC interrupts. This configuration ensures a very fast response time of the PAC1000 to DMA requests and SCC interrupts. Condition code latency is 125 ns and multi-way branching according to the CC7–CC4 lines requires additional 125 ns. Therefore, 250 ns after a high priority DMA request, the service routine will be initiated. The condition code lines CC3, CC2 and CC1 are continuously monitored by the PAC1000 during the time that it is the board master. Therefore it responds immediately when either a DMA request or an SCC interrupt is pending.

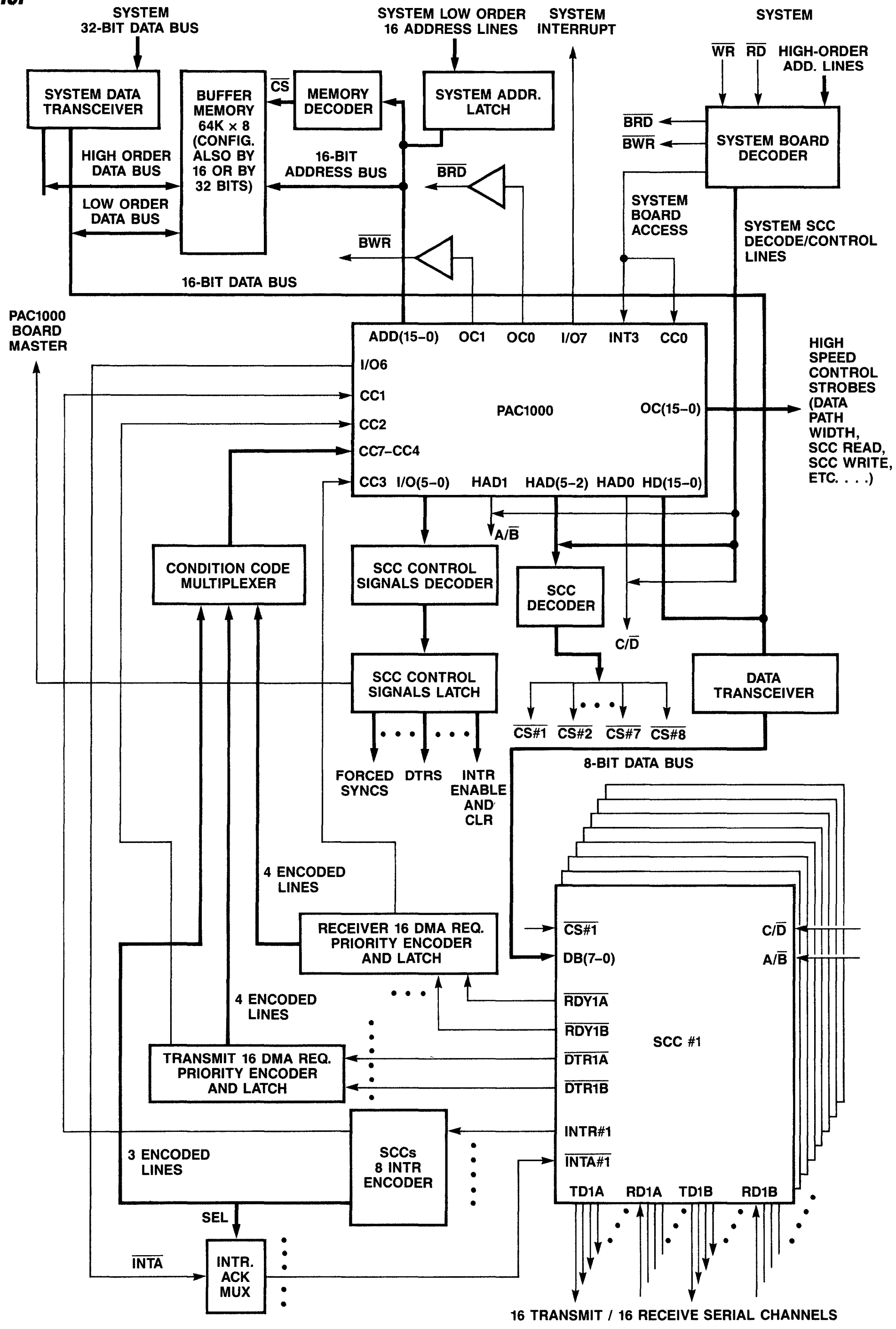
The regular SCC interrupt lines are also prioritized and latched by an 8 interrupt encoder. These interrupts are requested by erroneous SCC channels or whenever block transfers are completed. The interrupt priority encoder is also connected to the condition code multiplexer. The three encoded lines that denote the number of the serviced SCC route the INTA signal issued by the PAC1000 (via the I/O6 pin) to the corresponding SCC.

Miscellaneous

In addition to functioning as an SCC controller, the PAC1000 can also generate all the necessary signals for modem control and modem interface through the SCC control signal latch.

The PAC1000 output control (OC) port generates various control strobes such as data path width definition, read/write, multiplexer and decoder select, etc.

PAC1000 as a 16 Bi-Directional Serial Channel Controller



Programmable System™ Device

Application Note 008

PAC1000 User-Configurable Microcontroller with a Built-In-Self-Test Capability

By David Fong

Abstract

The objective of this Application Note is to demonstrate the Built-In-Self-Test (BIST) capability of the PAC1000 High-Performance User-Configurable Microcontroller. This article describes the basic instructions needed to implement BIST.

Introduction

With increasing device densities on one chip, more devices are needed for BIST to check the functionality of the internal logic. Current serial scan techniques for board level verification would take too much time and resources. The current PAC1000 will only test the ALU and its status flags, the address and block counter, and the sequencer. Future versions in the WS-PAC Family will have even larger sizes of EPROM and may test the control EPROM.

Usage and Limitations

The program is accessible by calling the BIST program. The program occupies forty-five lines of EPROM code. The program can be reduced in size by specifying extra CPU registers to hold the constants h'FFFF', h'0000', h'AAAA', h'5555' and h'FFF4'.

Certain conditions must be met prior to programming the code to ensure that this program will work correctly. The stack should be empty because the program exercises the stack. In addition, location h'3FF' must be reserved because the BIST uses this location to verify the contents of the stack as a '1.' The outputs should be placed in a mode where the existing system is not affected. The 'MAINT' instruction will ensure that the OC is the same throughout the program. However, this example was not implemented in that manner. Instead, it uses set values to assist in debugging the program. Users can do a global substitution of "OUT h'xxxx' " with "MAINT" in their word processor to fully implement this BIST program.

This BIST is not a panacea for system designers. A 'PASS' condition is indicated by a return to the main calling program. The output control will be h'0000'. A 'FAIL' condition will result in some endless loop or jump to some portion of the program. In the event that it does fail after about 170 clock cycles, the system must disable the PAC1000 from the rest of the system in some manner. Future versions of the PAC1000 may include a watchdog timer to interrupt and timeout the BIST.

The variables that can be altered by the user are listed at the beginning of the BIST.mal file. The current values used will only exercise the counters in a simple manner. The user can modify these variables to increase the confidence level of the program at the expense of a longer test cycle.

Usage and Limitations (Cont.)

A summary of the instructions used and the functional blocks follow below:

```
/*********************************************************************/
/* registers destroyed : R0,R1,R2,R3 and R4                          */
/*                       AOR,ACH,ACL,BC,LC and stack                 */
/*                                                                   */
/* stack should be empty before calling this program                 */
/*                                                                   */
/* the block counter, address counter, ALU with register file and    */
/* flags,and the sequencer with stack and counter are tested         */
/*                                                                   */
/* flags checked: BCZ,ACO,CY,Z,O,S,and STKF                          */
/* ALU instructions used: ADC,AND,ADD,MOV,NOP,SHRR,SHRL,SUB          */
/* CONTROL instructions used: ACSIZE,CONT,JMPNC,JMPC,LDLCD,          */
/*              LOOPNZ,PLDLC,POP,RET,RNC,RSTCON and SETCON           */
/*                                                                   */
/* DATA from EPROM used: 0000, FFFF, FFF4, AAAA, 5555 ,0008 , 0010,  */
/*              03FF, 0019                                           */
/*********************************************************************/
```

Confidence Level

The program executes some of the possible internal critical paths of the PAC1000. From tester and simulation measurements, the test of condition codes and branching were consistently the longest. Similarly for the ALU, flag generation such as adding with a carryout is considered a critical path. The counters have a critical path in propagating the carry. Overall, the confidence level of this test is considered to be high.

Analysis of the Program

The currently executing program calls the BIST program by using the 'CALL' instruction. The instruction following 'CALL' which is the return address is pushed to the stack and is not destroyed by the BIST program. See Figure 1 for the BIST flowchart. The BIST tests the PAC1000 functional blocks in the following order:

1. Block Counter and flag BCZ.
2. Address Counter and flag ACO.
3. ALU with shifter and flags CY, Z, O and S.
4. Sequencer with stack and loop counter, and flag STKF.

Some subtleties of programming the PAC1000 are presented. In the ALU section, certain flags must be forced to zero before being tested upon, unlike the normal microprocessors where the individual flags are set and reset by instructions. The ALU result of each cycle updates each flag on the next rising edge of the clock. For example, to check the zero flag (Z), some ALU instruction forces the Z flag to zero. See the instructions below:

```
       MOV R2 R2 , OUT h'0138' ;
       /* force zero flag Z=0 */
zero:  JMPNC Z zero, AND AOR R1 ,
       OUT h'0139' ;
```

Next, loading the loop counter from the ALU needs special treatment. The data must be present at the ALU output before the instruction to load the loop counter executes. See the instructions below:

```
MOV R4 short, OUT h'014B' ;
    /* force ALU output to the
    value of short = h'0010' */
LDLCD , MOV R4 R4, OUT h'014C' ;
    /* load 0010 to LC */
```

Analysis of the Simulation Output

Looking at the block counter outputs BC(15:0) from cycle 7 through 18, the counter counts continuously until disabled. The block counter contents wraps around from h'0000' to h'ffff' and down. Note that the BCZ flag remains latched until new data is loaded to the block counter. Because of the latched flag BCZ, there is a minimum of two cycles before the next instruction is executed after the loop. Figure 2 shows the loop with the minimum number of latency cycles before executing the next line of program code.

Figure 1. Built-In-Self-Test Flowchart

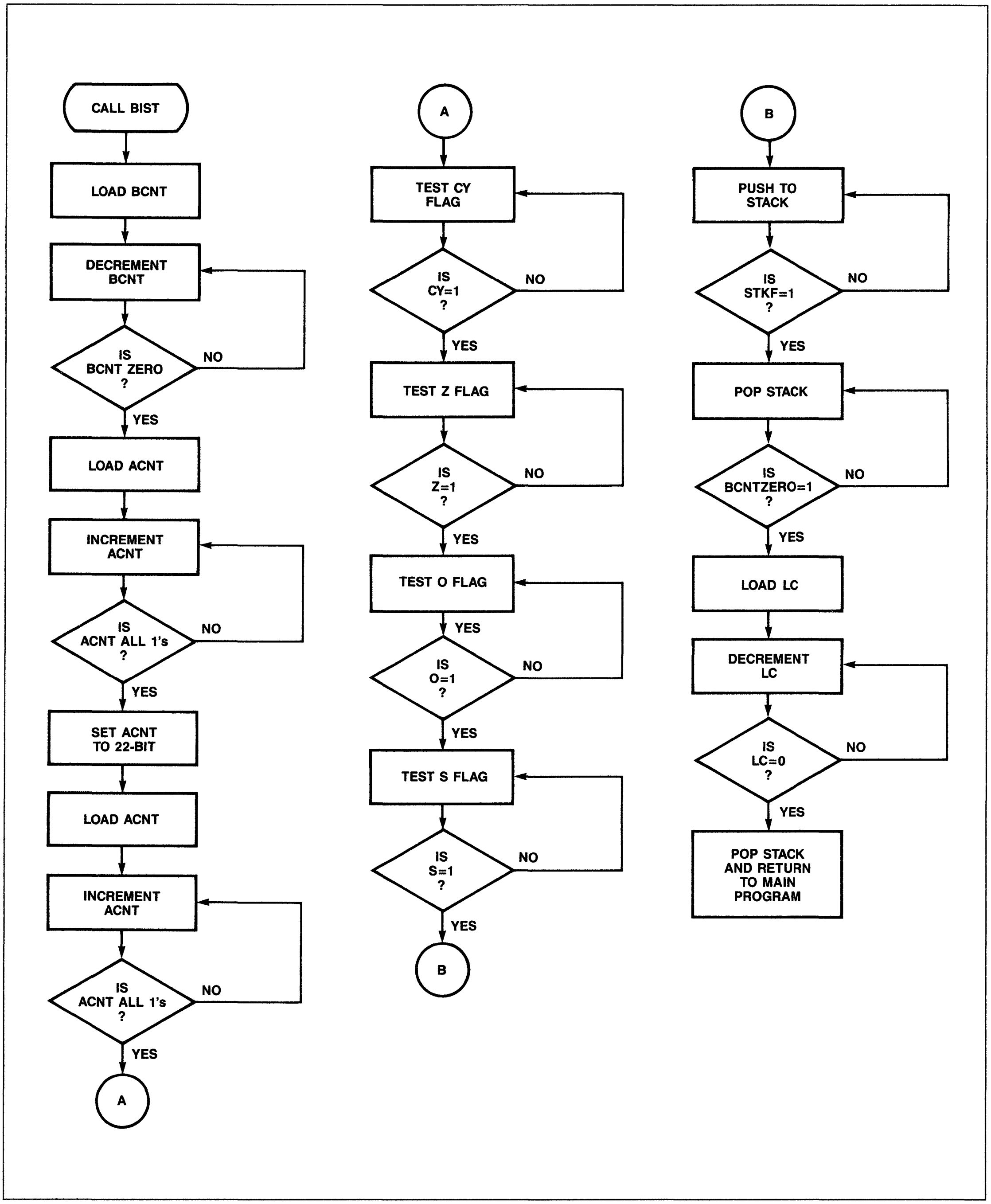
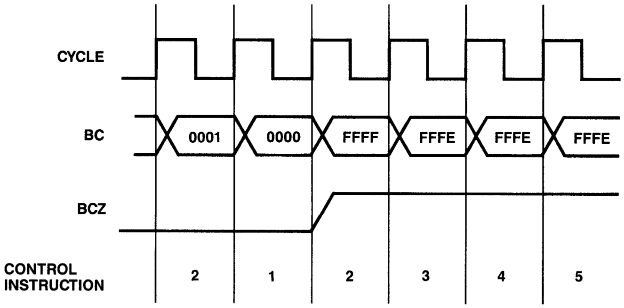


Figure 2. BCZ Flag: Example Cycle-by-Cycle Simulation



1. loop1: MOV R2 h‘5555’, OUT h‘0128’ ;
2. JMPNC BCZ loop1, OUT h‘0129’ ;
3. RESET BCEN, OUT h‘012a’ ;
4. ACSIZE 22, OUT h‘012b’ ;
5. MOV ACL long, OUT h‘012c’ ;

```
/*********************************************/
/* Main calling program        02/03/89      */
/* David Fong                  Rev. 1.0      */
/* main.mal                                  */
/*********************************************/

segment main ;

external bist ;

main1:

/* initialize */
/* not needed */

/* call bist program */

          CALL bist , OUT h’0123’ ;     /* call the BIST program */
          /* return to main program */
FORE:     JMP FORE , OUT h’0000’ ; /* loop forever */

end ;

/**********************************************/
/* Program to jump back to main bist program */
/* David A. Fong 02/03/89       Rev. 1.0      */
/* jmpf.mal                                   */
/**********************************************/

segment jmp ;
external jmpf ;

JMP jmpf , OUT h’FFFF’ ; /* jmpf is an external address */
/* this tests branching with all 1’s */
end ;
```

```
/*****************************************************************/
/* Built-In-Self-Test Program  02/03/89                          */
/* David A. Fong               Rev. 1.0                          */
/* bist.mal                                                      */
/*****************************************************************/
/* registers destroyed : R0,R1,R2,R3 and R4                      */
/*                       AOR,ACH,ACL,BC,LC and stack             */
/*                                                               */
/* stack should be empty before calling this program             */
/*                                                               */
/* the block counter, address counter, ALU with register file and*/
/* flags,and the sequencer with stack and counter are tested     */
/*                                                               */
/* flags checked: BCZ,ACO,CY,Z,O,S,and STKF                      */
/* ALU instructions used: ADC,AND,ADD,MOV,NOP,SHRR,SHRL,SUB      */
/* CONTROL instructions used: ACSIZE,CONT,JMPNC,JMPC,LDLCD,      */
/*             LOOPNZ,PLDLC,POP,RET,RNC,RSTCON and SETCON        */
/*                                                               */
/*DATA from EPROM used: 0000, FFFF, FFF4, AAAA, 5555 ,0008 , 0010*/
/*             03FF, 0019                                        */
/*****************************************************************/


segment c_bist ;
entry bist,jmpf ;   /* entry points into this program */

/* define equates for user to substitute */
shorter equ h'0008' ;
short     equ h'0010' ;
medium    equ h'03ff' ;
long      equ h'fff4' ;
popper    equ h'0019' ;

/****************************/
/* test the counters and    */
/* initialize the registers */
/****************************/

bist:     MOV R1 h'0000', OUT h'0124'; /*the outputs should be placed*/
                                   /* in a non-functional mode */
          MOV R0 h'FFFF' , OUT h'0125' ; /* in this program it is not*/
          MOV BC shorter , OUT h'0126' ;/*because it was needed to*/
          SETCON h'002' , OUT h'0127' ; /*debug enable block counter */
loop1:    MOV R2 h'5555' , OUT h'0128' ;
          JMPNC BCZ loop1 , OUT h'0129' ;
RSTCON h'002' , OUT h'012A' ; /* disable block counter */

/* R0 = FFFF ; R1 = 0000 ; R2 = 5555 */

/* test the 22-bit address counter */

          ACSIZE 22 , MOV ACH R0 , OUT h'012B' ;
          MOV ACL long , OUT h'012C' ;
          SETCON h'001' , OUT h'012D' ; /* enable address counter */
```

```
loop2:    MOV R3 h'AAAA' , OUT h'012E' ;
          JMPNC ACO loop2 , OUT h'012F' ;
          RSTCON h'001' , OUT h'0130' ; /* disable address counter */

/* R0 = FFFF ; R1 = 0000 ; R2 = 5555 ; R3 = AAAA */

/* test the 16-bit address counter */

          ACSIZE 16 , OUT h'0131' ;
          MOV ACH long , OUT h'0132' ;
          SETCON h'001' , OUT h'0133' ; /* enable address counter */
loop3:    MOV R4 h'0000' , OUT h'0134' ;
          JMPNC ACO loop3 , OUT h'0135' ;
          RSTCON h'001' , MOV R3 R3 , OUT h'0136' ;
          /* disable address counter */
/* and do a dummy ALU instruction so that Z=0 and CY=0 */
/* note: a NOP instruction will force Z=1 and CY=1 on the */
/* following cycle*/

/* R0 = FFFF ; R1 = 0000 ; R2 = 5555 ; R3 = AAAA ; R4 = 0000 */
/* R4 is the working register */

/****************/
/* test the ALU */
/****************/

carry:    JMPNC CY carry , ADC AOR R0 , OUT h'0137' ;/*test carryout */

          MOV R2 R2 , OUT h'0138' ;     /* force zero flag = 0 */
zero:     JMPNC Z zero , AND AOR R1 , OUT h'0139' ;/*test all the alu*/
                         /* outputs are zero */

over:     SUB AOR R3 R2 , OUT h'013A' ; /* test for overflow */
          JMPNC O over , OUT h'013B' ;  /* test for overflow */

f15:      JMPNC S f15 , ADD AOR R1 R0 , OUT h'013C' ;/*test sign bit*/

/* test the alu shifting */

shftl:    SHLR R2 Z , OUT h'013D' ;
          AND AOR R3 R2 , OUT h'013E' ; /*should not loop*/
                                        /*but fall-thru */
          JMPC Z shftl , OUT h'013F' ;

shftr:    SHRR R2 Z , OUT h'0140' ;
          AND AOR R3 R2 , OUT h'0141' ;
               /* should not loop,but fall-thru */
          JMPNC Z shftr , OUT h'0142' ;

/**********************/
/* test the sequencer */
/**********************/

          MOV BC short , OUT h'0143' ;
          SETCON h'002' , OUT h'0144' ; /* enable block counter */
stack:    PLDLC medium , OUT h'0145' ;
          JMPNC STKF stack , OUT h'0146';
          /*exit loop when stack is full */
                    /* the return address will not be */
                    /* overwritten , only the top of stack*/
          MOV BC popper , OUT h'0147' ;
jmpf:     RNC BCZ , OUT h'0148' ;
                    /*should come out of loop when empty+1*/
                    /* which is the return address */
          POP , NOP , OUT h'0149' ;
          /* pop one more time but don't pop */
          /* the last return address */
          RSTCON h'002' , OUT h'014A' ; /* disable block counter */

     /* test the loop counter */
          MOV R4 short , OUT h'014B' ;
          LDLCD , MOV R4 R4 , OUT h'014C' ;/* load 16 into the LC*/
lp:       ADC AOR R4 , OUT h'014D' ;    /* aor = aor + r4 */
          LOOPNZ lp , OUT h'014E';/*check that loop count is not zero*/

          RET , OUT h'014F' ;      /* return to calling program */

end ;


/**********************************/
/* bist linker file   02/03/89    */
/* David Fong         Rev. 1.0    */
/* exbist.ml                      */
/**********************************/

place main , c_bist , jmp ;   /* place the segments  */
load main , bist , jmpf ;     /* load the .mal files */

locate main , h'000' ;   /* locate main and init file */
locate c_bist , h'011' ; /* locate bist file after interrupt */
locate jmp , h'3ff' ;    /* locate jmp at 3ff to test '1' from stack */

end ;


.T
        RCCCCCCCCIIIIIIIIIIIICWRHHHHHHHHHHHHHHHHHHHHHHAAAAAAAAAAAAAAAA
        ECCCCCCCCOOOOOOOONNNNSRDDDDDDDDDDDDDDDDDAAAAAADDDDDDDDDDDDDDDD
        S7654321076543210TTTTBBB1111119876543210DDDDDDDDDDDDDDDDDDDDDD
        E                3210   543210          5432101111119876543210
        T                                             543210
        B
TIME
1       00000000000000000000011100000000000000000000000000000000000000
2       10000000000000000000011100000000000000000000000000000000000000
```

```
******************************************************************

                      O U T P U T   T A B L E

P A C S I M    Ver. 1.09                Mon Feb 13 15:12:09 198
************************************** *************************

        PPP OOOO LLL AAAA BBBB AAAA AA BASCOSZ RRRR RRRR RRRR RRRR
        CCC CCCC CCC OOOO CCCC CCCC CC CCTY    3333 2222 1111 0000
        173 1173 173 RRRR 1173 HHHH LL ZOK
        1:: 51:: 1:: 1173 51:: 1173 53   F     1173 1173 1173 1173
        :40 ::40 :40 51:: ::40 51:: ::         51:: 51:: 51:: 51::
        8   18   8   ::40 18   ::40 40         ::40 ::40 ::40 ::40
            2        18   2    18              18   18   18   18
                     2         2               2    2    2    2
   TIME
      1 000 0000 000 0000 0000 0000 00 0010000 0000 0000 0000 0000
      2 000 0000 000 0000 0000 0000 00 1010000 0000 0000 0000 0000
      3 011 0123 000 0000 0000 0000 00 1000001 0000 0000 0000 0000
      4 012 0124 000 0000 0000 0000 00 1001001 0000 0000 0000 0000
      5 013 0125 000 0000 0000 0000 00 1000001 0000 0000 0000 0000
      6 014 0126 000 0000 0000 0000 00 1000010 0000 0000 0000 ffff
      7 015 0127 000 0000 0008 0000 00 0000000 0000 0000 0000 ffff
      8 016 0128 000 0000 0007 0000 00 0001001 0000 0000 0000 ffff
      9 015 0129 000 0000 0006 0000 00 0000000 0000 5555 0000 ffff
     10 016 0128 000 0000 0005 0000 00 0001001 0000 5555 0000 ffff
     11 015 0129 000 0000 0004 0000 00 0000000 0000 5555 0000 ffff
     12 016 0128 000 0000 0003 0000 00 0001001 0000 5555 0000 ffff
     13 015 0129 000 0000 0002 0000 00 0000000 0000 5555 0000 ffff
     14 016 0128 000 0000 0001 0000 00 0001001 0000 5555 0000 ffff
     15 015 0129 000 0000 0000 0000 00 0000000 0000 5555 0000 ffff
     16 016 0128 000 0000 ffff 0000 00 1001001 0000 5555 0000 ffff
     17 017 0129 000 0000 fffe 0000 00 1000000 0000 5555 0000 ffff
     18 018 012a 000 0000 fffd 0000 00 1001001 0000 5555 0000 ffff
     19 019 012b 000 0000 fffd 0000 00 1001001 0000 5555 0000 ffff
     20 01a 012c 000 0000 fffd ffff 00 1000010 0000 5555 0000 ffff
     21 01b 012d 000 0000 fffd ffff 34 1000010 0000 5555 0000 ffff
     22 01c 012e 000 0000 fffd ffff 35 1001001 0000 5555 0000 ffff
     23 01b 012f 000 0000 fffd ffff 36 1000010 aaaa 5555 0000 ffff
     24 01c 012e 000 0000 fffd ffff 37 1001001 aaaa 5555 0000 ffff
     25 01b 012f 000 0000 fffd ffff 38 1000010 aaaa 5555 0000 ffff
     26 01c 012e 000 0000 fffd ffff 39 1001001 aaaa 5555 0000 ffff
     27 01b 012f 000 0000 fffd ffff 3a 1000010 aaaa 5555 0000 ffff
     28 01c 012e 000 0000 fffd ffff 3b 1001001 aaaa 5555 0000 ffff
     29 01b 012f 000 0000 fffd ffff 3c 1000010 aaaa 5555 0000 ffff
     30 01c 012e 000 0000 fffd ffff 3d 1001001 aaaa 5555 0000 ffff
     31 01b 012f 000 0000 fffd ffff 3e 1000010 aaaa 5555 0000 ffff
     32 01c 012e 000 0000 fffd ffff 3f 1001001 aaaa 5555 0000 ffff
     33 01d 012f 000 0000 fffd 0000 00 1100010 aaaa 5555 0000 ffff
     34 01e 0130 000 0000 fffd 0000 01 1101001 aaaa 5555 0000 ffff
     35 01f 0131 000 0000 fffd 0000 01 1101001 aaaa 5555 0000 ffff
     36 020 0132 000 0000 fffd 0000 01 1101001 aaaa 5555 0000 ffff
     37 021 0133 000 0000 fffd fff4 01 1000010 aaaa 5555 0000 ffff
```

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Note 009

In-Circuit Debugging for the PAC1000 User-Configurable Microcontroller

By David Fong

Abstract

This Application Note is used to illustrate the in-circuit debugging capabilities of the PAC1000 user-configurable microcontroller.

Introduction

With the increasing densities and complexities of integrated circuits, the usage of tools such as in-circuit debuggers and emulators is greatly desired by the heroic hardware designer. The PAC1000 supports the usage of in-circuit debuggers.

A review of BP (breakpoint) and SS (single step) is discussed. SS is the method of stepping through the program code one instruction at a time through manual means. In the case of the PAC1000, there is no manual means with a single-step switch. Instead, an interrupt which is set internally through the program is set. This interrupt can then call upon an ISR (interrupt service routine). This subroutine then dumps out the contents of all the possible registers that can be read out. These registers must then be written into the system memory by the user to use in his monitor program. SS is useful for checking that every cycle is executing correctly.

On the other hand, BP is the method of interrupting the program at a specific program location. This allows the program in the PAC1000 to run in real-time system conditions. This breakpoint is passed to the PAC1000 through the FIFO instead of having a fixed address through the program. BP is useful for intermittently checking the execution of the program.

There is no preference on which method is the best. Generally, it is determined by the situation. If the system designer doesn't trust their own system in the beginning of debug, then they will use SS. After the system becomes more debugged, breakpoint is needed occasionally.

Usage and Limitations

Either SS or BP interrupts can occur. Because both use the same initial ISR, the ISR will differentiate between the two by testing for a specific data pattern that accompanies the breakpoint/single-step data through the FIFO. One way was to test for a specific external condition code but that was determined to be inflexible since a specific condition code needed to be dedicated for this task. Instead, two words are written into the CPU registers. These two registers must be reserved for breakpoint/single-step operation. In this example, R0 and R1 are reserved. Register R1 is the mask that is 'AND'd with R0 which is written from the FIFO to produce the Z (zero) flag that is tested. See Figure 1 for the data format that is written into the FIFO and CPU register R0.

The BP state continues with its program by reading out the contents of some registers to the host interface bus. Note the usage of the FIFO to read out the contents of the register to the ADD bus. BP reads out only the input and output registers that can be read as source to the HD bus. Whereas, SS reads out the CPU registers as well as the input and output registers to ADD.

Not all the registers can be read out or if at all with difficulty. CPU registers as was illustrated by this program was read out using the FIFO. However, the user could have individually read out each register. Unfortunately, there would have been a lot of overhead program space taken. The stack cannot be read out because the contents of the stack would affect the program flow. The interrupt mask register and interrupt pending register cannot be read out or to the CPU. Future PAC1000 versions may support extra functions to allow the user to more easily access the internal registers.

In summary, the single-step program dumps out the following registers to the ADD bus: CPU registers R31–R0, DIR, AIR, ACH, ACL, IIR, and BC. Whereas, the breakpoint program dumps out the tollowing registers to the HD bus: DIR, AIR, AOR, ACH, ACL, IIR, and BC.

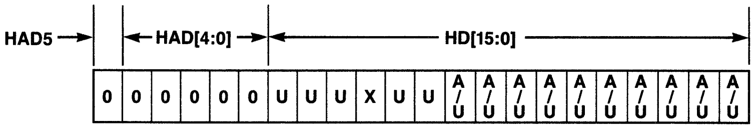
Analysis of Program

This single program incorporates essentially two programs. One for breakpoint and one for single-step. To differentiate between the two programs since they use the same interrupt INT6, the data in register R0 is tested upon and the corresponding action is taken. If Z is true, then breakpoint will occur, else single-step will occur. See BREAKPOINT/SINGLE-STEP algorithm Figure 2.

Note that the Interrupt Jump Table is located at h'008' through h'00f'. The PAC1000 interrupt vector from the internal interrupt jumps to these individual locations. In addition, note that neither conditional nor unconditional jumps were allowed to be executed when either the breakpoint or the single-step interrupts occurred. This also applies to other interrupts. The delay interval from the time of interrupt to executing the interrupt is two cycles. See Figure 3 for the timing relationship of interrupt to the beginning of execution of the interrupt service routine (ISR).

The single-step subroutine utilizes the FIFO to externally address the CPU dual-port registers. The usage of the FIFO in conjunction with loops reduces the size of the control store. However, the contents of the FIFO must be empty before using it.

Figure 1. Host to PAC1000 Commands (Via the FIFO)



0	0	0	0	0	0	U	U	U	X	U	U	A/U	A/U	A/U	A/U	A/U	A/U	A/U	A/U	A/U	A/U

LEGEND:

U: User-Defined
X: Test Bit
A/U: Breakpoint Address or User-Defined

HAD5 = FICD: The flag to indicate that the contents are data FICD=0 or a command FICD=1.

HAD[4:0] = The B address to the CPU register file which in this case is register R0.

HD[15:13] = User-defined.

HD[12] = Test bit to differentiate between breakpoint and single-step. HD12=0 for breakpoint and HD12=1 for single-step.

HD[11:10] = User-defined.

HD[9:0] = Breakpoint Address or for single-step user-defined.

Figure 2. Breakpoint/ Single-Step Flowchart

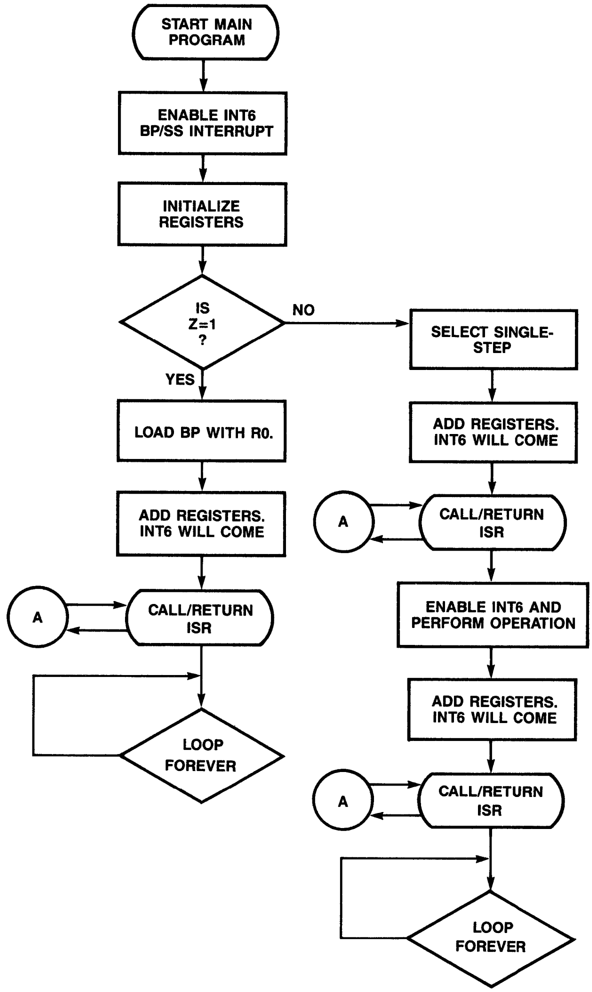


Figure 2. Breakpoint/ Single-Step Flowchart (Cont.)

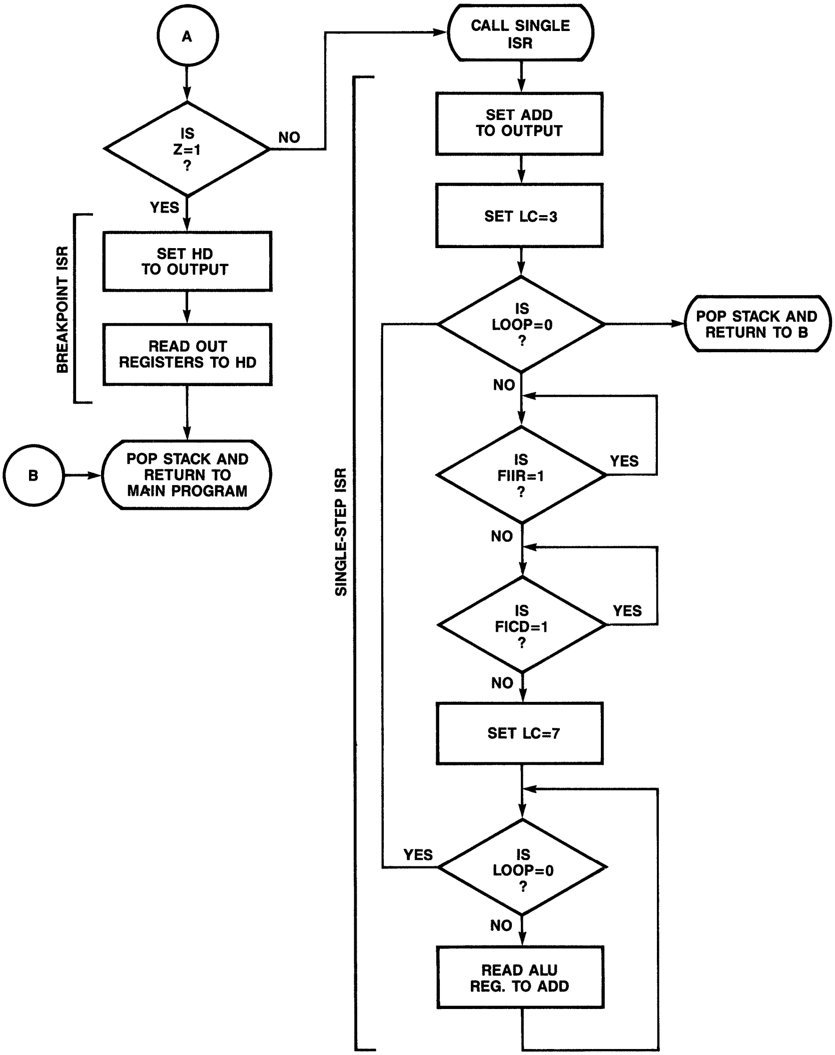
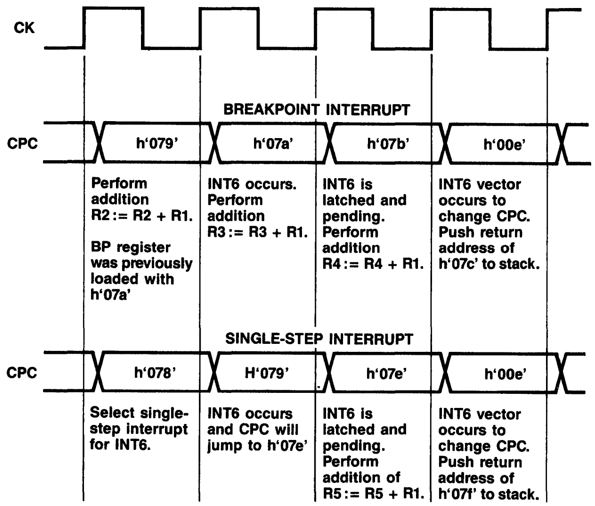


Figure 3. Sequence of Events for Interrupt Timing


CK
BREAKPOINT INTERRUPT
CPC
h'079'
h'07a'
h'07b'
h'00e'
Perform addition R2 := R2 + R1.
BP register was previously loaded with h'07a'
INT6 occurs. Perform addition R3 := R3 + R1.
INT6 is latched and pending. Perform addition R4 := R4 + R1.
INT6 vector occurs to change CPC. Push return address of h'07c' to stack.
SINGLE-STEP INTERRUPT
CPC
h'078'
H'079'
h'07e'
h'00e'
Select single-step interrupt for INT6.
INT6 occurs and CPC will jump to h'07e'
INT6 is latched and pending. Perform addition of R5 := R5 + R1.
INT6 vector occurs to change CPC. Push return address of h'07f' to stack.

Note: CPC is the name from the simulator PACSIM for currently executing program counter.

```
/******************************************/
/* BP and SS linker file      04/03/89   */
/* David Fong             Rev. 1.0       */
/* bpss.ml                               */
/******************************************/

place main, int, intserv, init, single ;     /* place the segments */
load main, int, intserv, init, single ; /* load the .mal files */

locate init , h'000' ;         /* locate the init file */
locate intserv , h'008' ;      /* locate the interrupt vectors */
locate main , h'050' ;         /* locate the main file */
locate int , h'100' ;          /* locate the ISR */
locate single , h'200' ; /* locate the single files */

end ;
```

```
/**********************************/
/* INITIALIZATION   04/03/89     */
/* David Fong       Rev. 1.0     */
/* init.mal                      */
/**********************************/

segment init ;

external main1 ;

SETMODE h'001' , OUT h'0002' ;     /* switch to interrupt mode */
ENABLE INT6 , OUT h'0001' ;
JMP main1 , OUT h'0000' ; /* jump to main program */

end ;
```

```
/**********************************/
/* Main program          04/03/89 */
/* David Fong       Rev. 1.0      */
/* main.mal                       */
/**********************************/

segment main ;

entry main1 ;

main1 :

/**********************************************/
/* BEGIN MAIN PROGRAM                         */
/**********************************************/

/* initialize registers */

R1 := h'1000' , OUT h'0050' ;/* the twelveth bit R1.12 tests for BP/SS*/
/* IF Z=1 (which means R1.12 = 0 ) THEN run breakpoint program */
/* ELSE run single-step program */
R2 := h'0002' , OUT h'0051' ;
R3 := h'0003' , OUT h'0052' ;
R4 := h'0004' , OUT h'0053' ;
R5 := h'0005' , OUT h'0054' ;
R6 := h'0006' , OUT h'0055' ;
R7 := h'0007' , OUT h'0056' ;
R8 := h'0008' , OUT h'0057' ;
R9 := h'0009' , OUT h'0058' ;
R10 := h'000a' , OUT h'0059' ;
R11 := h'000b' , OUT h'005a' ;
R12 := h'000c' , OUT h'005b' ;
R13 := h'000d' , OUT h'005c' ;
R14 := h'000e' , OUT h'005d' ;
R15 := h'000f' , OUT h'005e' ;
R16 := h'0010' , OUT h'005f' ;
R17 := h'0011' , OUT h'0060' ;
R18 := h'0012' , OUT h'0061' ;
R19 := h'0013' , OUT h'0062' ;
R20 := h'0014' , OUT h'0063' ;
R21 := h'0015' , OUT h'0064' ;
R22 := h'0016' , OUT h'0065' ;
R23 := h'0017' , OUT h'0066' ;
R24 := h'0018' , OUT h'0067' ;
R25 := h'0019' , OUT h'0068' ;
R26 := h'001a' , OUT h'0069' ;
R27 := h'001b' , OUT h'006a' ;
R28 := h'001c' , OUT h'006b' ;
R29 := h'001d' , OUT h'006c' ;
R30 := h'001e' , OUT h'006d' ;
R31 := h'001f' , OUT h'006e' ;

ACH := R31 , OUT h'006f' ;
ACL := R0 , OUT h'0070' ;
AOR := R1 , OUT h'0071' ;
DOR := R15 , OUT h'0072' ;
BC  := R7 , OUT h'0073' ;
```

```
/* all input registers are initialized to zero from RESET */

/* to integrate two different programs 1. BREAKPOINT 2. SINGLE-STEP*/
/* The result of masking R0 with R1 is used to differentiate */
/* between BP and SS. */
/* IF Z = 1 Breakpoint; ELSE Z = 0 Single-Step */

/***************** READ IN FIFO AND TEST FOR BP/SS ********************/
g0:  JMPC FICD g0 , OUT h'0074' ; /*check that the fifo contents is data
LDBPD , RDFIFO , OUT h'0075'; /* FIFO was loaded with h'0 00 007a' */
/* first 0 is FICD ; 00 is B address ; 0 is the test bit ; */
/* 07a is the EPROM breakpoint address. */
/* Load loop counter with same data read from FIFO : LDLCD; */
/* the data written into the CPU is the same as the CPU output bus */

AND R1 R0 , OUT h'0076' ; /* the Z flag is tested in the next cycle */
JMPC Z b0 , OUT h'0077' ;
/* select single-step interrupt */
ESS , OUT h'0078' ;
JMP c0 , OUT h'0079' ; /* skip breakpoint routine */

/**************** BREAKPOINT ***************************************/
/* perform alu operations till interrupt comes */
b0:  R2 := R2 + R1 , OUT h'007a' ; /* breakpoint on this address h'07a'
R3 := R3 + R1 , OUT h'007b' ;
R4 := R4 + R1 , OUT h'007c' ; /* breakpoint interrupt comes here */
/* return from ISR to here */
e0 : JMP e0 , OUT h'007d' ;   /* loop forever ; end of breakpoint */


/*************** SINGLE-STEP *************************************/
c0:  R5 := R5 + R1 , OUT h'007e' ; /* execute till interrupt comes */
R6 := R6 + R1 , OUT h'007f' ; /* interrupt should after here */

/* return from single-step ISR to here */
/* enable single-step interrupt and perform an operation */
ENABLE INT6 , R7 := R7 + R1 , OUT h'0080' ; /* the output for R2 */
                         /* should be h'1002' */
R8 := R8 + R1 , OUT h'0081' ; /* interrupt should come here */
/* return from single-step ISR to here */

f0 : JMP f0 , OUT h'0082' ;   /* loop forever */
end ;


/*********************************/
/*SINGLE-STEP SUBROUTINE 04/03/89*/
/* David Fong      Rev. 1.0     */
/* single.mal                   */
/*********************************/

segment single ;
entry single1 ;

single1 :
     /* read out the registers from the ALU */
     /* use the addressing scheme from the FIFO */
     SETCON h'010' , OUT h'2000' ; /* set ADD bus to output */
     /* to read out AOR to ADD */

     /* loop four times to address the 32 registers */

     FOR 3 , OUT h'2001' ;

     /* FIFO should already be full */

f0 : JMPC FIIR f0 , OUT h'2002' ;  /* loop till FIFO is full*/

     /* check that the first value in the FIFO is a data */
f1 : JMPC FICD f1 , OUT h'2003' ;

     /* loop eight times to empty the FIFO */
          FOR 7 , OUT h'2004' ;

     /* use the FIFO as an address pointer */
     /* the data is not needed; write the data back to CPU */
     /* and output the CPU output to AOR */
     /* the default CPU instruction is add which adds zero and */
     /* the address pointed by the FIFO which is the B address */

          RDFIFO , alu_src = zb , ybus_sel = y_aoreg ,
          OUT h'2005' ;
          ENDFOR , OUT h'2006' ;

     ENDFOR , OUT h'2007' ;

     /* read out the source registers to ADD */
     MOV AOR DIR , OUT h'2008' ; /* 0000 should come out next cyle */
     MOV AOR AIR , OUT h'2009' ; /* 0000 */
     MOV AOR ACH , OUT h'200a' ; /* 001f */
     MOV AOR ACL , OUT h'200b' ; /* 0000 */
     MOV AOR IIR , OUT h'200c' ; /* 0000 */
     MOV AOR BC  , OUT h'200d' ; /* 0007 */

     RET , OUT h'200e' ; /* return to ISR 6 */

end ;


/*********************************/
/* INTERRUPT JUMP TABLE  04/03/89*/
/* David Fong       Rev. 1.0     */
/* intserv.mal                   */
/*********************************/

segment intserv ;
entry int_serv ;
external int0,int1,int2,int3,int4,int5,int6,int7 ;

int_serv :
     JMP int0 , OUT h'0008' ;
     JMP int1 , OUT h'0009' ;
     JMP int2 , OUT h'000a' ;
     JMP int3 , OUT h'000b' ;
     JMP int4 , OUT h'000c' ;
     JMP int5 , OUT h'000d' ;
     JMP int6 , OUT h'000e' ;
     JMP int7 , OUT h'000f' ;

end ;


/*********************************************************/
/* Interrupt Service Routines      04/03/89             */
/* David Fong                 Rev. 1.0                  */
/* int.mal                                              */
/*********************************************************/

segment int ;
entry int0 , int1 , int2 , int3 , int4 , int5 , int6 , int7 ;
external single1 ;

int0 :
     /* clear all the external interrupts */
     CLI h'00f' , OUT h'0100' ;
     RET , OUT h'0101' ;

int1 :
     /* clear all the external interrupts */
     CLI h'00f' , OUT h'0102' ;
     RET , OUT h'0103' ;

int2 :
     /* clear all the external interrupts */
     CLI h'00f' , OUT h'0104' ;
     RET , OUT h'0105' ;

int3 :
     /* clear all the external interrupts */
     CLI h'00f' , OUT h'0106' ;
     RET , OUT h'0107' ;

int4 :
     /* mask that interrupt */
     DISABLE INT4 , OUT h'0108' ;
     RET , OUT h'0109' ;

int5 :
     /* mask that interrupt */
     DISABLE INT5 , OUT h'010a' ;
     RET , OUT h'010b' ;

int6 :  /* Breakpoint and Single-step ISR */
     /* mask that interrupt */
     DISABLE INT6 , OUT h'010c' ;  /* mask interrupt 6 INT6 */
     CLI h'0ff' , OUT h'010d' ;    /* clear all interrupts */

/************** TEST for Breakpoint/Single-Step **************/
      AND R1 R0 , OUT h'010e' ;
      JMPC Z a0 , OUT h'010f' ; /* if Z=1 then breakpoint,Z=0 SS */
     CALL single1 , OUT h'0110' ;/* call single step program */
     JMP b0 , OUT h'0111' ; /*finish SS ISR , return to main progr */

a0:  SET HDOE HDSEL0 , OUT h'0112' ; /* set HD to output */
                           /* select DOR to HD output bus*/


     /* move out the source registers to HD */

     MOV DOR DIR , OUT h'0113' ; /* 0000 should come out next cycle*/
     MOV DOR AIR , OUT h'0114' ; /* 0000 */
     MOV DOR AOR , OUT h'0115' ; /* 0001 */
     MOV DOR ACH , OUT h'0116' ; /* 001f */
     MOV DOR ACL , OUT h'0117' ; /* 0000 */
     MOV DOR IIR , OUT h'0118' ; /* 0000 */
     MOV DOR BC  , OUT h'0119' ; /* 0007 */

b0:
     RET , OUT h'011a' ;
int7 :
     /* mask that interrupt */
     DISABLE INT7 , OUT h'011a' ;
     RET , OUT h'011b' ;
end ;


.T
        RCCCCCCCCIIIIIIIIIIIICWRHHHHHHHHHHHHHHHHHHHHHHAAAAAAAAAAAAAAAA
        ECCCCCCCCOOOOOOOONNNNSRDDDDDDDDDDDDDDDDDAAAAAADDDDDDDDDDDDDDDD
        S7654321076543210TTTTBBB1111119876543210DDDDDDDDDDDDDDDDDDDDDD
        E                   3210   543210       5432101111119876543210
        T                                             543210
        B
TIME
    1   00000000000000000000011100000000000000000000000000000000000000
    2   10000000000000000000011100000000000000000000000000000000000000
# bps0.stl file for single-stepping
# write the single-step mode bit hd12=1
   20   10000000000000000000000100010000000000000000000000000000000000
   21   10000000000000000000011100010000000000000000000000000000000000
# write into FIFO for single-step
   55   1000000000000000000001110000000000000000000000ZZZZZZZZZZZZZZZZ
   56   1000000000000000000000010000000000000000000000ZZZZZZZZZZZZZZZZ
   57   1000000000000000000001110000000000000000000000ZZZZZZZZZZZZZZZZ
   58   1000000000000000000000010000000000000000000001ZZZZZZZZZZZZZZZZ
   59   1000000000000000000001110000000000000000000001ZZZZZZZZZZZZZZZZ
   60   1000000000000000000000010000000000000000000010ZZZZZZZZZZZZZZZZ
   61   1000000000000000000001110000000000000000000010ZZZZZZZZZZZZZZZZ
   62   1000000000000000000000010000000000000000000011ZZZZZZZZZZZZZZZZ
   63   1000000000000000000001110000000000000000000011ZZZZZZZZZZZZZZZZ
   64   1000000000000000000000010000000000000000000100ZZZZZZZZZZZZZZZZ
   65   1000000000000000000001110000000000000000000100ZZZZZZZZZZZZZZZZ
   66   1000000000000000000000010000000000000000000101ZZZZZZZZZZZZZZZZ
   67   1000000000000000000001110000000000000000000101ZZZZZZZZZZZZZZZZ
   68   1000000000000000000000010000000000000000000110ZZZZZZZZZZZZZZZZ
   69   1000000000000000000001110000000000000000000110ZZZZZZZZZZZZZZZZ
   70   1000000000000000000000010000000000000000000111ZZZZZZZZZZZZZZZZ
   71   1000000000000000000001110000000000000000000111ZZZZZZZZZZZZZZZZ
```

```
  95     10000000000000000000000100000000000000000001000ZZZZZZZZZZZZZZZZ
  96     10000000000000000000011100000000000000000001000ZZZZZZZZZZZZZZZZ
  97     10000000000000000000000100000000000000000001001ZZZZZZZZZZZZZZZZ
  98     10000000000000000000011100000000000000000001001ZZZZZZZZZZZZZZZZ
  99     10000000000000000000000100000000000000000001010ZZZZZZZZZZZZZZZZ
 100     10000000000000000000011100000000000000000001010ZZZZZZZZZZZZZZZZ
 101     10000000000000000000000100000000000000000001011ZZZZZZZZZZZZZZZZ
 102     10000000000000000000011100000000000000000001011ZZZZZZZZZZZZZZZZ
 103     10000000000000000000000100000000000000000001100ZZZZZZZZZZZZZZZZ
 104     10000000000000000000011100000000000000000001100ZZZZZZZZZZZZZZZZ
 105     10000000000000000000000100000000000000000001101ZZZZZZZZZZZZZZZZ
 106     10000000000000000000011100000000000000000001101ZZZZZZZZZZZZZZZZ
 107     10000000000000000000000100000000000000000001110ZZZZZZZZZZZZZZZZ
 108     10000000000000000000011100000000000000000001110ZZZZZZZZZZZZZZZZ
 109     10000000000000000000000100000000000000000001111ZZZZZZZZZZZZZZZZ
 110     10000000000000000000011100000000000000000001111ZZZZZZZZZZZZZZZZ
 135     1000000000000000000000010000000000000000010000ZZZZZZZZZZZZZZZZ
 136     1000000000000000000001110000000000000000010000ZZZZZZZZZZZZZZZZ
 137     1000000000000000000000010000000000000000010001ZZZZZZZZZZZZZZZZ
 138     1000000000000000000001110000000000000000010001ZZZZZZZZZZZZZZZZ
 139     1000000000000000000000010000000000000000010010ZZZZZZZZZZZZZZZZ
 140     1000000000000000000001110000000000000000010010ZZZZZZZZZZZZZZZZ
 141     1000000000000000000000010000000000000000010011ZZZZZZZZZZZZZZZZ
 142     1000000000000000000001110000000000000000010011ZZZZZZZZZZZZZZZZ
 143     1000000000000000000000010000000000000000010100ZZZZZZZZZZZZZZZZ
 144     1000000000000000000001110000000000000000010100ZZZZZZZZZZZZZZZZ
 145     1000000000000000000000010000000000000000010101ZZZZZZZZZZZZZZZZ
 146     1000000000000000000001110000000000000000010101ZZZZZZZZZZZZZZZZ
 147     1000000000000000000000010000000000000000010110ZZZZZZZZZZZZZZZZ
 148     1000000000000000000001110000000000000000010110ZZZZZZZZZZZZZZZZ
 149     1000000000000000000000010000000000000000010111ZZZZZZZZZZZZZZZZ
 150     1000000000000000000001110000000000000000010111ZZZZZZZZZZZZZZZZ
 175     1000000000000000000000010000000000000000011000ZZZZZZZZZZZZZZZZ
 176     1000000000000000000001110000000000000000011000ZZZZZZZZZZZZZZZZ
 177     1000000000000000000000010000000000000000011001ZZZZZZZZZZZZZZZZ
 178     1000000000000000000001110000000000000000011001ZZZZZZZZZZZZZZZZ
 179     1000000000000000000000010000000000000000011010ZZZZZZZZZZZZZZZZ
 180     1000000000000000000001110000000000000000011010ZZZZZZZZZZZZZZZZ
 181     1000000000000000000000010000000000000000011011ZZZZZZZZZZZZZZZZ
 182     1000000000000000000001110000000000000000011011ZZZZZZZZZZZZZZZZ
 183     1000000000000000000000010000000000000000011100ZZZZZZZZZZZZZZZZ
 184     1000000000000000000001110000000000000000011100ZZZZZZZZZZZZZZZZ
 185     1000000000000000000000010000000000000000011101ZZZZZZZZZZZZZZZZ
 186     1000000000000000000001110000000000000000011101ZZZZZZZZZZZZZZZZ
 187     1000000000000000000000010000000000000000011110ZZZZZZZZZZZZZZZZ
 188     1000000000000000000001110000000000000000011110ZZZZZZZZZZZZZZZZ
 189     1000000000000000000000010000000000000000011111ZZZZZZZZZZZZZZZZ
 190     1000000000000000000001110000000000000000011111ZZZZZZZZZZZZZZZZ
# write into FIFO second time around for single-step
 240     1000000000000000000001110000000000000000000000ZZZZZZZZZZZZZZZZ
 241     1000000000000000000000010000000000000000000000ZZZZZZZZZZZZZZZZ
 242     1000000000000000000001110000000000000000000000ZZZZZZZZZZZZZZZZ
 243     1000000000000000000000010000000000000000000001ZZZZZZZZZZZZZZZZ
 244     1000000000000000000001110000000000000000000001ZZZZZZZZZZZZZZZZ
 245     1000000000000000000000010000000000000000000010ZZZZZZZZZZZZZZZZ
 246     1000000000000000000001110000000000000000000010ZZZZZZZZZZZZZZZZ
 247     1000000000000000000000010000000000000000000011ZZZZZZZZZZZZZZZZ
```

```
248     100000000000000000000111000000000000000000000011ZZZZZZZZZZZZZZZZ
249     100000000000000000000001000000000000000000000100ZZZZZZZZZZZZZZZZ
250     100000000000000000000111000000000000000000000100ZZZZZZZZZZZZZZZZ
255     100000000000000000000001000000000000000000000101ZZZZZZZZZZZZZZZZ
256     100000000000000000000111000000000000000000000101ZZZZZZZZZZZZZZZZ
257     100000000000000000000001000000000000000000000110ZZZZZZZZZZZZZZZZ
258     100000000000000000000111000000000000000000000110ZZZZZZZZZZZZZZZZ
259     100000000000000000000001000000000000000000000111ZZZZZZZZZZZZZZZZ
260     100000000000000000000111000000000000000000000111ZZZZZZZZZZZZZZZZ
285     100000000000000000000001000000000000000000001000ZZZZZZZZZZZZZZZZ
286     100000000000000000000111000000000000000000001000ZZZZZZZZZZZZZZZZ
287     100000000000000000000001000000000000000000001001ZZZZZZZZZZZZZZZZ
288     100000000000000000000111000000000000000000001001ZZZZZZZZZZZZZZZZ
289     100000000000000000000001000000000000000000001010ZZZZZZZZZZZZZZZZ
290     100000000000000000000111000000000000000000001010ZZZZZZZZZZZZZZZZ
291     100000000000000000000001000000000000000000001011ZZZZZZZZZZZZZZZZ
292     100000000000000000000111000000000000000000001011ZZZZZZZZZZZZZZZZ
293     100000000000000000000001000000000000000000001100ZZZZZZZZZZZZZZZZ
294     100000000000000000000111000000000000000000001100ZZZZZZZZZZZZZZZZ
295     100000000000000000000001000000000000000000001101ZZZZZZZZZZZZZZZZ
296     100000000000000000000111000000000000000000001101ZZZZZZZZZZZZZZZZ
297     100000000000000000000001000000000000000000001110ZZZZZZZZZZZZZZZZ
298     100000000000000000000111000000000000000000001110ZZZZZZZZZZZZZZZZ
299     100000000000000000000001000000000000000000001111ZZZZZZZZZZZZZZZZ
300     100000000000000000000111000000000000000000001111ZZZZZZZZZZZZZZZZ
325     100000000000000000000001000000000000000000010000ZZZZZZZZZZZZZZZZ
326     100000000000000000000111000000000000000000010000ZZZZZZZZZZZZZZZZ
327     100000000000000000000001000000000000000000010001ZZZZZZZZZZZZZZZZ
328     100000000000000000000111000000000000000000010001ZZZZZZZZZZZZZZZZ
329     100000000000000000000001000000000000000000010010ZZZZZZZZZZZZZZZZ
330     100000000000000000000111000000000000000000010010ZZZZZZZZZZZZZZZZ
331     100000000000000000000001000000000000000000010011ZZZZZZZZZZZZZZZZ
332     100000000000000000000111000000000000000000010011ZZZZZZZZZZZZZZZZ
333     100000000000000000000001000000000000000000010100ZZZZZZZZZZZZZZZZ
334     100000000000000000000111000000000000000000010100ZZZZZZZZZZZZZZZZ
335     100000000000000000000001000000000000000000010101ZZZZZZZZZZZZZZZZ
336     100000000000000000000111000000000000000000010101ZZZZZZZZZZZZZZZZ
337     100000000000000000000001000000000000000000010110ZZZZZZZZZZZZZZZZ
338     100000000000000000000111000000000000000000010110ZZZZZZZZZZZZZZZZ
339     100000000000000000000001000000000000000000010111ZZZZZZZZZZZZZZZZ
340     100000000000000000000111000000000000000000010111ZZZZZZZZZZZZZZZZ
365     100000000000000000000001000000000000000000011000ZZZZZZZZZZZZZZZZ
366     100000000000000000000111000000000000000000011000ZZZZZZZZZZZZZZZZ
367     100000000000000000000001000000000000000000011001ZZZZZZZZZZZZZZZZ
368     100000000000000000000111000000000000000000011001ZZZZZZZZZZZZZZZZ
369     100000000000000000000001000000000000000000011010ZZZZZZZZZZZZZZZZ
370     100000000000000000000111000000000000000000011010ZZZZZZZZZZZZZZZZ
371     100000000000000000000001000000000000000000011011ZZZZZZZZZZZZZZZZ
372     100000000000000000000111000000000000000000011011ZZZZZZZZZZZZZZZZ
373     100000000000000000000001000000000000000000011100ZZZZZZZZZZZZZZZZ
374     100000000000000000000111000000000000000000011100ZZZZZZZZZZZZZZZZ
375     100000000000000000000001000000000000000000011101ZZZZZZZZZZZZZZZZ
376     100000000000000000000111000000000000000000011101ZZZZZZZZZZZZZZZZ
377     100000000000000000000001000000000000000000011110ZZZZZZZZZZZZZZZZ
378     100000000000000000000111000000000000000000011110ZZZZZZZZZZZZZZZZ
379     100000000000000000000001000000000000000000011111ZZZZZZZZZZZZZZZZ
380     100000000000000000000111000000000000000000011111ZZZZZZZZZZZZZZZZ
```

```
.T
          RCCCCCCCCIIIIIIIIIIIICWRHHHHHHHHHHHHHHHHAAAAAAAAAAAAAAAAAAAAAA
          ECCCCCCCCOOOOOOOONNNNSRDDDDDDDDDDDDDDDDDAAAAAADDDDDDDDDDDDDDDD
          S7654321076543210TTTTBBB1111119876543210DDDDDDDDDDDDDDDDDDDDDD
          E                3210   543210          5432101111119876543210
          T                                             543210
          B
TIME
 1        00000000000000000000011100000000000000000000000000000000000000
 2        10000000000000000000011100000000000000000000000000000000000000
20        10000000000000000000000100000000011110100000000000000000000000
21        10000000000000000000011100000000011110100000000000000000000000
53        100000000000000000000111ZZZZZZZZZZZZZZZZ0000000000000000000000
# bps1.stl uses Z=1 for breakpoint ISR; HD12=0;
```

```
                    *********The bps0.out file **********

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

                           O U T P U T   T A B L E

   P A C S I M     Ver. 1.09                   Tue Apr 04 15:43:42 1989
*****************************************************************************

        CCC OOOO A AAAA AAAA FFFIB PPP LLL BBBB
        PPP CCCC D DDDD OOOO IIINR CCC CCC CCCC
        CCC 1173 O DDDD RRRR CIOTP 173 173 1173
        173 51:: E 1173 1173 DRRRT 1:: 1:: 51::
        1:: ::40   51:: 51::     E :40 :40 ::40
        :40 18     ::40 ::40     Q 8   8   18
        8   2      18   18       U         2
                   2    2        L
TIME
    1   000 0000 0 0000 0000 00001 000 000 0000
    2   000 0000 0 0000 0000 00001 000 000 0000
    3   000 0002 0 0000 0000 01000 001 000 0000
    4   001 0001 0 0000 0000 01000 002 000 0000
    5   002 0000 0 0000 0000 01000 050 000 0000
    6   050 0050 0 0000 0000 01000 051 000 0000
    7   051 0051 0 0000 0000 01000 052 000 0000
    8   052 0052 0 0000 0000 01000 053 000 0000
    9   053 0053 0 0000 0000 01000 054 000 0000
   10   054 0054 0 0000 0000 01000 055 000 0000
   11   055 0055 0 0000 0000 01000 056 000 0000
   12   056 0056 0 0000 0000 01000 057 000 0000
   13   057 0057 0 0000 0000 01000 058 000 0000
   14   058 0058 0 0000 0000 01000 059 000 0000
   15   059 0059 0 0000 0000 01000 05a 000 0000
   16   05a 005a 0 0000 0000 01000 05b 000 0000
   17   05b 005b 0 0000 0000 01000 05c 000 0000
   18   05c 005c 0 0000 0000 01000 05d 000 0000
   19   05d 005d 0 0000 0000 01000 05e 000 0000
   20   05e 005e 0 0000 0000 01000 05f 000 0000
   21   05f 005f 0 0000 0000 01000 060 000 0000
   22   060 0060 0 0000 0000 01100 061 000 0000
   23   061 0061 0 0000 0000 01100 062 000 0000
```

```
  24  062 0062 0 0000 0000 01100 063 000 0000
  25  063 0063 0 0000 0000 01100 064 000 0000
  26  064 0064 0 0000 0000 01100 065 000 0000
  27  065 0065 0 0000 0000 01100 066 000 0000
  28  066 0066 0 0000 0000 01100 067 000 0000
  29  067 0067 0 0000 0000 01100 068 000 0000
  30  068 0068 0 0000 0000 01100 069 000 0000
  31  069 0069 0 0000 0000 01100 06a 000 0000
  32  06a 006a 0 0000 0000 01100 06b 000 0000
  33  06b 006b 0 0000 0000 01100 06c 000 0000
***Due to the length of the file,the rest of the output is not shown ***
```

```
                  ***********The bps1.out file *************

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

                        O U T P U T   T A B L E

P A C S I M    Ver. 1.09                     Mon Apr 03 13:08:15 1989
**********************************************************************

      CCC OOOO M CC DI BBB B HHHH LLL BBBB
      PPP CCCC D CC ON RRR R DDDD CCC CCCC
      CCC 1173 O 73 RT EEE P 1173 173 1173
      173 51:: E ::  R AAA T 51:: 1:: 51::
      1:: ::40   40    KKK E ::40 :40 ::40
      :40 18           RRR Q 18   8   18
      8   2            EEE U 2        2
                       GGG L
                       973
                       :::
                       840
TIME
   1  000 0000 0 00 00 000 1 0000 000 0000
   2  000 0000 0 00 00 000 1 0000 000 0000
   3  000 0002 0 00 00 000 0 0000 000 0000
   4  001 0001 0 00 00 000 0 0000 000 0000
   5  002 0000 0 00 00 000 0 0000 000 0000
   6  050 0050 0 00 00 000 0 0000 000 0000
   7  051 0051 0 00 00 000 0 0000 000 0000
   8  052 0052 0 00 00 000 0 0000 000 0000
   9  053 0053 0 00 00 000 0 0000 000 0000
  10  054 0054 0 00 00 000 0 0000 000 0000
  11  055 0055 0 00 00 000 0 0000 000 0000
  12  056 0056 0 00 00 000 0 0000 000 0000
  13  057 0057 0 00 00 000 0 0000 000 0000
  14  058 0058 0 00 00 000 0 0000 000 0000
  15  059 0059 0 00 00 000 0 0000 000 0000
  16  05a 005a 0 00 00 000 0 0000 000 0000
  17  05b 005b 0 00 00 000 0 0000 000 0000
  18  05c 005c 0 00 00 000 0 0000 000 0000
  19  05d 005d 0 00 00 000 0 0000 000 0000
  20  05e 005e 0 00 00 000 0 007a 000 0000
  21  05f 005f 0 00 00 000 0 007a 000 0000
  22  060 0060 0 00 00 000 0 007a 000 0000
  23  061 0061 0 00 00 000 0 007a 000 0000
```

```
24  062 0062 0 00 00 000 0 007a 000 0000
25  063 0063 0 00 00 000 0 007a 000 0000
26  064 0064 0 00 00 000 0 007a 000 0000
27  065 0065 0 00 00 000 0 007a 000 0000
28  066 0066 0 00 00 000 0 007a 000 0000
29  067 0067 0 00 00 000 0 007a 000 0000
30  068 0068 0 00 00 000 0 007a 000 0000
31  069 0069 0 00 00 000 0 007a 000 0000
32  06a 006a 0 00 00 000 0 007a 000 0000
33  06b 006b 0 00 00 000 0 007a 000 0000
34  06c 006c 0 00 00 000 0 007a 000 0000
35  06d 006d 0 00 00 000 0 007a 000 0000
36  06e 006e 0 00 00 000 0 007a 000 0000
37  06f 006f 0 00 00 000 0 007a 000 0000
38  070 0070 0 00 00 000 0 007a 000 0000
39  071 0071 0 00 00 000 0 007a 000 0000
40  072 0072 0 00 00 000 0 007a 000 0000
41  073 0073 0 00 00 000 0 007a 000 0000
42  074 0074 0 00 00 000 0 007a 000 0007
43  075 0075 0 00 00 07a 0 007a 000 0007
44  076 0076 0 00 00 07a 0 007a 000 0007
45  077 0077 0 00 00 07a 1 007a 000 0007
46  07a 007a 0 00 00 07a 0 007a 000 0007
47  07b 007b 0 00 01 07a 0 007a 000 0007
48  00e 000e 0 00 01 07a 0 007a 000 0007
49  10c 010c 0 00 00 07a 0 007a 000 0007
50  10d 010d 0 00 00 07a 0 007a 000 0007
51  10e 010e 0 00 00 07a 0 007a 000 0007
52  10f 010f 0 00 00 07a 0 007a 000 0007
53  112 0112 1 00 00 07a 0 000f 000 0007
54  113 0113 1 00 00 07a 0 000f 000 0007
55  114 0114 1 00 00 07a 0 0000 000 0007
56  115 0115 1 00 00 07a 0 0000 000 0007
57  116 0116 1 00 00 07a 0 1000 000 0007
58  117 0117 1 00 00 07a 0 001f 000 0007
59  118 0118 1 00 00 07a 0 0000 000 0007
60  119 0119 1 00 00 07a 0 0000 000 0007
61  11a 011a 1 00 00 07a 0 0007 000 0007
62  07c 007c 1 00 00 07a 0 0007 000 0007
63  07d 007d 1 00 00 07a 0 0007 000 0007
64  07d 007d 1 00 00 07a 0 0007 000 0007
65  07d 007d 1 00 00 07a 0 0007 000 0007
66  07d 007d 1 00 00 07a 0 0007 000 0007
67  07d 007d 1 00 00 07a 0 0007 000 0007
68  07d 007d 1 00 00 07a 0 0007 000 0007
69  07d 007d 1 00 00 07a 0 0007 000 0007
70  07d 007d 1 00 00 07a 0 0007 000 0007
```

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Brief 007

Hardware Interfacing the PAC1000 as a Micro Channel Bus Controller

By Arye Ziklik

Abstract

This application brief describes how to use the PAC1000 High-Performance User-Configurable Microcontroller as a Micro Channel (MCA) bus controller.

The MCA bus uses asynchronous and synchronous procedures to control and transfer data on the bus. The data is transferred from a master board to a slave board, or from the PS/2 mother-board (the system) to a slave. This application brief describes the use of the PAC1000 on a master board and on a slave board.

In both applications the PAC1000 is handling the synchronous functions, the asynchronous functions are implemented by external PALs.

MCA Signal Descriptions

The bus signals described in this chapter are the most important and essential signals to understand the application brief. The buffers needed per each signal are summarized in Table 2. The timing relations between the signals is drawn in Figure 1.

A0-A23

Address bits generated by the bus master to address memory and IO slaves attached to the bus. The address bits are unlatched and must be latched by the slaves using either the trailing edge of $\overline{\text{ADL}}$ or the leading edge of $\overline{\text{CMD}}$ signals.

D0-D15

Data bits, valid during the period $\overline{\text{CMD}}$ signal is low. The data is driven by bidirectional three-state drivers.

$\overline{\text{ADL}}$

Address Decode Latch, driven by the bus master. The signal is used by the slaves to latch valid address and status bits.

$\overline{\text{CD}}$_DS16

Card Data Size 16, driven by 16 bit slaves to provide an indication to the master about their data bus width. Eight-bit slaves do not drive this line.

$\overline{\text{DS}}$_16_RTN

Data Size 16 Return. A signal generated by the PS/2 system by AND-ing all the $\overline{\text{CD}}$_DS16 signals received from all the slave connected to the bus. The signal is provided by the PS/2 system to the bus masters.

M/$\overline{\text{IO}}$

Memory/$\overline{\text{IO}}$, driven by the bus master and indicates a memory or IO cycle. M/$\overline{\text{IO}}$ is latched by the slave at the leading edge of $\overline{\text{CMD}}$ signal.

$\overline{\text{S0}}$, $\overline{\text{S1}}$

Status bits, driven by the bus master and indicate the start of read or write cycle. The status bits are latched by the slaves using the leading edge of $\overline{\text{CMD}}$.

$\overline{\text{CMD}}$

Command signal is driven by the bus master and defines the period data is valid on the data bus. The leading edge of $\overline{\text{CMD}}$ is used to latch the unlatched signals: A0–23, M/$\overline{\text{IO}}$, $\overline{\text{S0}}$, and $\overline{\text{S1}}$. The trailing edge of $\overline{\text{CMD}}$ indicates the end of the bus cycle.

$\overline{\text{CD}}$_SFDBK

Card Select Feedback. When a bus master addresses a memory or an IO slave, the addressed slave drives $\overline{\text{CD}}$_SFDBK active as a positive acknowledgement of its presence at the specified address.

CD_CHRDY

Channel Ready. This line is pulled inactive (not ready) by a slave to allow additional time to complete a bus cycle.

CHRDYRTN

Channel Ready Return. Generated on the PS/2 system board by AND-ing the CD_CHRDY signals driven by all the slaves. The signal is provided by the system to the mastr driving the bus.

MCA Signal Descriptions (Cont.)

ARB0-ARB3

Arbitration Bus priority signals. These four signals represent the priority levels for masters seeking control on the bus. The four signals represent 16 priority levels, level 15 represents the lowest priority, level 0 represents the highest priority and belongs to the PS/2 system.

ARB/$\overline{\text{GNT}}$

Arbitrate/$\overline{\text{Grant}}$. When high, this signal indicates an arbitration cycle is in process. When low, indicates that a master has been granted. ARB/$\overline{\text{GNT}}$ is driven by the system.

$\overline{\text{PREEMPT}}$

Used by the arbitration bus masters to request the bus.

$\overline{\text{BURST}}$

Indicates that the master requests the bus for transferring a block of data.

$\overline{\text{IRQ}}$

Interrupt Request is used to signal the system that a device requires attention.

CHRESET

Channel Reset, active high reset signal generated by the system and sent to all the boards on the bus.

Figure 1. Micro Channel Basic Transfer Cycle

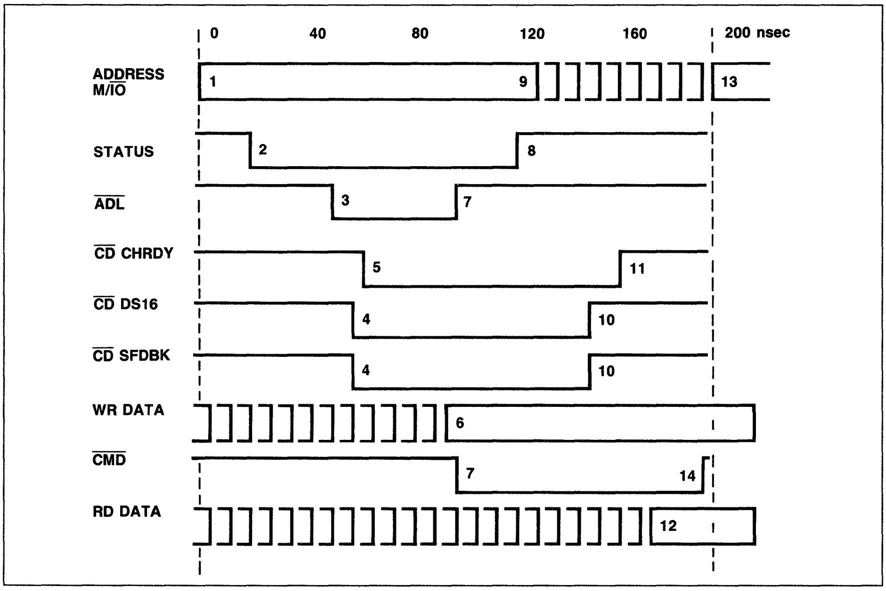

0
40
80
120
160
200 nsec
ADDRESS M/$\overline{\text{IO}}$
STATUS
$\overline{\text{ADL}}$
$\overline{\text{CD}}$ CHRDY
$\overline{\text{CD}}$ DS16
$\overline{\text{CD}}$ SFDBK
WR DATA
$\overline{\text{CMD}}$
RD DATA
1
9
13
2
8
3
7
5
11
4
10
4
10
6
7
14
12

Table 1. The States Generated M/$\overline{\text{IO}}$, $\overline{\text{S0}}$ and $\overline{\text{S1}}$

M/$\overline{\text{IO}}$	$\overline{\text{S0}}$	$\overline{\text{S1}}$	
0	0	1	I/O write.
0	1	0	I/O read.
1	0	1	Memory write.
1	1	0	Memory read.

MCA Timing Parameters

The PAC1000 as a bus master transfers data on the MCA bus with a control sequence based on the following events:

- ❑ The addres bus and M/$\overline{\text{IO}}$ signal become valid.
- ❑ The status signals $\overline{\text{S0}}$ and $\overline{\text{S1}}$ are valid 10 nsec minimum after (1).
- ❑ $\overline{\text{ADL}}$ is valid 45 nsec minimum after (1).
- ❑ In response to the unlatched address decode, the selected slave responses with $\overline{\text{CD}}$_SFDBK (and $\overline{\text{CD}}$_DS16 if it is a 16 bit slave). The maximum allowable response time of the slave is 55 nsec maximum from (1).

MCA Timing Parameters (Cont.)

□ In response to (1), the slave responds with $\overline{\text{CD}}$_CHRDY. The maximum allowable response time is 60 nsec maximum from (1).

□ Write data appears on the bus for the write cycle. The data has to be valid before the leading edge of $\overline{\text{CMD}}$.

□ $\overline{\text{CMD}}$ becomes active and $\overline{\text{ADL}}$ inactive typically 85 nsec minimum after (1). The unlatched signals on the bus are latched.

□ The status signals become inactive after they were latched.

□ The address bus becomes inactive after the address was latched.

□ In response to the address change, the slave's unlatched responses ($\overline{\text{CD}}$_CFDBK AND $\overline{\text{CD}}$_DS16) are invalid.

□ System stays in this state until $\overline{\text{CD}}$_CHRDY is ready.

□ The slave places data on the bus in response to a read.

□ The address and M/$\overline{\text{IO}}$ are valid for the next cycle.

□ $\overline{\text{CMD}}$ goes inactive, ending the cycle.

Operation Modes

The PAC1000 working as a MCA controller can handle the following functions:

□ Bus signal generator.

□ Card setup.

The bus arbitration logic and signal decoding are pure asynchronous functions and implemented by two PALs.

Bus Slave Board

On a bus slave board the PAC1000 may be used to implement the POS registers.

The Programmable Option Select (POS) registers main objectives are:

□ Eliminate switches from the board.

□ Positively identify any card connected to the system.

The POS registers on a PS/2 board replace the switches by using software writeable registers. There are eight POS registers, each one is 8-bit wide. The POS registers are addressed by $\overline{\text{CD}}$_SETUP signal and by address bits A0-2. The POS registers are located at I/O addresses 100H to 107H. The eight POS registers are located in the PAC1000 and control the board's functions.

The POS registers' interface to the MCA is a decoder which decodes the sytem's access to the registers and generates the RD and WR signals to the PAC1000.

The address decoder and slave logic are most of the circuitry needed for the slave functions. The decoder has to decode the address on the bus and to respond with $\overline{\text{CD}}$_SFDBK, $\overline{\text{CD}}$_CHRDY and $\overline{\text{CD}}$_DS16 signals. The address decoder might be for memory, I/O or for both. The decoder's size depends on the number of address bits it is decoding. The decoder's $\overline{\text{CS}}$ outputs are latched by the leading edge of $\overline{\text{CMD}}$ and are stable until the end of the bus cycle. The decoder generates the feedbacks to the bus, $\overline{\text{CD}}$_SFDBK, $\overline{\text{CD}}$_DS16 and $\overline{\text{CD}}$_CHRDY. These signals are not latched and are very time critical. The decoder responds with these outputs at 55 nsec maximum after the address is stable.

Bus Master Board

A master board is a board with a CPU which requests the MCA bus. When granted by the PS/2 system, the master board is driving the bus signals.

On a master board the PAC1000 can handle the following functions:

□ POS registers (similar to the bus slave board).

□ Generation of the bus signals

The other functions of a bus controller are implemented by PALs because the functions are pure asynchronous.

The bus signals are generated by the PAC1000 after the CPU is granted to be a bus master. The process of getting the bus is done in the following sequence:

□ The CPU is requesting the bus through one of the interface lines with the PAC1000.

□ The PAC1000 is setting the bus request line which is buffered by drivers and sent to the MCA system.

□ The system gets the request, and sets a bus arbitration cycle which is handled by the bus arbiter circuit (a PAL).

Operation Modes (Cont.)

□ The bus arbiter sends the PAC1000 the signal MASTER which tells the board that the bus was granted and the board may drive the bus.

□ The PAC1000 signals the CPU that it is the bus master.

□ The PAC1000 is enabling the address and data drivers, and the CPU drives the address and data to the bus.

□ The PAC1000 generates all the bus signals in the right sequence and the right timing requirements as defined by the MCA bus standard.

□ After the CPU is done, it releases the bus request. The PAC1000 translates it to the right signal sequence on the MCA bus and releases the bus buffers.

On the bus master board the PAC1000 may implement a lot of control functions and save glue logic.

For example:
The PAC1000 can handle several DMA operations on the board, or be used as a high speed controller for various applications.

PAC1000 in a Micro Channel Slave Board

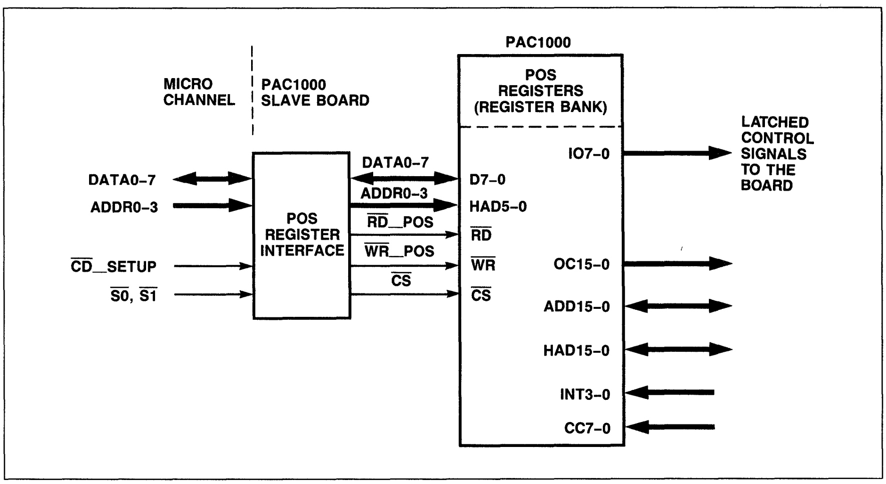
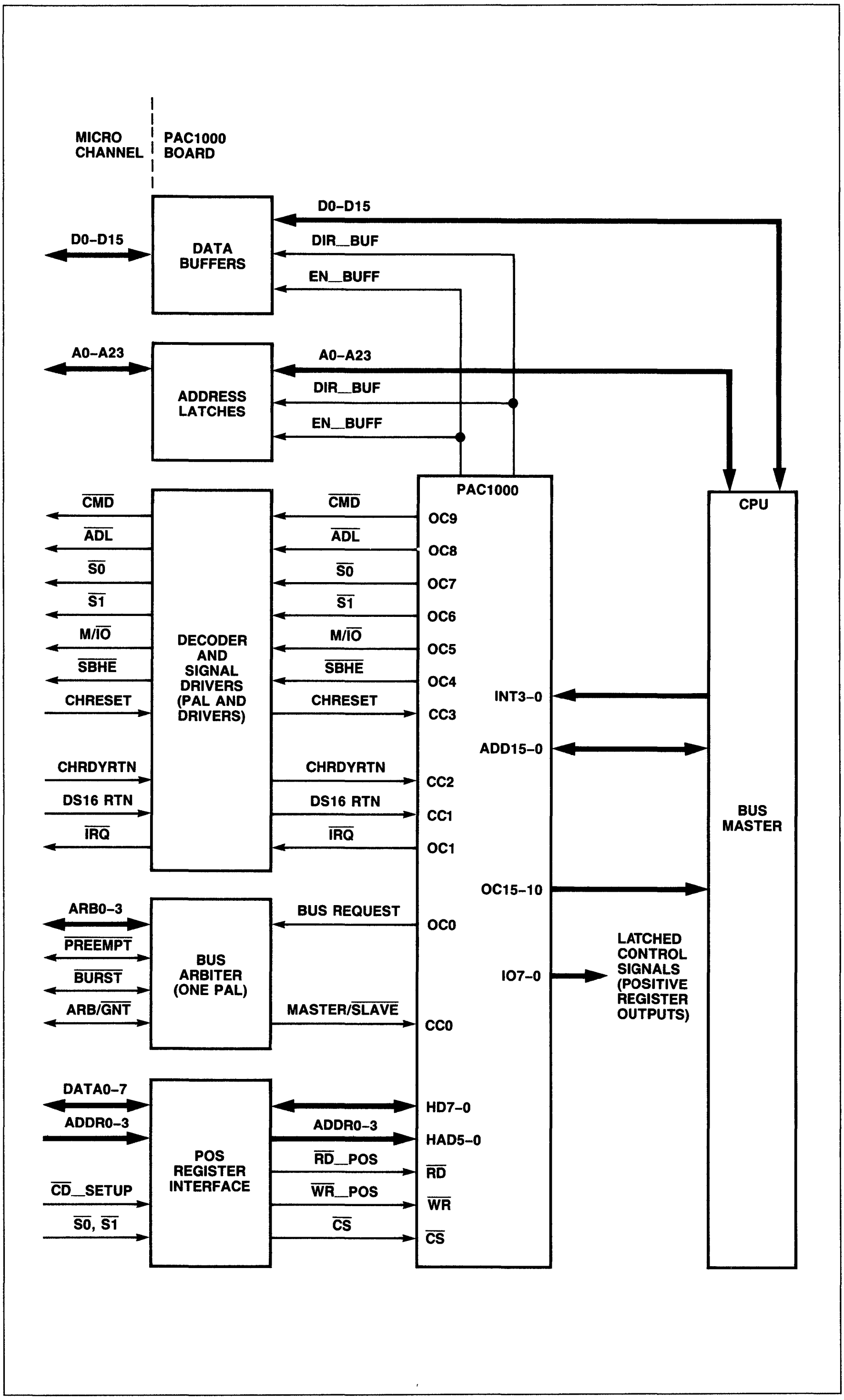


Table 2. Driver Requirement for PS/2 Signals

Signal Name	Driver Type
A(0–23)	TS 24 mA (TS = Three-State)
D(0–15)	TS 24 mA
$\overline{\mathrm{ADL}}$	TS 24 mA
$\overline{\mathrm{CD}}$_DS16	TP 6 mA (TP = Totem Pole)
$\overline{\mathrm{DS}}$_16RTN	BD 24 mA (BD = Bus Driver)
M/$\overline{\mathrm{IO}}$	TS 24 mA
$\overline{\mathrm{S0}}$, $\overline{\mathrm{S1}}$	TS 24 mA
$\overline{\mathrm{CMD}}$	TS 24 mA
$\overline{\mathrm{CD}}$_SFDBK	TP 6 mA
$\overline{\mathrm{CD}}$_CHRDY	TP 6 mA
CHRDYRTN	BD 24 mA
ARB(0–3)	OC 24 mA (OC = Open Collector)
$\overline{\mathrm{PREEMPT}}$	OC 24 mA
$\overline{\mathrm{BURST}}$	OC 24 mA
ARB/$\overline{\mathrm{GNT}}$	BD 24 mA

PAC1000 as a Micro Channel Master



WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Note 003

High-End SAM Applications Using Microassembler Design Entry

Scope of This Application Note

This Application Note describes the SAM microsequencer design entry process utilizing ASM microassembler input syntax and provides illustrations of all basic concepts needed to execute a SAM microassembler design. Basic microassembler functionality is reviewed, its utilization of SAM internal resources, as well as user convenience features. Cascading of multiple SAM devices to address large design problems is also covered. To illustrate a practical application of SAM, a graphics controller application is presented in detail along with annotated ASM source code.

The reader is referred to WSI's SAM448 Data Sheet for details concerning device architecture and performance. A general knowledge of SAM device architecture is assumed as background for this Application Note.

The SAM Solution

The SAM (Stand-Alone Microsequencer) User-Configurable device provides a unique solution for high-performance control functions. The combination of a microcoded engine with a branch EPLD front-end gives SAM the ability to handle high-complexity tasks while still achieving high clock rates. The basic SAM architecture is shown in Figure 1.

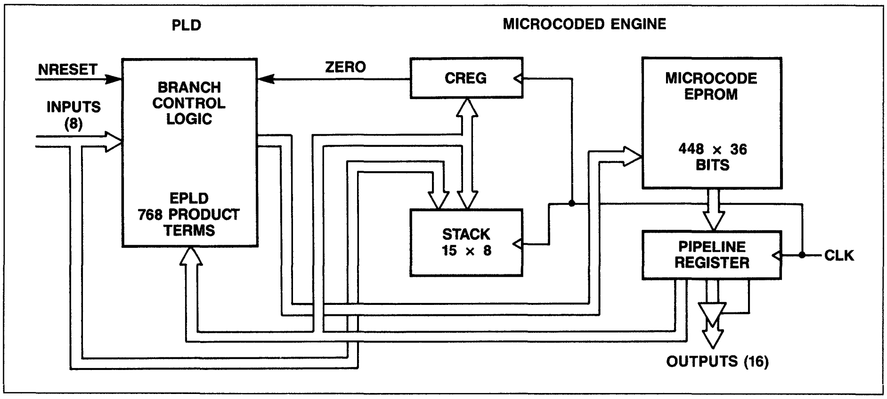
Programming the SAM device for a particular application involves specifying multi-way branch transition specifications for the branch EPLD, and instruction and output strings for the required number of microcode words in SAM's EPROM control memory. (See the SAM448 Data Sheet for further information). This task is eased by the use of the SAM+PLUS development system.

SAM+PLUS System Overview

The SAM+PLUS PC-based design development system provides an efficient mechanism for entry and automatic compilation of SAM designs. Interactive functional simulation is provided in SAM+PLUS to enable rapid verification of design flows and operation. PC-compatible programming hardware is also available to allow device programming right at the designer's desk. Given the fact that control logic is frequently difficult to design, and particularly prone to design alterations, the ability to enter, compile, simulate and test a design in rapid fashion results in an effective design system.

SAM+PLUS actually supports two design entry methods, one using ASMILE state machine input language, the other ASM

Figure 1. SAM448 Block Diagram



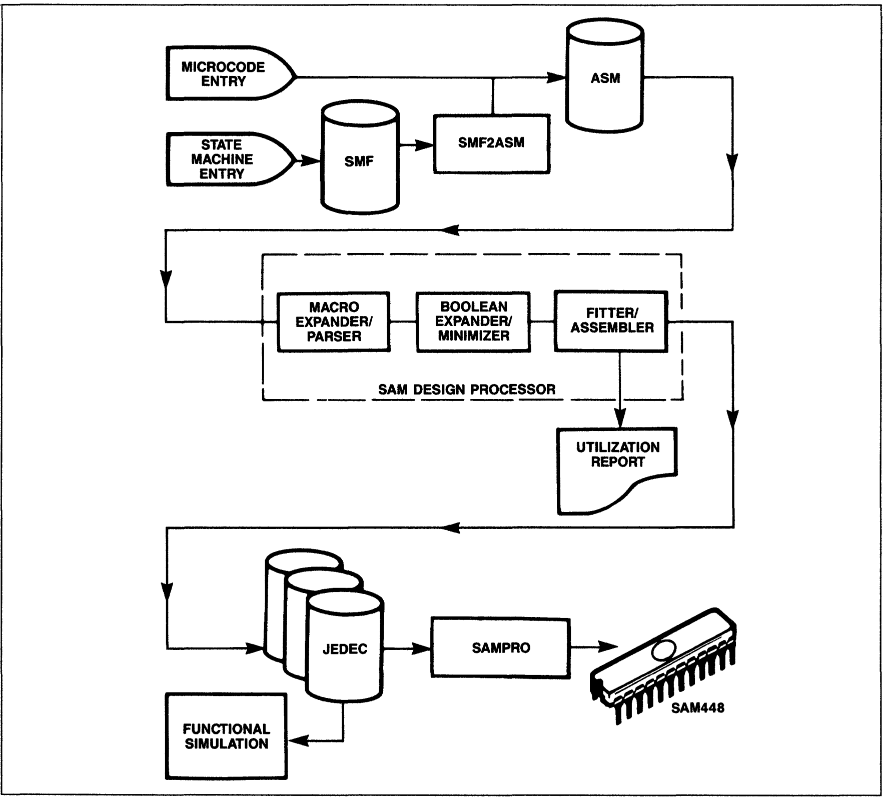
SAM+PLUS System Overview (Cont.)

microassembler format, as shown in Figure 2. SAM ASMILE input is described in WSI Application Note #4, referenced below. This Application Note will focus on microassembler input.

Microassembler design entry begins with the creation of a design file on the PC using any standard text editor. Next, the SAM Design Processor (SDP) takes the ASM input file, automatically minimizes transition equations and generates the device programming code. A Utilization Report is generated which reports total resources consumed, absolute memory assignments of microassembler instructions and compiler-assigned pinouts. A standard JEDEC file is generated to allow programming of the device right on the PC.

For larger designs, multiple SAM devices may be horizontally cascaded to increase the number of available control outputs. The microassembler supports the specification of a single source file for a multiple-SAM application, and automatically generates the separate JEDEC files for the programming of each of the devices at compile time. The JEDEC file, which represents the actual template of the specific application implemented, may be used as input to the SAMSIM (SAM SIMulator) program which provides functional simulation capability. Hard-copy output of simulation results may be obtained, as well as on-line "logic analyzer" viewing capability. Multi-chip applications using horizontal cascading is also supported by the functional simulator.

Figure 2. SAM+PLUS Block Diagram



Choosing Appropriate Applications for SAM

The SAM architecture supports high-performance synchronous control applications. It is important to realize that all outputs from SAM are asserted synchronously with respect to the device clock, and as such SAM implements a classic Moore machine architecture. Similarly, as can be seen in the SAM Data Sheet, all inputs must obey a required set-up time (Tsu) relative to the Clock input.

Choosing Appropriate Applications for SAM (Cont.)

In order to obtain greater than 16 outputs in a SAM design, the concept of horizontal cascading may be used. Similarly, if greater control store (microcode) depth is required, multiple SAM devices may be vertically cascaded, sharing a common control output bus. Both cascading approaches may be simultaneously used for problems requiring increased capacity in both dimensions.

In order to determine whether a given application will be suitable for SAM, the following "rules-of-thumb" derived from the device architecture and specifications are useful. These guidelines are for single SAM implementations. Cascaded SAM configurations may expand output count and memory depth substantially. For example, SAM+PLUS supports horizontal cascading of up to 8 SAM448 devices, for a total output count of 128 lines!

SAM Application Guidelines

- Operating frequency up to specified SAM's Fmax
- Synchronous operation
- Up to eight control inputs (exclusive of Clock and nRESET)
- Up to sixteen control outputs (single device)
- Up to 256 primary microcode locations
- Up to 64 of 256 primary microcode locations may be multi-way (external conditional) branches (single device)
- Transition expressions reducible to four product terms per IF . . . THEN expression

Applications which satisfy this list will in all likelihood fit into a single SAM device.

Microassembler Input Overview

Shown in Figure 3 is an example of the structure of a SAM ASM input file. This file may be created using any standard text editor. It is important that the text editor is used in non-document mode in order to prevent the insertion of any spurious format control characters which may be detected by the ASM microassembler parser at compile time as input errors. Other than this constraint, input is essentially free-form and may be structured for readability and overall clarity.

The case of characters inserted into the ASM file is significant, so be sure that case significance is maintained. For example, the names "RWB" and "rwb" are not the same.

Comments may be inserted freely into the source code, delimited by leading and trailing percent signs (%).

The basic format of a SAM ASM file consists of the following sections:

```
[HEADER]
 PART
 INPUTS
 OUTPUTS
[PINS]
[DEFAULT]
[MACROS]
[EQUATIONS]
 PROGRAM
 END$
```

Those sections noted within brackets are optional and may be omitted if not required.

Header

The HEADER contains user-specified design identifier information. It may include design title, designer's name, date, revision information, etc.

Part

The PART section of the ASM file specifies the target SAM device or devices the application is intended for. By specifying AUTO, the user permits the SAM+PLUS software to pick the optimal device or set of devices for the application based upon minimal pin count. Multiple devices may be invoked for designs requiring a larger number of total outputs than a single SAM device can supply, i.e., the SAM+PLUS software supports horizontal cascading (see SAM Data Sheet) of devices at a source code level. This cascading capability may be invoked by utilizing AUTO with a design requiring high output count as noted, or may be explicitly defined by supplying a list of devices after PART which the design is to be fitted into. As shown in the example below, two SAM448 devices are going to be used in this application, and have been explicitly entered. Devices may be cascaded horizontally up to a width of 128 outputs in a single source code listing and simulated

Microassembler Input Overview (Cont.)

as one large virtual SAM. Separate JEDEC files are generated for each device to support programming devices when design is complete.

Inputs

The single INPUTS section of the ASM file defines all external inputs into the design, as well as any required user pin assignments. Pin assignments are specified by the format input_name @ pin_number. Note that since in a horizontally cascaded design all design inputs must be common, there will never be more inputs specified in a source file than are available in a single SAM device.

Only user-defined inputs should appear in the INPUTS section: the CLOCK and nRESET inputs to SAM, being fixed-function pins, should not be included.

Outputs

The OUTPUTS section(s) of the ASM file contains a list of all outputs from the design as well as any pin assignments. Pin assignment syntax is similar to input pin assignments. If multiple SAMs are specified in the PART: section of the design file (horizontal cascading), there will be multiple OUTPUTS sections in the ASM file, one for each SAM component. If AUTO parts selection is used for a cascaded design, a single OUTPUTS declaration may be used to specify all required outputs. At compile time, outputs will be assigned to the various devices automatically.

Output names must be unique across all OUTPUTS section declarations.

AUTO parts selection may not be used in conjunction with user-defined pin assignments.

Pins

The PINS section allows mapping of external variable names onto internal variable names for convenience. For example, a user may have an active-low signal in his system he has called /WR which enters into his transition specifications in his SAM design. To keep the logical sense of such specifications clear, it is wise to transform all active-low external signals into equivalent active-high names internally, e.g., /WR = WRint.

Default

The DEFAULT section allows the specification of a default output combination to be used whenever the output string is not explicitly defined in an instruction. In a single SAM device specification the syntax is simply DEFAULT: [O0 . . . On], where O0 through On represents a binary string corresponding to the n outputs specified for the SAM design. Default output values are matched to output pins in the order they appear in the OUTPUTS declaration. If multiple OUTPUTS sections appear in a cascaded SAM application, the DEFAULT specifier is increased in width to accommodate this change as shown in the example. Only one DEFAULT section may appear per ASM file.

Macros

The MACROS section allows the user to define string equivalences to be substituted universally throughout his ASM source code listing. For example, the user may wish to redefine instruction mnemonics for efficiency or clarity, or may wish to redefine binary output strings with alphanumeric labels. For example,

```
REG1TOALU = "0101111001100000"
```

The left hand side of this expression is undoubtedly easier to remember and type repeatedly into a listing than the right.

Imbedded strings are not macro substituted. Macro instances must be delimited by white space to be recognized. For example, if a macro substitution is defined as

```
REG = "0110"
```

the string 0110 would be substituted into

```
[REG ALU OP] CONTINUE;
```

but not into

```
[BREG4 ALU OP] CONTINUE;
```

Equations

The EQUATIONS section of the ASM file is available for the definition of intermediate equations to be used later in the design. Entry of transition specifications may be eased by defining intermediate variables initially, and then invoking them during the design. For example,

```
EventClk = I1 */I4 + I3*I6*/I7
```

might be defined in the EQUATIONS

Microassembler Input Overview (Cont.)

section, and then utilized later in an IF . . . THEN. . . ELSE statement or statements, such as

```
IF EventClk THEN [ ] JUMP START;
```

Program

The PROGRAM section of the ASM file actually specifies the sequence of instructions to be executed and associated outputs required from the SAM device. The format of a basic instruction specification in the PROGRAM section is

```
label: [output-spec] opcode;
```

label is an optional alphanumeric string which may be used to identify the instruction in branching expressions, etc.

[output-spec] represents an actual numeric string of the correct length (in either binary, hexadecimal or decimal notation), a Macro substitution with numeric equivalence (as defined above), or the special character Z which signifies tri-state output pins. Hexadecimal and decimal strings are defined by a string of valid digits of correct length, followed by H or D respectively. In horizontally cascaded applications, all outputs are specified in the single output-spec within brackets. The output-spec defined in the DEFAULT statement will be utilized whenever the output-spec has length zero, i.e., [ ] implies default output-spec.

END$

Every SAM ASM source file must terminate with the END$ terminator.

Multi-Way Branch Syntax

The syntax for multi-way branching within the SAM ASM source file is by way of a complex expression of the form

```
IF (expression1) THEN (instruction1)
ELSEIF (expression2)
  THEN (instruction2)
ELSEIF (expression3)
  THEN (instruction3)
ELSE (instruction4)
```

For example, a complex instruction of this type might look like

```
IF I0*I1*I5*/I7 + I3*I4 + I6*/I0 + /I3*/I1 THEN
  [1111001110010000] CALL
  label1 RETURNTO label2;
ELSEIF I3*/I2 + I5*I6 + /I0*I4*I1 THEN
  [1011000011100011] LOADC
  255 GOTO label3;
ELSEIF I4*I6*I0 THEN [ ] PUSH 15
  GOTO label4;
ELSE [1111111100000001] PUSHI
  GOTO label5;
```

Each expression may be a function of any of the eight SAM external inputs containing up to four product terms.

If more than four product terms are needed to define a transition from one state to another, it is possible to trade-off product term counts for number of multi-way branch destinations. For example, it is perfectly valid to enter

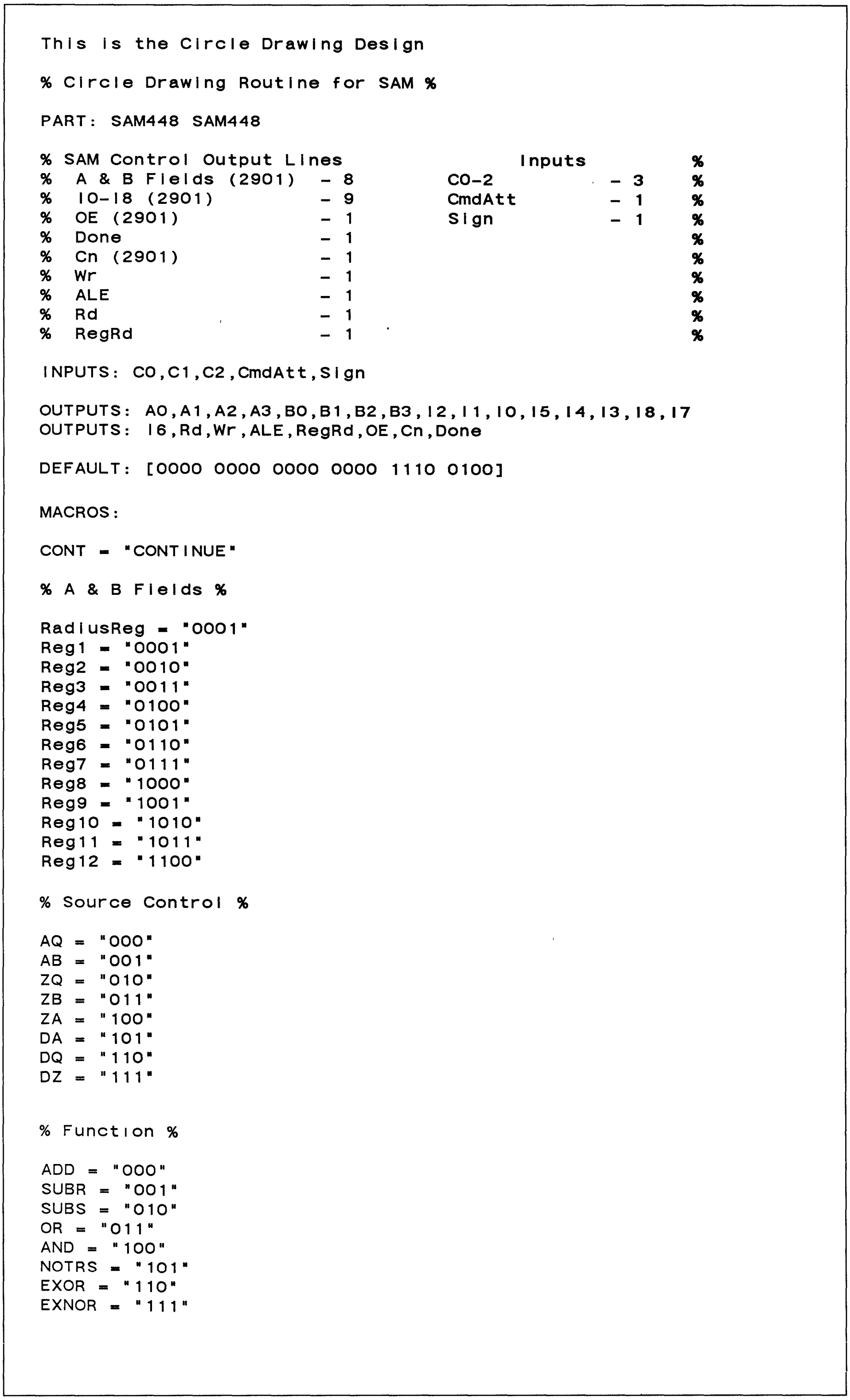
```
IF (expression1) THEN [ ] JUMP START;
ELSEIF (expression2) THEN [ ] JUMP START;
ELSEIF (expression3) THEN [ ] JUMP NEXT1;
ELSE [ ] JUMP NEXT2;
```

Here, expression1 and expression2 could each be four product term expressions, resulting in eight product terms which can be used to specify the transition to START.

Note the inherent priority scheme in the above statements. The SAM architecture physically implements such a priority scheme in the Branch Control Block: the first occurrence of a valid expression results in the execution of the corresponding instruction. If the first three expressions are all false, then instruction4 will be subsequently executed.

Up to 64 such IF . . . THEN . . . ELSE constructs may be implemented in a single SAM program, along with 192 conventional instructions without IF . . . THEN . . . ELSE. The result is a total microcode memory capacity of (64 x 4) + 192 = 448 words.

Figure 3. Circle Drawing Routine



```
This is the Circle Drawing Design

% Circle Drawing Routine for SAM %

PART: SAM448 SAM448

% SAM Control Output Lines                 Inputs              %
%   A & B Fields (2901)   - 8       CO-2                - 3   %
%   I0-I8 (2901)          - 9       CmdAtt              - 1   %
%   OE (2901)             - 1       Sign                - 1   %
%   Done                  - 1                                 %
%   Cn (2901)             - 1                                 %
%   Wr                    - 1                                 %
%   ALE                   - 1                                 %
%   Rd                    - 1                                 %
%   RegRd                 - 1                                 %

INPUTS: CO,C1,C2,CmdAtt,Sign

OUTPUTS: A0,A1,A2,A3,B0,B1,B2,B3,I2,I1,I0,I5,I4,I3,I8,I7
OUTPUTS: I6,Rd,Wr,ALE,RegRd,OE,Cn,Done

DEFAULT: [0000 0000 0000 0000 1110 0100]

MACROS:

CONT = "CONTINUE"

% A & B Fields %

RadiusReg = "0001"
Reg1 = "0001"
Reg2 = "0010"
Reg3 = "0011"
Reg4 = "0100"
Reg5 = "0101"
Reg6 = "0110"
Reg7 = "0111"
Reg8 = "1000"
Reg9 = "1001"
Reg10 = "1010"
Reg11 = "1011"
Reg12 = "1100"

% Source Control %

AQ = "000"
AB = "001"
ZQ = "010"
ZB = "011"
ZA = "100"
DA = "101"
DQ = "110"
DZ = "111"


% Function %

ADD = "000"
SUBR = "001"
SUBS = "010"
OR = "011"
AND = "100"
NOTRS = "101"
EXOR = "110"
EXNOR = "111"
```

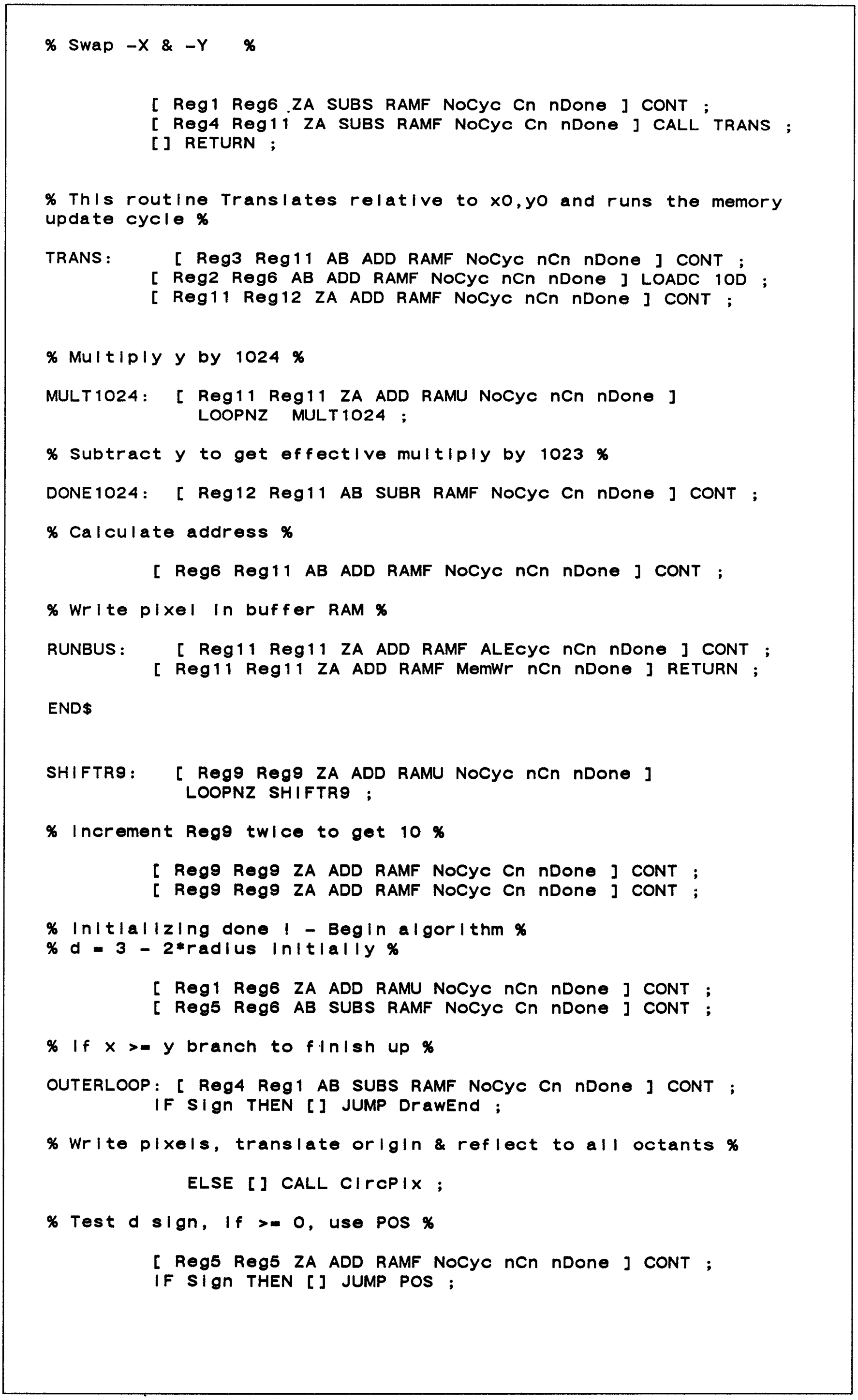
Figure 3. Circle Drawing Routine (Cont.)

```
% Destination  Control %

QREG = "000"
NOP = "001"
RAMA = "010"
RAMF = "011"
RAMQD = "100"
RAMD = "101"
RAMQU = "110"
RAMU = "111"

% Bus Cycle  %

MemWr = "10001"
RegWr = "10011"
ALEcyc = "11100"
NoCyc = "11000"

% Misc %

Cn = "1"
nCn = "0"
Done = "1"
nDone = "0"


EQUATIONS:

PROGRAM:

% Processor Initializes: %

0D:[] JUMP WAIT;
% o Load Coloreg, Radius, X0, Y0 %
% o Issues DrawCirc Command      %

WAIT:       IF CmdAtt*C0'*C1'*C2' THEN [] JUMP DOIT ;
             ELSE [] JUMP WAIT ;

% Move parameters from buffer to 2901 internal registers %
% Radius -> Reg1 (Y) %

DOIT:       [ Reg1 Reg1 AQ ADD NOP RegWr nCn nDone ] CONT ;
          [ Reg1 Reg1 AQ ADD NOP RegWr nCn nDone ] CONT ;


% X0 -> Reg2 %

          [ Reg2 Reg2 AQ ADD NOP NoCyc nCn nDone ] CONT ;
          [ Reg2 Reg2 AQ ADD NOP RegWr nCn nDone ] CONT ;
          [ Reg2 Reg2 AQ ADD NOP RegWr nCn nDone ] CONT ;

% Y0 -> Reg3 %

          [ Reg2 Reg2 AQ ADD NOP NoCyc nCn nDone ] CONT ;
          [ Reg3 Reg3 AQ ADD NOP RegWr nCn nDone ] CONT ;
          [ Reg3 Reg3 AQ ADD NOP RegWr nCn nDone ] CONT ;

% Load constants to 2901 registers %
% 0 -> Reg4 (X) (AND 0 & anything gives 0) %

          [ Reg4 Reg4 ZB AND RAMF NoCyc nCn nDone ] CONT ;
```

Figure 3. Circle Drawing Routine (Cont.)

```
% 3 -> Reg5 (d) %
% Put "1" in Reg5 %

          [ Reg4 Reg5 ZA ADD RAMF NoCyc Cn nDone ] CONT ;


% Shift Reg5 Up one to give 2 %

          [ Reg5 Reg5 ZB ADD RAMU NoCyc nCn nDone ] CONT ;

% While we have it, preload 2 into Reg9 %

          [ Reg5 Reg9 ZA ADD RAMF NoCyc nCn nDone ] CONT ;

% Increment Reg5 to get 3  (whew!!) %

          [ Reg5 Reg5 ZA ADD RAMF NoCyc Cn nDone ] CONT ;

% 6 -> Reg8 (const) - just shift 3 up one! %
% Load 1 in CREG to set-up for next instruction %

          [ Reg5 Reg8 ZA ADD RAMU NoCyc nCn nDone ] LOADC 1D ;

% 10 -> Reg9 (const) %
% Start by shifting Reg9 (now contains 2) up twice to get 8 %
% Reg6 (Temp register)  %

CircPix:    [ Reg4 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
          [ Reg1 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Reflect X to -X %

          [ Reg4 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
          [ Reg1 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Swap X & Y %

          [ Reg1 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
          [ Reg4 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;


% Swap -X & Y %

          [ Reg4 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
          [ Reg1 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Reflect Y %

          [ Reg1 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
          [ Reg4 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Swap -Y & X %

          [ Reg1 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
          [ Reg4 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Reflect -X, -Y  %


          [ Reg4 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
          [ Reg1 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CALL TRANS ;
```

Figure 3. Circle Drawing Routine (Cont.)



```
% Swap -X & -Y   %

          [ Reg1 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
          [ Reg4 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CALL TRANS ;
          [] RETURN ;


% This routine Translates relative to x0,y0 and runs the memory
update cycle %

TRANS:      [ Reg3 Reg11 AB ADD RAMF NoCyc nCn nDone ] CONT ;
          [ Reg2 Reg6 AB ADD RAMF NoCyc nCn nDone ] LOADC 10D ;
          [ Reg11 Reg12 ZA ADD RAMF NoCyc nCn nDone ] CONT ;


% Multiply y by 1024 %

MULT1024:   [ Reg11 Reg11 ZA ADD RAMU NoCyc nCn nDone ]
              LOOPNZ  MULT1024 ;

% Subtract y to get effective multiply by 1023 %

DONE1024:   [ Reg12 Reg11 AB SUBR RAMF NoCyc Cn nDone ] CONT ;

% Calculate address %

          [ Reg6 Reg11 AB ADD RAMF NoCyc nCn nDone ] CONT ;

% Write pixel in buffer RAM %

RUNBUS:     [ Reg11 Reg11 ZA ADD RAMF ALEcyc nCn nDone ] CONT ;
          [ Reg11 Reg11 ZA ADD RAMF MemWr nCn nDone ] RETURN ;

END$


SHIFTR9:    [ Reg9 Reg9 ZA ADD RAMU NoCyc nCn nDone ]
             LOOPNZ SHIFTR9 ;

% Increment Reg9 twice to get 10 %

          [ Reg9 Reg9 ZA ADD RAMF NoCyc Cn nDone ] CONT ;
          [ Reg9 Reg9 ZA ADD RAMF NoCyc Cn nDone ] CONT ;

% Initializing done ! - Begin algorithm %
% d = 3 - 2*radius initially %

          [ Reg1 Reg6 ZA ADD RAMU NoCyc nCn nDone ] CONT ;
          [ Reg5 Reg6 AB SUBS RAMF NoCyc Cn nDone ] CONT ;

% If x >= y branch to finish up %

OUTERLOOP: [ Reg4 Reg1 AB SUBS RAMF NoCyc Cn nDone ] CONT ;
          IF Sign THEN [] JUMP DrawEnd ;

% Write pixels, translate origin & reflect to all octants %

             ELSE [] CALL CircPix ;

% Test d sign, if >= 0, use POS %

          [ Reg5 Reg5 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
          IF Sign THEN [] JUMP POS ;
```

Figure 3. Circle Drawing Routine (Cont.)

```
  % Compute d = d + 4*x + 6 %
  % First 4*x %

             ELSE [ Reg4 Reg6 ZA ADD RAMU NoCyc nCn nDone ] CONT ;
          [ Reg6 Reg6 ZA ADD RAMU NoCyc nCn nDone ] CONT ;

  % Add 6 %

          [ Reg8 Reg6 AB ADD RAMF NoCyc nCn nDone ] CONT ;
          [ Reg6 Reg5 AB ADD RAMF NoCyc nCn nDone ] JUMP IncX ;

  % Compute d = d + 4*(x-y) + 10 %
  % First x-y %

  POS:      [ Reg1 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
          [ Reg4 Reg6 AB SUBS RAMF NoCyc Cn nDone ] LOADC 1D ;

  % Then 4*(x-y) %

  SHIFTR6:  [ Reg6 Reg6 ZA ADD RAMU NoCyc nCn nDone ]
              LOOPNZ SHIFTR6 ;


  % Add 10 %

          [ Reg9 Reg6 AB ADD RAMF NoCyc nCn nDone ] CONT ;
          [ Reg6 Reg5 AB ADD RAMF NoCyc nCn nDone ] CONT ;


% Decrement y %

        [ Reg1 Reg1 ZA SUBR RAMF NoCyc nCn nDone ] CONT ;

% Increment x  and repeat til x = y %

IncX:   [ Reg4 Reg4 ZA ADD RAMF NoCyc Cn nDone ] JUMP OUTERLOOP ;

% Last pixel write / ends octant with x = y (45 degrees) %

DrawEnd:   [] Call CircPix ;
        [] LOADC 16D ;

% Issue Done to processor for 16 clocks %

DoDone:    [ Reg1 Reg1 ZA ADD RAMF NoCyc nCn Done ]
             LOOPNZ DoDone ONZERO WAIT ;

% End Main Routine %

%  This  routine reflects the pixel into all octants and calls  a
routine which translates the pixel relative to x0,y0,
calculates the pixel address as addr = x + y*1023 and runs the
memory cycle. %
```

SAM Microassembler Opcodes

The basic SAM device instruction set accessible by the user through the microassembler consists of:

CONTINUE
Execute next sequential instruction

JUMP (label1)
Jump to instruction specified @ label1

LOOPNZ (label1) ONZERO (label2)
If Count Register (CREG) is zero, execute instruction @ label2, else decrement CREG and execute instruction @ label1. Useful for one-instruction timing and delay loops.

DECNZ GOTO (label1)
Decrement the CREG if non-zero; execute instruction @ label1.

POPC GOTO (label1)
Top-of-Stack is popped into CREG and the instruction @ label1 is executed.

POPXORC (constant1) GOTO (label1)
Top-of-Stack is popped, bitewise XORed with (constant1) and loaded to CREG. Instruction @ label1 is next executed. Useful for comparing Top-of-Stack to a value by subsequently testing CREG zero-flag using a LOOPNZ instruction.

LOADC (constant1) GOTO (label1)
CREG is loaded with the value constant1, and instruction @ label1 is next executed.

RETURN
Address of the next instruction is popped from Top-of-Stack and subsequently executed. Used to terminate subroutines.

PUSHLOADC (constant1) GOTO (label1)
CREG value is pushed onto the Stack and CREG is reloaded with constant1.

PUSHI GOTO (label1)
The eight input lines are pushed onto the Top-of-Stack and the instruction at label1 is subsequently executed. May be used to implement a "dispatch" function in conjunction with a subsequent RETURN instruction: external inputs provide address of next SAM instruction.

ANDPUSHI (constant1) GOTO (label1)
The eight input lines are bitwise ANDed with constant1, the result is pushed onto the Stack and the instruction @ label1 is subsequently executed. May be used to mask inputs before loading to CREG or next address.

CALL (label1) RETURNTO (label2)
Label2 is pushed onto the Stack, and the instruction @ label1 is executed next. Used for subroutines.

PUSH (constant1) GOTO (label1)
Constant1 is pushed onto the Stack and the instruction @ label1 is next executed.

The Branch Control Block of SAM is invoked automatically by use of IF . . . THEN . . . ELSE constructs in conjunction with the above instructions. This allows program flow control based upon external inputs as in conventional state machines and multi-way branching in a single clock.

An Actual Design Example

Now that the basic syntax and elements of a SAM ASM file have been covered, a detailed example of a SAM application will be presented: a high-performance Graphics Controller. In this particular application, two SAM devices will be horizontally cascaded to generate the control outputs for a graphics subsystem. This subsystem provides graphics primitive drawing capability for a larger microprocessor-based system.

Figure 4 shows a typical 8086 microprocessor-based system. Beneath the Address/Data Buses is the graphics subsystem to be controlled by the SAM devices, the primary elements of which are a 1 Megabyte high-speed static RAM video frame buffer (giving individual pixel addressing capability), five WS5901 bit-slice elements used to construct a 20 bit ALU/data path engine, and two SAM devices as previously mentioned to provide overall control within the subsystem.

This basic graphics engine represents a user-microcodeable arrangement which can potentially support many primitive graphic drawing operations such as line drawing, polygon filling, drawing of conic sections and others. For the purposes of this example, a single primitive drawing operation which draws circles of arbitrary radius and origin into the frame buffer will

An Actual Design Example (Cont.)

be discussed. The basic concept behind this algorithm will be discussed below.

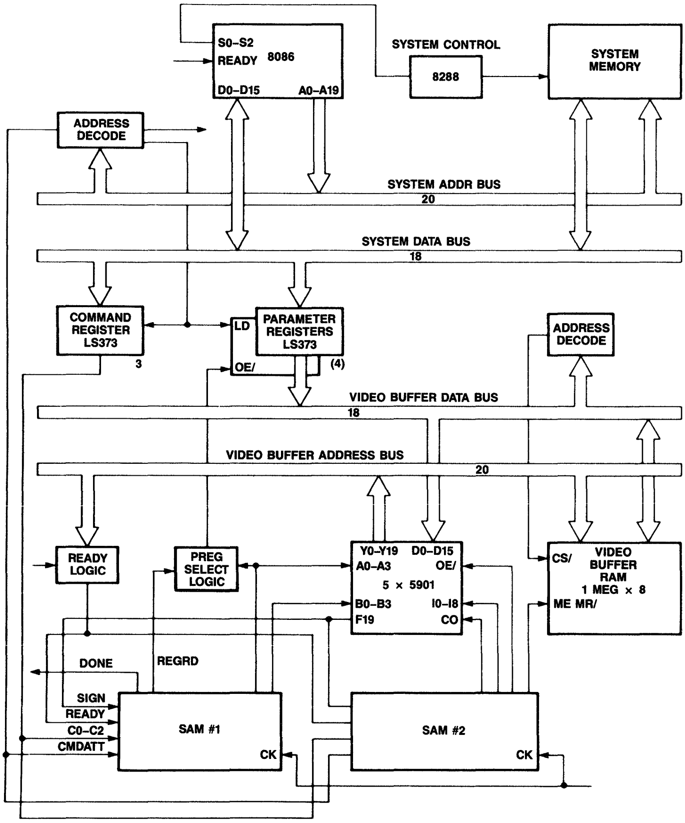
In order to execute its role of controller for this subsystem, the pair of SAM devices must be able to execute the following subfunctions

- Read Commands issued by main microprocessor
- Transfer Parameters associated with commands to Register File in WS5901's
- Initialize Constant Registers in WS5901's to specified values for algorithm
- Compute values for pixels on circle as function of specified Radius for first octant [Assume circle origin = (0,0)]
- Translate x,y coordinates into RAM addresses
- Reflect circle pixel coordinates into remaining seven octants
- Translate pixel coordinates relative to actual origin
- Perform Video Buffer write to all pixel addresses specified
- Issue DONE interrupt to main processor

This activity is done independently of the main microprocessor and frees it up to do other tasks while the operation is performed.

These operations fall into two general categories of controlling bus transfers between various elements (Registers, ALU, RAM, etc.) and sequencing computations performed by the WS5901 ALU in generating the pixel addresses to be set to draw the required circle. The structure of the SAM microassembler code shown above generally follows this flow.

Figure 4. SAM448 Graphics Engine



Circle Drawing Algorithm Overview

The sample algorithm to be implemented in the SAM code to draw the circle is one based upon a methodology developed by Bresenham. In order to speed computation, it exploits the fundamental symmetry of a circle, by calculating the circle points in the first octant (see Figure 5), and then reflecting those coordinates into the other seven octants. For a given pixel location (x,y), reflection involves drawing points

Figure 5. Circle Symmetry Exploited by Bresenham

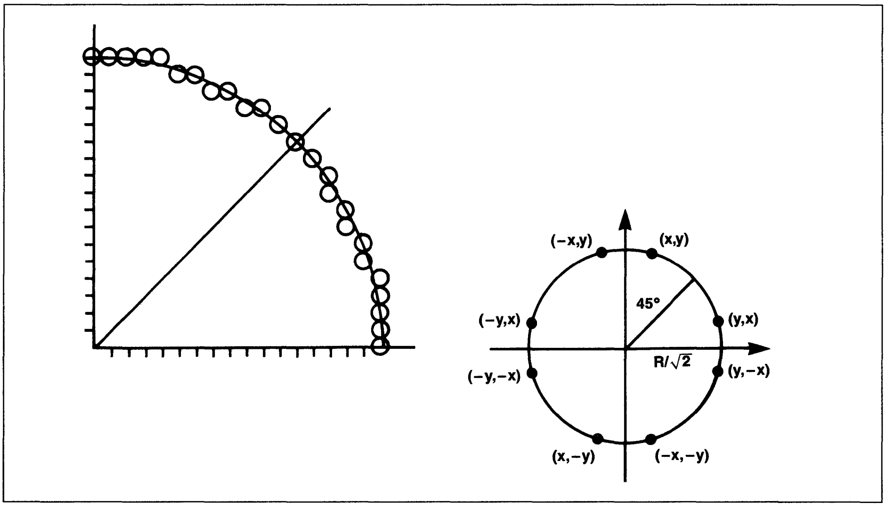


Figure 6. Circle Drawing Algorithm

```
Procedure Circle (radius, value: integer) ;

        var x,y,d : integer ;

begin

        x := 0 ;
        y := raduis ;
        d := 3 -2 * radius ;

      while x < y do begin

                CircleDraw (x,y,value);

                if d < 0

                      then d := d + 4 * + 6

                      else begin

                        d := d + 4 * (x-y) + 10 ;

                        y := y - 1

                      end

                x := x + 1

        end

        if x = y then CircleDraw (x,y, value) ;

end
```

Circle Drawing Algorithm Overview (Cont.)

(-x,y), (x,-y), (-x,-y), as well as those points with x and y swapped. In drawing the points for a circle in the first octant, one can easily see that, having just calculated one of the pixel locations, there are only two possible choices for the next pixel location: increment x (horizontal move) and increment x and y (diagonal move). The trick is how to decide, based upon current location, which of the two to pick next.

The entire derivation of the algorithm will not be presented here. However, a complete discussion of the algorithm may be found in Foley and Van Dam (1981), referenced below. Suffice it to say, it is obvious that the best match between actual pixel coordinates and the ideal circle points can be obtained by checking an error term equal to the difference in distance from the circle's center to each of the two potential next pixel choices: the sign of the term will indicate which point to pick to obtain the best fit.

The basic algorithm implemented is shown in Figure 6.

Timing Considerations

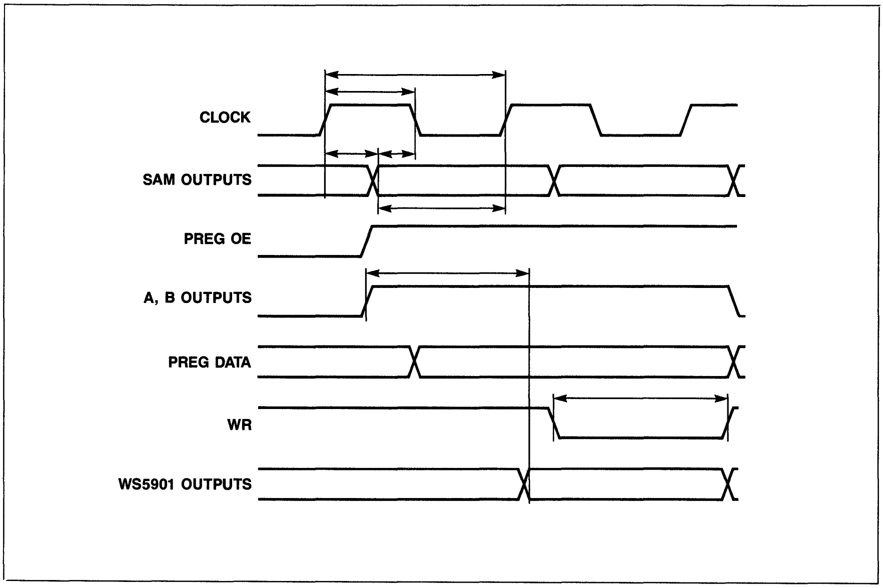
SAM timing analysis is straightforward, as all times are relative to the synchronous clock input. Tsu specifies minimum set-up time for inputs to gain recognition at the next clock edge, while Tco specifies clock-to-output delays for the user-configured output pins. Output tri-state and enable times are speced as Tcz, but are not relevent in this particular application as outputs are always enabled.

For this particular design example, the SAM448-25-controlled graphics subsystem is being driven by a 15 MegaHertz clock. This implies a clock period of 66 nanoseconds. SAM control outputs will reflect a Tco of approximately 18 nanoseconds, while inputs must obey a 18 nanosecond set-up time (Tsu) relative to the clock edge.

High-speed Static RAM will be used for the video frame buffer for two reasons: one is raw speed. The memory must be fast enough to keep up with SAM's high-speed bus cycles. The second is that SRAM requires no refresh cycles, unlike DRAM. Thus more time is available to perform buffer drawing functions: no time is lost for refresh cycles.

Memory consists of CMOS SRAM components organized 8K × 8 with an access time of 45 nanoseconds, and a minimum Write Pulse width of 30 nanoseconds. The CMOS WS5901 bit-

Figure 7. Primary SAM — WS5901 Graphics Controlled Timing



Timing Considerations (Cont.)

slices require a 30 nanosecond propagation delay from A and B Register Address inputs to valid Y output, and a 10 nanosecond set-up time prior to the Clock high-to-low transition on A and B inputs. A timing diagram is shown in Figure 7.

The bus cycle uses a two clock approach. During the first cycle, the WS5901 will generate a pixel address to be set, and during the second cycle, the actual write pulse will be generated by SAM to write the frame buffer.

Operations performed entirely within the WS5901 slices (register transfers, ALU operations, etc.) are all executed in a single clock cycle. Note that Carry Lookahead circuitry is employed with the WS5901 slices to improve arithmetic computation times, but is not explicitly shown in the block diagram.

The algorithm below uses many of the WS5901's operations, as well as many of the internal addressing modes. In the following listings, standard mnemonics have been used for the various Source, Destination, and Operation specifiers. These control lines for the WS5901's are all generated by the SAM devices. These mnemonics, and resulting WS5901 functions, may be found in the WSI WS5901 Data Sheet.

Example Program Listing

Figure 3 is a source listing of the basic circle drawing process. The following comments are worth noting before going further:

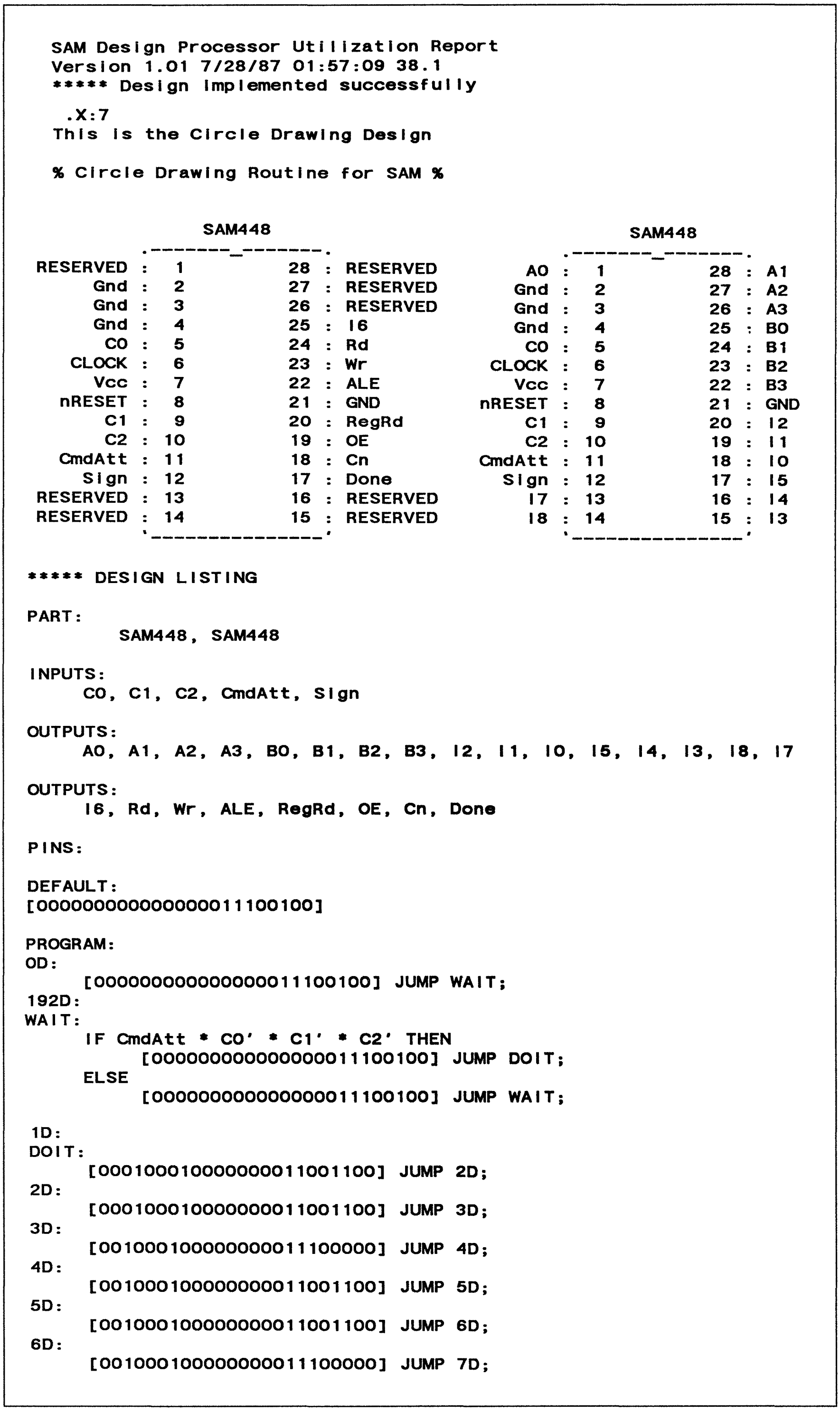
- Two SAM devices are used in a horizontal cascade configuration.
- Extensive MACRO definitions to ease design entry and allow the use of user- and WS5901-specified mnemonics.
- Two subroutines, CircPix and Trans, are invoked multiple times to draw the circle pixels. CircPix reflects the pixels into all octants of the circle as mentioned above, while Trans translates the pixels relative to the actual circle origin and runs the memory update cycle. These functions utilize the Stack and subroutining resources on SAM.
- Since the display is assumed to be 1024 x 1024 pixels, x and y pixel coordinates must be converted to SRAM address locations by multiplying the y coordinate by 1023 and adding the x coordinate.
- The signal CmdAtt is an input to the SAMs from the main processor, signaling that all parameters are loaded to the Parameter Registers, and that a circle drawing operation should be executed. DoneInt is a signal from SAM to the processor, asserted when the drawing operation is complete.

4

Compiling the Design

By convention, microassembler source files are given the extension .ASM. This file is called CIRC.ASM. Compilation of this design involves invoking the SAM+PLUS software and specifying ASM microassembler input format. A variety of runtime options for SAM+PLUS are available, which provide special reporting modes and logging simulation input and output to a special file. For detailed descriptions of the SAM+PLUS user interface and options, see the SAM+PLUS User's Manual. Compilation is an automatic process resulting in the generation of programming "object code" for the EPLD and EPROM blocks on SAM. In this case, two programming files wil be generated, since two devices are required to implement the design. These two files are given the extensions .JD1 and .JD2 to distinguish them. These JEDEC files are not intended to be user readable (as with any object code). Functional simulation uses these programming files for its modeling of SAM operation. An additional product of the compilation is a single Report file (extension -.RPT) which describes the resources which have been used in the SAM devices, pin assignments which have been selected and absolute locations within SAM's microcode assigned to the instructions entered. Figure 8 shows key portions of the CIRC.RPT report file. Notice the assigned pinouts for the two devices, as well as the substitution of absolute addresses for logical labels.

Figure 8. Report File for Circle Drawing Routine

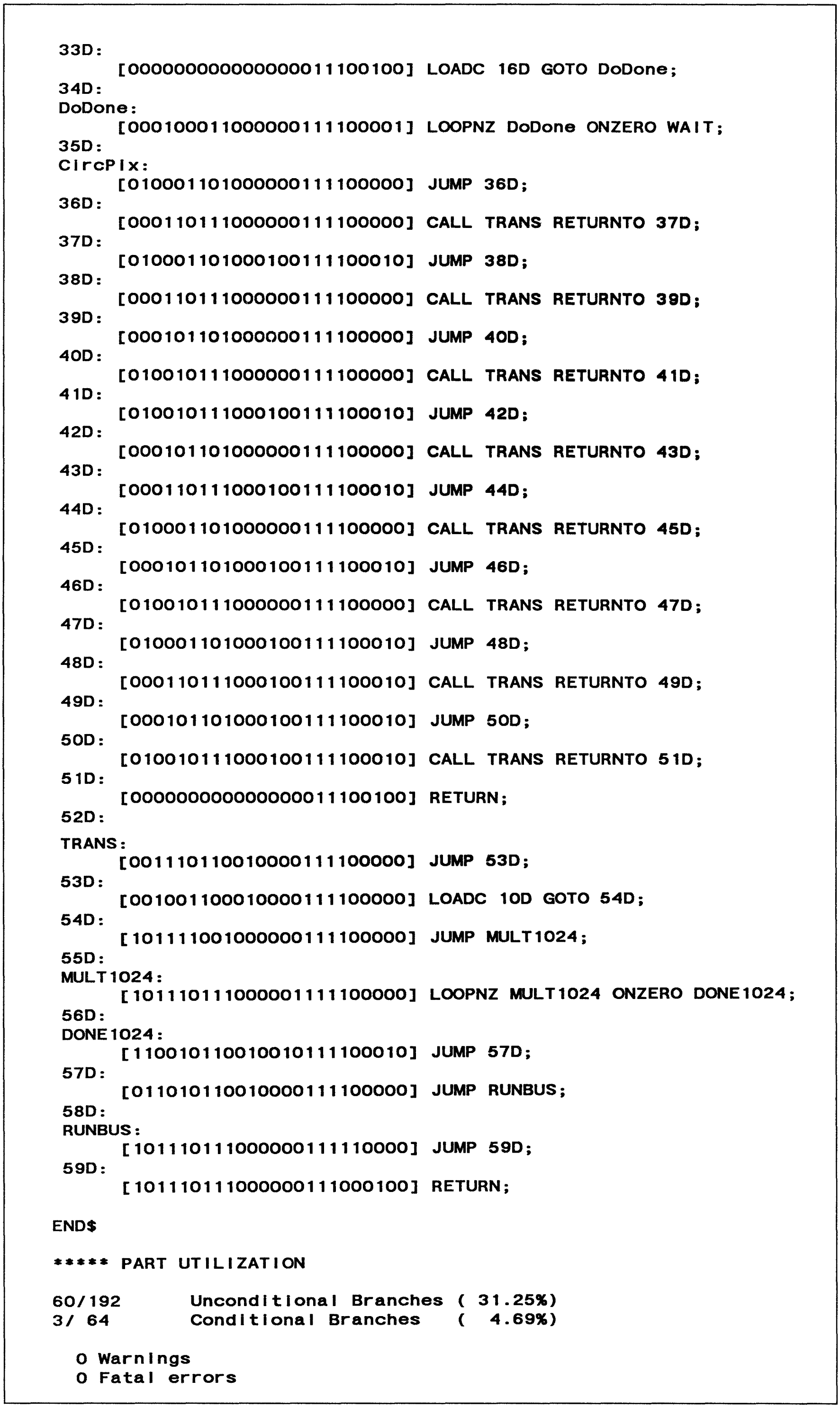


```
  SAM Design Processor Utilization Report
  Version 1.01 7/28/87 01:57:09 38.1
  ***** Design implemented successfully
   .X:7
  This is the Circle Drawing Design

  % Circle Drawing Routine for SAM %


                    SAM448                                        SAM448
             .-------_-------.                             .-------_-------.
 RESERVED :  1             28  :  RESERVED          AO  :  1             28  :  A1
      Gnd :  2             27  :  RESERVED         Gnd  :  2             27  :  A2
      Gnd :  3             26  :  RESERVED         Gnd  :  3             26  :  A3
      Gnd :  4             25  :  I6               Gnd  :  4             25  :  BO
       CO :  5             24  :  Rd                CO  :  5             24  :  B1
    CLOCK :  6             23  :  Wr             CLOCK  :  6             23  :  B2
      Vcc :  7             22  :  ALE              Vcc  :  7             22  :  B3
   nRESET :  8             21  :  GND           nRESET  :  8             21  :  GND
       C1 :  9             20  :  RegRd             C1  :  9             20  :  I2
       C2 : 10             19  :  OE                C2  : 10             19  :  I1
   CmdAtt : 11             18  :  Cn            CmdAtt  : 11             18  :  IO
     Sign : 12             17  :  Done            Sign  : 12             17  :  I5
 RESERVED : 13             16  :  RESERVED          I7  : 13             16  :  I4
 RESERVED : 14             15  :  RESERVED          I8  : 14             15  :  I3
             '---------------'                             '---------------'

***** DESIGN LISTING

PART:
        SAM448, SAM448

INPUTS:
     CO, C1, C2, CmdAtt, Sign

OUTPUTS:
     AO, A1, A2, A3, BO, B1, B2, B3, I2, I1, IO, I5, I4, I3, I8, I7

OUTPUTS:
     I6, Rd, Wr, ALE, RegRd, OE, Cn, Done

PINS:

DEFAULT:
[000000000000000011100100]

PROGRAM:
OD:
     [000000000000000011100100] JUMP WAIT;
192D:
WAIT:
     IF CmdAtt * CO' * C1' * C2' THEN
          [000000000000000011100100] JUMP DOIT;
     ELSE
          [000000000000000011100100] JUMP WAIT;

 1D:
 DOIT:
      [000100010000000011001100] JUMP 2D;
 2D:
      [000100010000000011001100] JUMP 3D;
 3D:
      [001000100000000011100000] JUMP 4D;
 4D:
      [001000100000000011001100] JUMP 5D;
 5D:
      [001000100000000011001100] JUMP 6D;
 6D:
      [001000100000000011100000] JUMP 7D;
```

Figure 8. Report File for Circle Drawing Routine (Cont.)

```
7D:
      [001100110000000011001100] JUMP 8D;
8D:
      [001100110000000011001100] JUMP 9D;
9D:
      [010001000111000111100000] JUMP 10D;
10D:
      [010001011000000111100010] JUMP 11D;
11D:
      [010101010110001111100000] JUMP 12D;
12D:
      [010110011000000111100000] JUMP 13D;
13D:
      [010101011000000111100010] JUMP 14D;
14D:
      [010110001000001111100000] LOADC 1D GOTO SHIFTR9;

15D:
SHIFTR9:
      [100110011000001111100000] LOOPNZ SHIFTR9 ONZERO 16D;
16D:
      [100110011000000111100010] JUMP 17D;
17D:
      [100110011000000111100010] JUMP 18D;
18D:
      [000101101000001111100000] JUMP 19D;
19D:
      [010101100010100111100010] JUMP OUTERLOOP;
20D:
OUTERLOOP:
      [010000010010100111100010] JUMP 193D;
193D:
      IF SIgn THEN
            [000000000000000011100100] JUMP DrawEnd;
      ELSE
            [000000000000000011100100] CALL CIrcPIx RETURNTO 21D;
21D:
      [010101011000000111100000] JUMP 194D;
194D:
      IF SIgn THEN
            [000000000000000011100100] JUMP POS;
      ELSE
            [010001101000001111100000] JUMP 22D;
22D:
      [011001101000001111100000] JUMP 23D;
23D:
      [100001100010000111100000] JUMP 24D;
24D:
      [011001010010000111100000] JUMP IncX;
25D:
POS:
      [000101101000000111100000] JUMP 26D;
26D:
      [010001100010100111100010] LOADC 1D GOTO SHIFTR6;
27D:
SHIFTR6:
      [011001101000001111100000] LOOPNZ SHIFTR6 ONZERO 28D;
28D:
      [100101100010000111100000] JUMP 29D;
29D:
      [011001010010000111100000] JUMP 30D;
30D:
      [000100011000010111100000] JUMP IncX;
31D:
IncX:
      [010001001000000111100010] JUMP OUTERLOOP;
32D:
DrawEnd:
      [000000000000000011100100] CALL CIrcPIx RETURNTO 33D;
```

Figure 8. Report File for Circle Drawing Routine (Cont.)



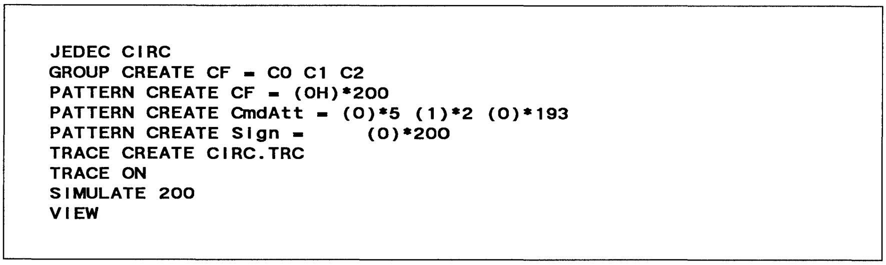
```
33D:
     [000000000000000011100100] LOADC 16D GOTO DoDone;
34D:
DoDone:
     [000100011000000111100001] LOOPNZ DoDone ONZERO WAIT;
35D:
CircPix:
     [010001101000000111100000] JUMP 36D;
36D:
     [000110111000000111100000] CALL TRANS RETURNTO 37D;
37D:
     [010001101000100111100010] JUMP 38D;
38D:
     [000110111000000111100000] CALL TRANS RETURNTO 39D;
39D:
     [000101101000000111100000] JUMP 40D;
40D:
     [010010111000000111100000] CALL TRANS RETURNTO 41D;
41D:
     [010010111000100111100010] JUMP 42D;
42D:
     [000101101000000111100000] CALL TRANS RETURNTO 43D;
43D:
     [000110111000100111100010] JUMP 44D;
44D:
     [010001101000000111100000] CALL TRANS RETURNTO 45D;
45D:
     [000101101000100111100010] JUMP 46D;
46D:
     [010010111000000111100000] CALL TRANS RETURNTO 47D;
47D:
     [010001101000100111100010] JUMP 48D;
48D:
     [000110111000100111100010] CALL TRANS RETURNTO 49D;
49D:
     [000101101000100111100010] JUMP 50D;
50D:
     [010010111000100111100010] CALL TRANS RETURNTO 51D;
51D:
     [000000000000000011100100] RETURN;
52D:
TRANS:
     [001110110010000111100000] JUMP 53D;
53D:
     [001001100010000111100000] LOADC 10D GOTO 54D;
54D:
     [101111001000000111100000] JUMP MULT1024;
55D:
MULT1024:
     [101110111000001111100000] LOOPNZ MULT1024 ONZERO DONE1024;
56D:
DONE1024:
     [110010110010010111100010] JUMP 57D;
57D:
     [011010110010000111100000] JUMP RUNBUS;
58D:
RUNBUS:
     [101110111000000111110000] JUMP 59D;
59D:
     [101110111000000111000100] RETURN;

END$

***** PART UTILIZATION

60/192      Unconditional Branches ( 31.25%)
3/ 64       Conditional Branches   (  4.69%)

   0 Warnings
   0 Fatal errors
```

Design Simulation

The SAMSIM functional simulator allows simulation of single-, as well as multiple-SAM designs. Once a design has been successfully compiled, the user can specify input stimulus in a variety of formats and observe the device response. SAMSIM supports both hard-copy waveform and tabular output, as well as interactive "virtual logic analyzer" viewing on the PC monitor. Split-window, multiple zoom levels, and delta time display are a few of the capabilities available for analyzing the simulation results in this fashion.

SAMSIM supports both interactive and command file input. Shown in Figure 9 is a sample input stimulus command file for this design. Command files are typically given the design name with extension .CMD. In this example, CIRC.CMD is the name of the command file. The first line specifies the source JEDEC files. Note only the primary file name is given and not the extensions. GROUP CREATE creates a group called CF containing 3 signals (C0–C2). By creating this group, the input pattern for the group can be specified in the PATTERN CREATE CF statement immediately following, rather than having to enter each signal's stimulus separately. The PATTERN CREATE statement shows the sequential values the given input (or group of inputs) is to take beginning at the start of the simulation and continuing onward. Hex format (as shown) can be used to streamline group pattern entry further. The notation ( )*n indicates repeat the enclosed stimulus pattern n times. TRACE CREATE creates a trace buffer file CIRC.TRC into which the state of SAM wil be dumped after each simulation step. This information includes internal information such as value on Top-of-Stack, counter value, etc., as shown in Figure 10. TRACE ON turns the trace process on and may be discontinued with a TRACE OFF command later in the command file. SIMULATE 200 specifies a 200 clock simulation is to be run, and finally VIEW enables interactive viewing of the results of the simulation when complete.

Figure 9. Command File



```
JEDEC CIRC
GROUP CREATE CF = C0 C1 C2
PATTERN CREATE CF = (0H)*200
PATTERN CREATE CmdAtt = (0)*5 (1)*2 (0)*193
PATTERN CREATE Sign =     (0)*200
TRACE CREATE CIRC.TRC
TRACE ON
SIMULATE 200
VIEW
```

Other useful commands supported by SAMSIM, but not used in our example include (among others):

SET — Modifies values of internal stack, counter, etc.

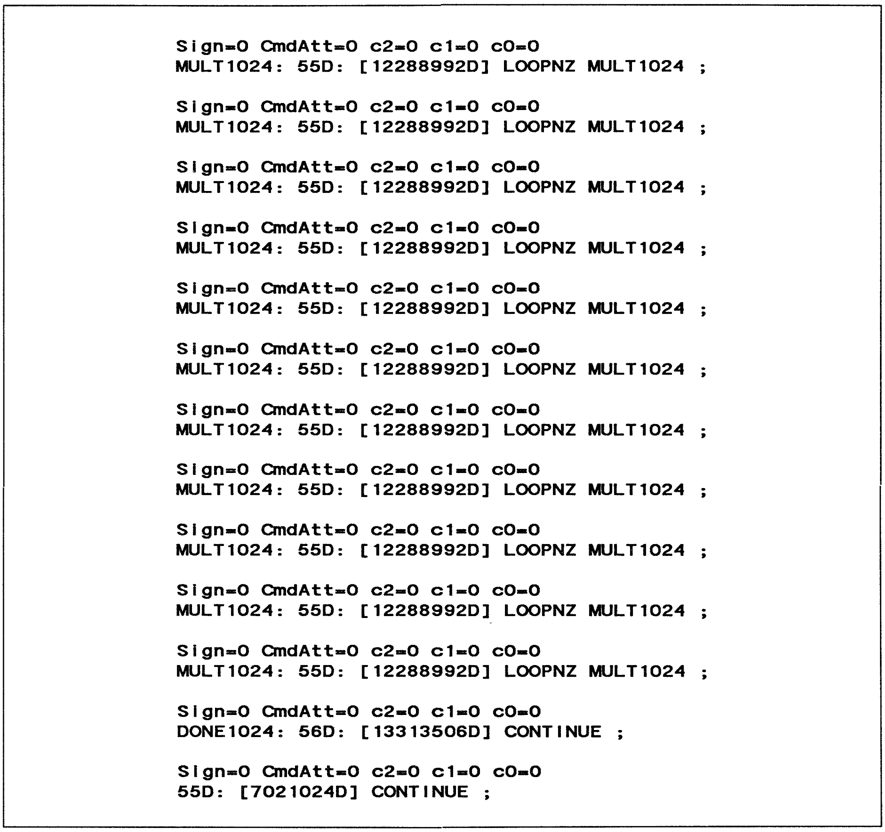
RADIX — Defines default radix for all SAMSIM input. Options are decimal, binary and hex.

LINK — Logically links device pins for simulation purposes.

Running SAMSIM with the above command file gives the output shown in Figure 11.

In reviewing the simulation output figure, a few words of explanation are required. It is immediately apparent that there are two types of output displayed, two examples of which are CmdAtt and AF. CmdAtt is an example of a single signal waveform, in this case corresponding to a device input. AF corresponds to a group of four signals (note the (4) after the name AF) which includes A0–A3. For AF, the values in the group are displayed in a vertical hex notation each time any signal in the group changes. (If an explicit value is not displayed, it is the same as the previous time step's value). By grouping common signals, much more information can be displayed in a single screen than might otherwise be visible. In our example A (AF = A3–A0), B (BF = B3–B0), and I outputs (IL = I2–I0, IM = I5–I3, IH = I8–I6) are viewing groups which have been formed.

Figure 10. Trace File Output



```
Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
DONE1024: 56D: [13313506D] CONTINUE ;

Sign=0 CmdAtt=0 c2=0 c1=0 c0=0
55D: [7021024D] CONTINUE ;
```

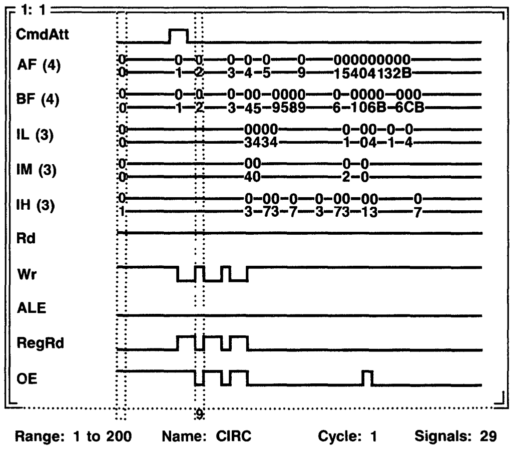
The virtual logic analyzer supports commands which allow the order of waveforms to be changed interactively, arbitrary signal groups to be constructed, among others. An on-line HELP command gives instant explanations for all commands. An extremely flexible interactive analysis tool is the result.

The simulation results shown in Figure 11 correspond to the first 40 or so clocks after the graphics controller receives a CmdAtt signaling the beginning of a circle drawing operation. The three RegRd pulses correspond to reading the circles' radius and x-y origin from the parameter register. The single OE pulse two-thirds of the way across the display is the point where the CircPix routine is first entered. It is left as an exercise to the reader to verify the intermediate output values by following the CIRC.ASM source file.

Conclusion

The SAM device provides an efficient solution for sophisticated control problems such as the graphics controller just described. SAM's capability is applicable to a wide range of problems, including industrial control, graphics and disk controllers, programmable sequence generators and the like. The SAM+PLUS tool set makes the design, verification and debug of such designs straightforward. The combination represents a winning approach to control design.

Figure 11. SAMSIM Interactive Output



Range: 1 to 200 Name: CIRC Cycle: 1 Signals: 29

References

WSI 1990 Data Book

SAM448 Data Sheet

WSI Application Note #4: SAM Applications Using State Design Entry

J. Foley & A. Van Dam, Fundamentals of Interactive Computer Graphics, Addison Wesley, 1981

WAFERSCALE INTEGRATION, INC.

Programmable System™ Device

Application Note 004

SAM Applications Using State Machine Design Entry

Scope of This Application Note

This Application Note is intended to acquaint the user with ASMILE (WSI State Machine Input Language) state machine language syntax as used for entering designs into the SAM448. Basic functionality and syntax is reviewed as well as its use of SAM internal resources. An application utilizing ASMILE input in the form of a 68020 Microprocessor Bus Arbiter is presented. This Application Note provides illustrations of all basic concepts needed to execute a SAM design with ASMILE. For information on microassembler-based entry of SAM designs, please refer to WSI's Application Note #3.

The reader is referred to WSI's SAM448 Data Sheet for details concerning device architecture and performance. A general knowledge of SAM device architecture is assumed as background for this Application Note.

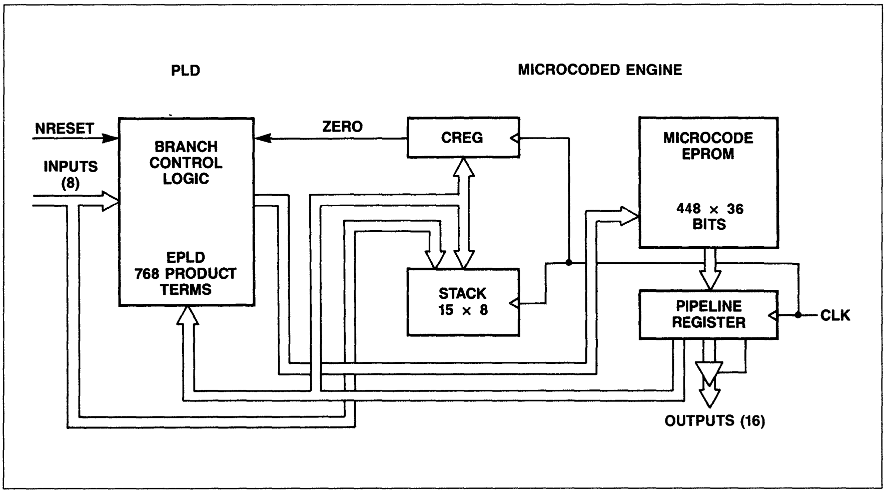
The SAM Solution

WSI's SAM (Stand-Alone Microsequencer) User-Configurable Sequencer Architecture provides a solution for high-performance control functions found in typical digital systems designed today. There have been, previously, two main approaches used in the design of high performance state machine/control functions in digital systems: Logic Array-based sequencers, and microcoded designs. Each approach has presented the designer with a set of benefits and drawbacks to be considered when deciding how to implement a specific application.

Logic Array-based sequencers have been used for very fast state machines of low-to-medium complexity which required few outputs and relatively simple state flows or machine "algorithms." Ability to perform multi-way control branching in a single clock cycle is a plus for this approach. Devices such as conventional registered PLDs are representative of this class. Product term count limitations, resulting in the inability to generate complex output waveforms or state transitions, limits the utility of this approach when addressing larger control problems.

Microcoded approaches have been used for the implementation of complex control

Figure 1. SAM448 Block Diagram



The SAM Solution (Cont.)

functions, requiring high control output counts. Until recently, however, the only mechanism for implementing this approach has been to glue together an assortment of bit-slice component building-blocks. In addition, the approach also did not lend itself to rapid multi-way branching (a strength of Logic Arrays), instead being relegated to a serial test-and-binary-branch mechanism.

An enhanced vehicle for state machine implementation really requires a marriage of these two architectures, to obtain the high performance, multi-way branching based on real-time inputs characteristics of Logic Array-based sequencers, while having the ability to manage complex algorithms and generate high output counts characteristic of microcoded approaches. WSI's SAM448 does exactly this.

SAM+PLUS System Overview

The versatility of the SAM architecture, and its applicability to both State Machine and complex Controller functions, has necessitated the need for multiple design input formats. WSI's SAM+PLUS PC-based Design Software allows the designer to enter his design in either a high-level state machine description using WSI's ASMILE language, or in an efficient microcode assembler format known as ASM. A block diagram of this system is shown in Figure 2. Given these options, the user can employ the design description most appropriate for his particular problem, or which he is personally most comfortable with.

The SAM Design Processor (SDP) takes the input file and automatically minimizes the transition specification logic and fits the resultant resource requests to the SAM architecture. A Utilization Report is generated which reports total resources consumed, any unfittable requests, and assigned pinouts. Upon successful fitting, a standard JEDEC file is generated to allow programming of the device using a hardware programming card installed in the PC.

In addition, this JEDEC file, which represents the actual template of the specific application implemented, may be used as input to the SAMSIM (SAM SIMulator) program which provides functional simulation capability integrated into the total design environment. Hard-copy output of simulation results may be obtained, as well as on-line "logic analyzer" viewing capability. The result is a design entry, compilation and verification system which can be iterated rapidly until the desired functionality is obtained.

Sizing-Up a Potential SAM Design

There are two broad categories of state machines. Mealy and Moore machines (see Figure 3). Given the SAM architecture, one can see that Moore machines may be directly implemented into a SAM component: SAM's outputs are a function of the currently addressed microcode location (state). Mealy machines specify outputs as functions of state and inputs. However, Mealy machines can frequently be converted to equivalent Moore machines. The general rule for this conversion is that for each transition into a state in the Mealy machine with a unique set of outputs, insert a state into the Moore machine with that output combination. Figure 4 illustrates this concept.

ASMILE supports the resources available on SAM for state machine design. Additional feaures, such as the stack and counter, are supported in the microassembler format which lends itself to their efficient use.

In order to determine whether a given application is suitable for SAM, a few brief "rules-of-thumb" derived from the device architecture and specifications can prove helpful:

- Operating frequency less than or equal to specified SAM device's Fmax
- Synchronous, Moore machine operation
- Up to eight state machine inputs (not including CLOCK or RESET)
- Up to sixteen state machine Outputs
- Up to 64 Multi-Way (conditional) state branches
- Transition expressions reducible to four product terms per IF . . . THEN expression
- 192 or fewer unconditional state transitions

An application which meets the above list of requirements will probably fit into a SAM device.

Figure 2. SAM+PLUS System Diagram

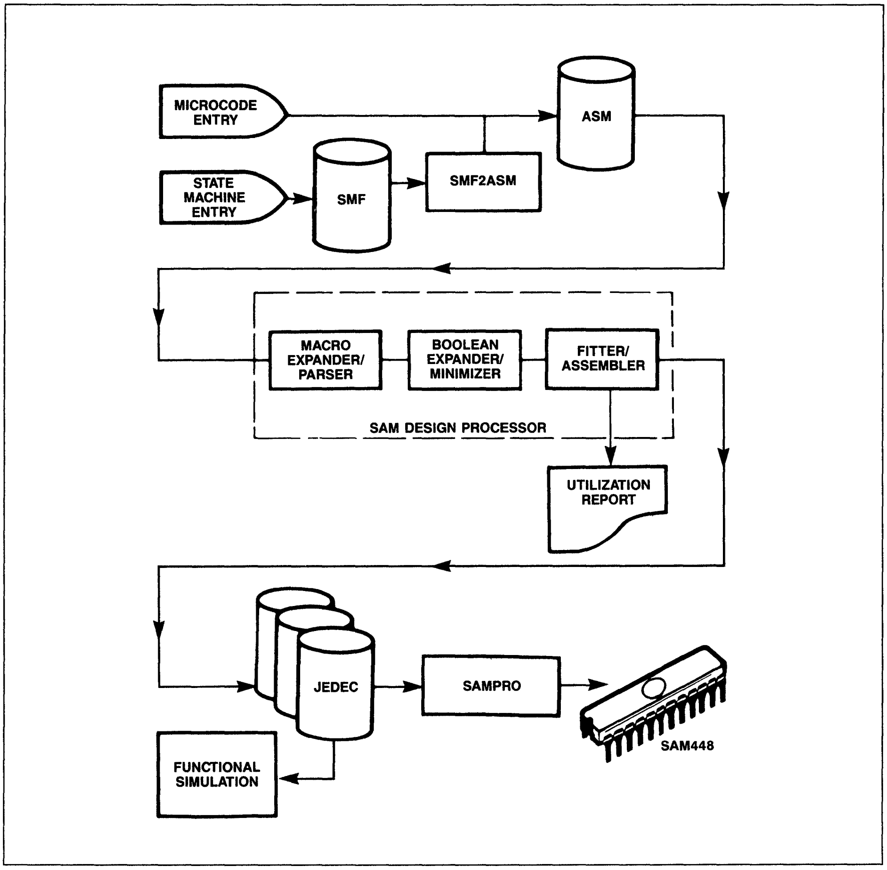


Figure 3. Types of Synchronous State Machines

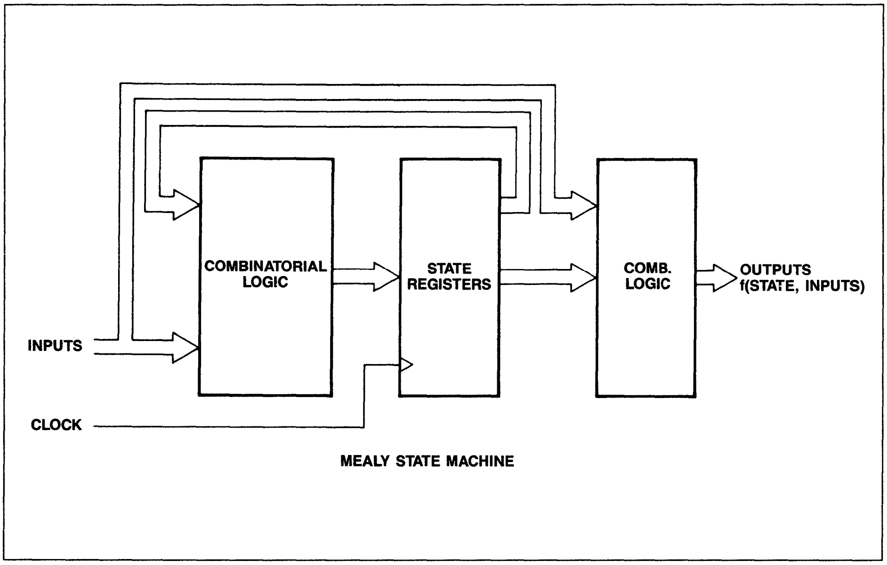


Figure 3. Types of Synchronous State Machines (Cont.)

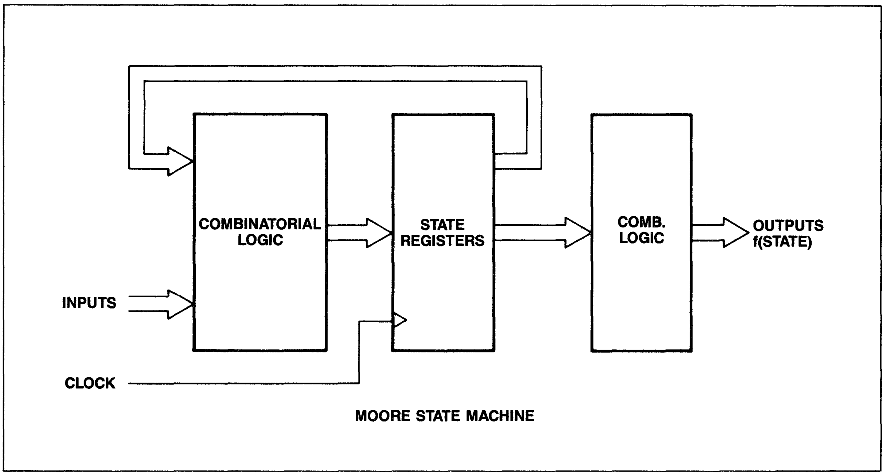
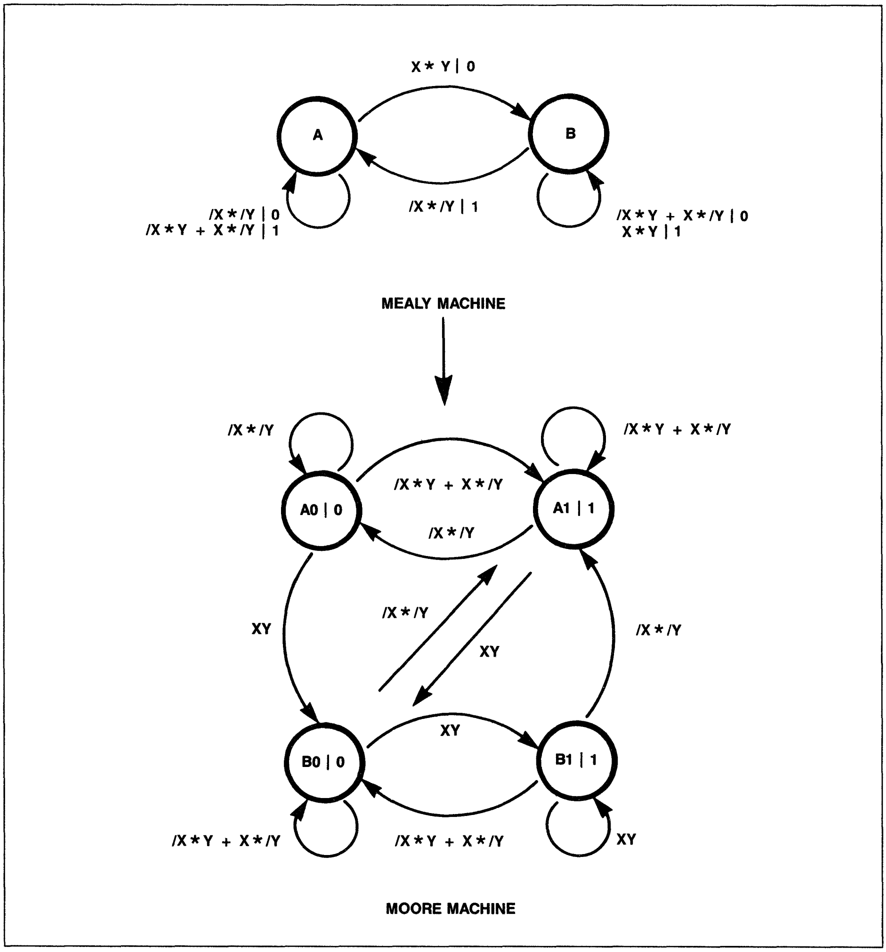


Figure 4. Mealy/Moore Transformation



ASMILE Entry Overview

The basic format of a SAM ASMILE file consists of the following sections:

```
[Header]
 PART
 INPUTS
 OUTPUTS
[EQUATIONS]
 MACHINE
   CLOCK
   STATES
   Transition Specifications
 END$
```

Those sections surrounded by [ ] are optional and may be deleted if their use is not required in a given application.

ASMILE files may be constructed utilizing any standard text editor in non-document mode. Using an editor in document mode may inject spurious format control characters which will be detected as syntax error by the ASMILE parser at compile time. Other than this constraint, input is essentially free-form and may be structured for readability and overall clarity.

The case of characters inserted into the ASMILE file is significant, so it is important to insure that character case is maintained as text is entered. For example, the names "RWB" and "rwb" are not the same.

Comments may be inserted freely into the source code, delimited by leading and trailing percent signs, for example,

```
% This is a comment %
```

Header

The header contains user-specified design identifier information. Typical information includes:

Designer's Name
Company
Date
Design Number
Revision
SAM Part Number
Other Comments

Part

The PART section of the ASMILE file specifies the target SAM device the application is intended for.

Inputs

The single INPUTS section of the ASMILE file defines all external inputs into the design, as well as any required user pin assignments. Pin assignments are optional and will be assigned by SAM+PLUS if not specified. Pin assignments are specified by the format

```
input_name @ pin_number
```

Outputs

The OUTPUTS section of the ASMILE file contains a list of all outputs from the design as well as any pin assignments. Pin assignment syntax is similar to input pin assignments.

Equations

The EQUATIONS section of the ASMILE file is available for the definition of intermediate equations to be used later in the design. Entry of transition specifications may be eased by defining intermediate variables initially, and then invoking them during the design. For example,

```
EventClk = I1*/I4 + I3*I6*/I7
```

might be defined in the EQUATIONS section, and then utilized later in an IF . . . THEN statement.

Machine

The format for the MACHINE declaration is

```
MACHINE: machine_name
```

The MACHINE section of the ASMILE file actually specifies the state machine's state, output, and transition definitions required from the SAM device. There are three subsections which are to be included: CLOCK, STATES, and Transition Specifications.

Clock

The CLOCK subsection specifies the clock signal which will act as the synchronous clock source for the state machine and the resulting SAM device.

States

The STATES section specifies all states in the target machine, as well as outputs corresponding to these states. The general form of this statement, when used in a SAM design, is

```
STATES: [output_name_1 . . .
    output_name_n]
state_name [output_value_list]
```

In the above, the output_names are a list of all SAM output names used in the design, separated by whitespace. Following this initial declaration, a list of all

ASMILE Entry Overview (Cont.)

state__names appears, each followed by a binary string in brackets which specifies all output values to be provided when the machine is in that state.

For example,

```
STATES: [A B C D]
     S0 [0 0 0 0]
     S1 [0 1 1 0]
     S2 [1 0 0 0]
     S3 [0 0 0 1]
```

Specifies a machine with four outputs A through D, State S_0 has all outputs low, S_1 takes B and C to logic one, S_2 has only output A high, etc.

Transition Specification

The form of the Transition Specifications in a SAM ASMILE design is

```
state__name: transition__specification
```

Every state in the machine must have a transition__specification which will specify successor states, either unconditionally

```
S0: S2
```

or conditionally using IF . . . THEN statements.

The first state__name encountered in the Transition Specification section will be defined as the initial state of the machine coming out of Reset. As such, it has special significance. Typically, this might be defined as an "inactive" or passive machine state. Other Transition Specifications have no positional significance.

If . . . Then Statements

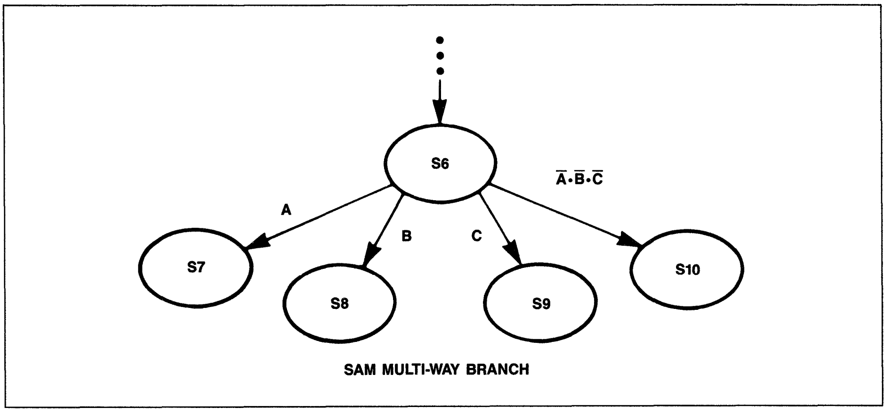
The SAM architecture implements in silicon the state transition specifications defined by a user in the chip's Branch Control logic block. This block allows, by its structure, the specification of up to 64 complex branching expressions in a single machine. [As noted above, up to 192 unconditional state transitions may be specified for a single SAM device]. Each IF . . . THEN expression may specify a direct branch from the current state to as many as four other successor states, based upon inputs to the SAM device. This is illustrated in Figure 5. Examples are shown below.

In specifying IF . . . THEN expressions, it is valuable to note that the order of the expression is important and can determine the machine flow. Transition specifications need not be mutually exclusive in such expressions. For example, the expression

```
S0: IF I1*I2 + I5 THEN S1
    IF I5*I6 + I4*/I3 THEN S2
    IF I4 THEN S3
    S4
```

might appear ambiguous under the condition that inputs I5 and I6 to the SAM device become true during S0. Is S1 or S2 the next state? At this point SAM's priority logic comes into play. Since the S1 transition is specified before the S2 in the design definition, it will be the next state entered. Similarly, if I4*/I3 become valid, S2 will be the next state entered in preference over S3. This precedence-resolving capability is provided in the SAM silicon architecture which employs a hardware priority encoder in selecting the next state transition. This capability resolves conflicts, and may be exploited in the design to prioritize transitions.

Figure 5. SAM Multi-Way Branch



ASMILE Entry Overview (Cont.)

Default Transitions

One other benefit of this approach is the implicit "default" transition to be made. In the example above, S4 will be the next state entered if S1, S2, and S3 are not selected by the appropriate conditions being true. This feature can reduce design effort and resource requirements substantially, since default transitions are frequently defined as the negation of non-default transitions and such inverted expressions have a tendency to consume logic product terms or resources quickly. For example,

```
S0: IF I1*I2 + I5*/I7 + I0 THEN S1
    IF I3 + /I6*I4 THEN S2
    IF I2*I3*I4*I5/I7 THEN S3
    S4
```

is a valid ASMILE SAM transition specification. If the notion of a default transition (S4) was not in the ASMILE syntax, and had to be explicitly defined, we might have to specify the last transition as (unminimized)

```
IF /(I1*I2 + I5*/I7 + I0) * /(I3 + /I6*I4)
    * /(I2*I3*I4*I5*/I7) THEN S4
```

Each expression (IF . . . THEN) may be a function of any of the eight SAM external inputs, and may contain up to four product terms after logic minimization. For most designs, this should prove ample.

A trade-off between number of branch destinations and product terms per destination can be made, as multiple IF . . . THEN expressions can point to the same destination. For example, the expression

```
S0: IF (cond1) THEN S1
    IF (cond2) THEN S1
    IF (cond3) THEN S2
    S3
```

provides a three-way branch, with up to eight product terms available for the specification of transitions to state S1.

End$

Every SAM ASMILE source file must terminate with the END$ terminator.

SAM ASMILE Design Example

To illustrate SAM ASMILE input syntax in a real example, a 68020 Microprocessor Bus Arbiter state machine will be examined. This machine, while not overly complex, illustrates most of the concepts of ASMILE entry.

Shown in Figure 7 is a state machine diagram for the Bus Arbiter. The 68020-based system runs at 25 MegaHertz, and therefore the Bus Arbiter machine must also run with a 40 nanosecond clock period. To understand its operation, a review of the bus exchange protocol used on the 68020 bus is useful.

Three signal lines on a 68020 bus define the handshake required to arbitrate bus exchanges between multiple bus masters: Request, Grant, and Acknowledge. Given a bus master which desires access to the bus, the procedure is as follows (illustrated in Figure 6):

In the above flow description, the state labels S0–S6 designate correspondence between the operations shown and the state machine diagram above.

Relating this sequence to the state diagram, S0 represents the "normal," active state of the processor, S1 and S2 correspond to the Grant phase, S5 and S6 the Acknowledge phase, and S3 and S4 the rearbitration phase if requests are pending at the end of the current bus exchange.

The Design

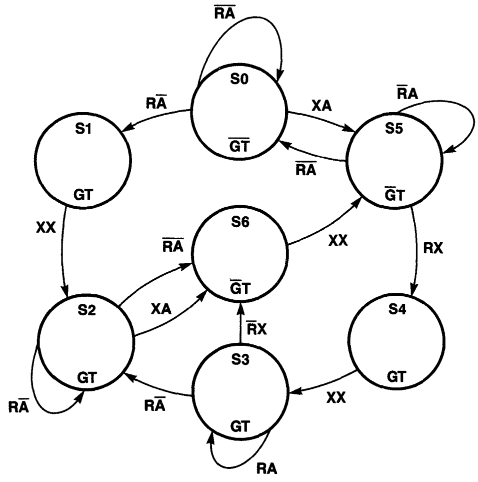
The file shown below in Figure 8 is the actual ASMILE file generated for the machine from the state diagram. It conforms to the general file outline as described above. ASMILE source files are given the extension .SMF (for state machine file) when generated. In this case, the file would be 68020ARB.SMF. Note that in the OUTPUTS and STATES sections, output variables OS0–OS6 have been defined which are each valid only during a unique state. As the design is simulated, these will give an indication of which state the machine is at any given point in time.

To compile this design, the SAM+PLUS software is invoked, specifying that ASMILE (and not microassembler) input format is being used. For a detailed description of

Figure 6. 68020 Bus Arbiter Operation

```
****************************************************************

                 PROCESSOR            REQUESTING BUS MASTER
                 ---------            ---------------------

SO                               * ASSERT REQUEST

S1 & S2
-- - --
* ASSERT GRANT

                                 EXTERNAL ARBITRATION (IF REQUIRED)
                                 AMONG MULTIPLE REQUESTS

                                 * WAIT FOR COMPLETION OF CURRENT CYCLE

S6 & S5
-- - --

                                 * NEXT BUS MASTER ASSERTS ACKNOWLEDGE
                                   (ACK)
                                 * NEXT BUS MASTER DEASSERTS
* DEASSERTS GRANT                         REQUEST
  [WAIT FOR ACK TO BE
  DEASSERTED]

                                 * PERFORM BUS OPERATIONS
                                 * DEASSERT ACK
SO
--

* RESUME OPERATION OR

S4 & S3
-- - --

    RE-ARBITRATE
```

Figure 7. Arbiter State Flow

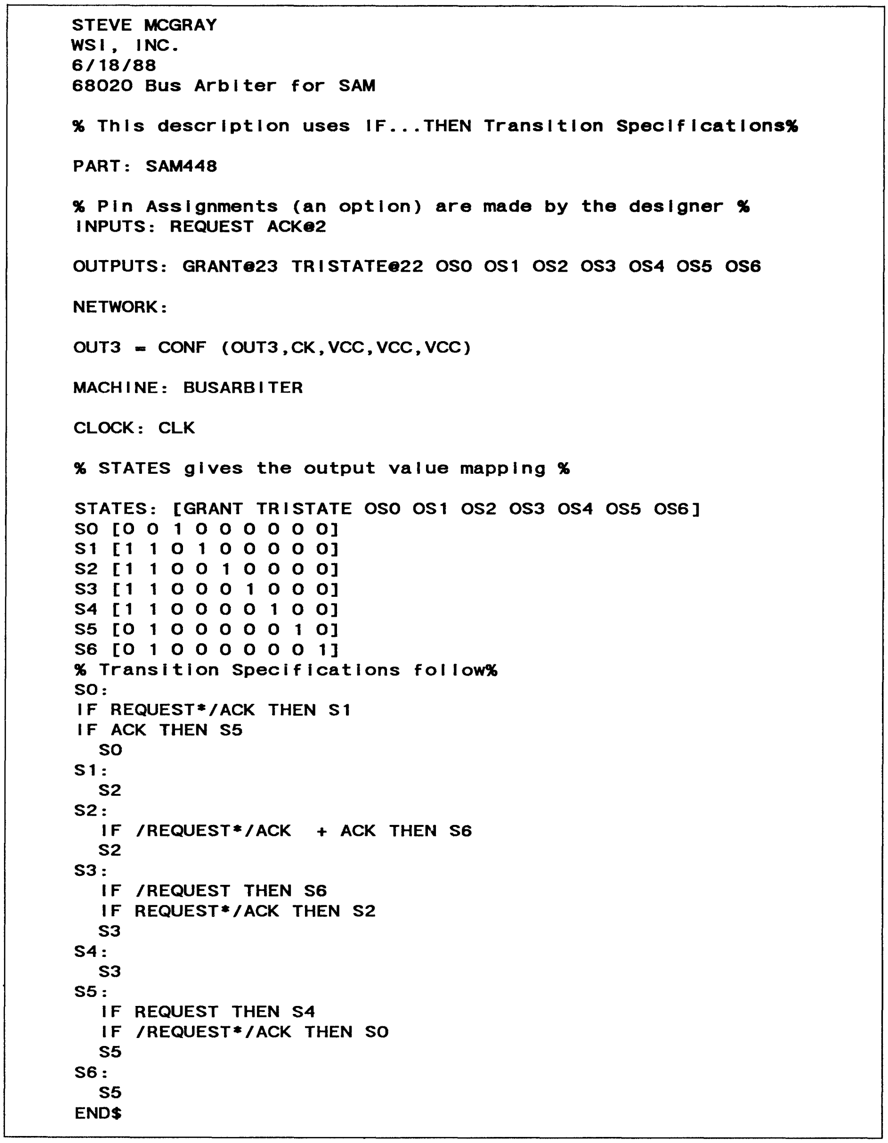


R — BUS REQUEST INPUT
A — BUS GRANT ACKNOWLEDGE INPUT
G — BUS GRANT OUTPUT
T — THREE-STATE CONTROL TO BUS CONTROL LOGIC
X — DON'T CARE

The Design (Cont.)

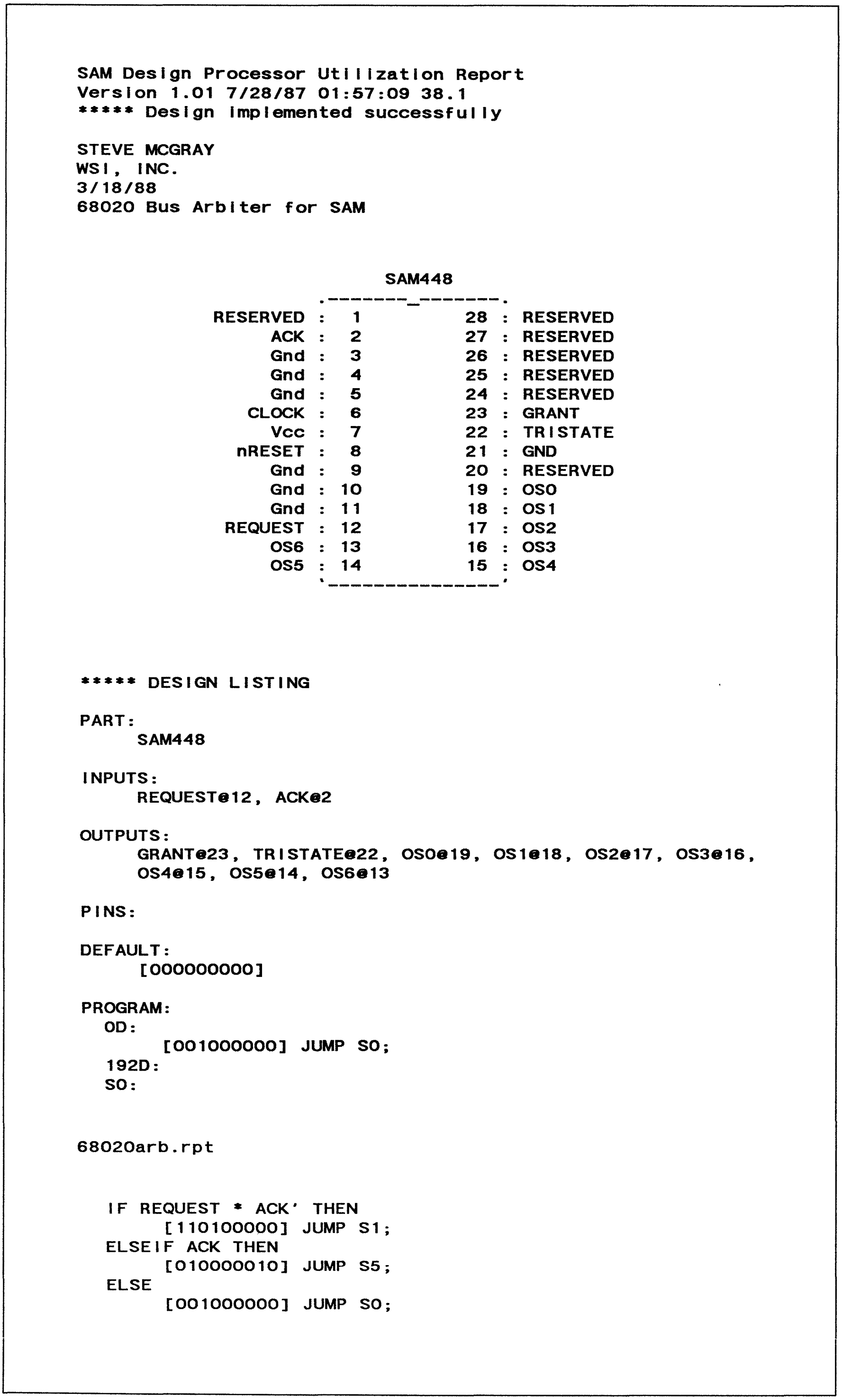
the SAM+PLUS user interface and options, the SAM+PLUS User's Manual should be consulted. Compilation then proceeds automatically. Transition equations are automatically minimized, and "object code" generated for the EPLD and EPROM blocks. As a result, JEDEC programming file (.JED) is generated, as well as a Utilization Report file (.RPT) reporting the results of the compilation process. Functional simulation of the design can be performed using the .JED file as a design template as described below. The .JED file is not intended to be user-readable. The .RPT file contains valuable information such as design pin assignments and resource utilization. Figure 9 shows key portions of this file. All ASMILE input is transformed into microassembler format before subsequent processing, and the equivalent microassembler code for the design is given in the .RPT file as well. More information on the interpretation of this code can be obtained from the references shown below.

Figure 8. 68020 Bus Arbiter State Machine Input File (68020ARB.SMF)



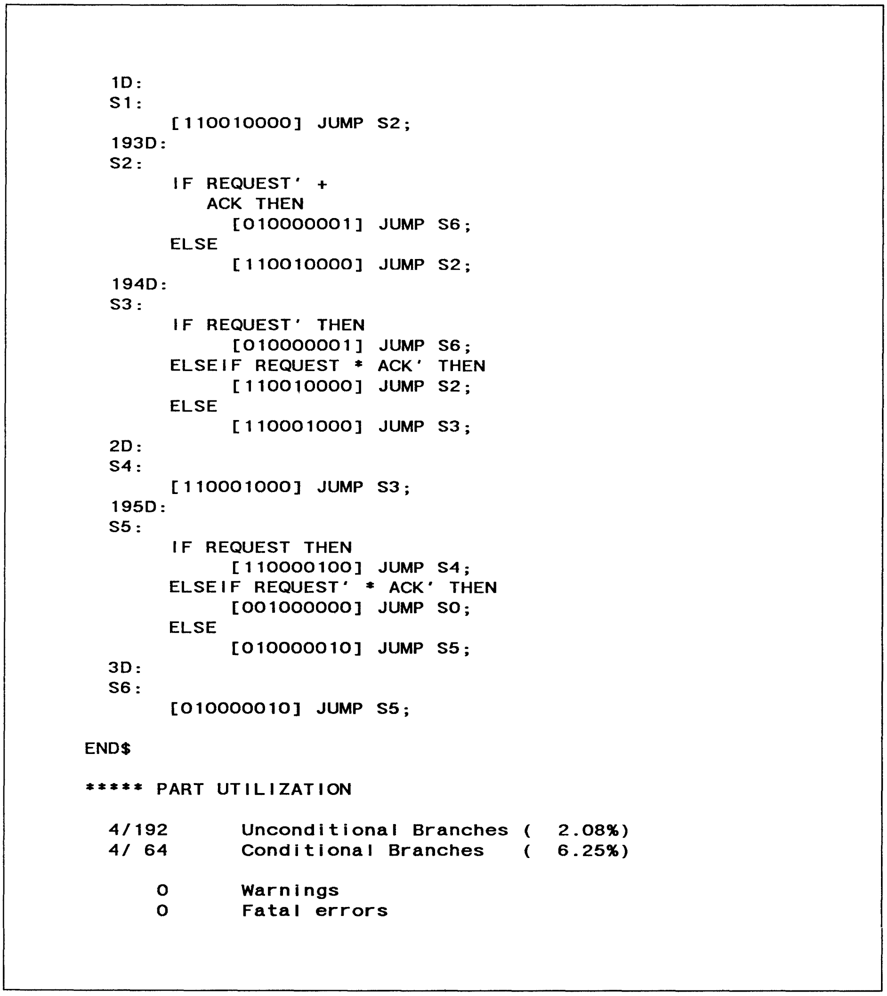
```
STEVE MCGRAY
WSI, INC.
6/18/88
68020 Bus Arbiter for SAM

% This description uses IF...THEN Transition Specifications%

PART: SAM448

% Pin Assignments (an option) are made by the designer %
INPUTS: REQUEST ACK@2

OUTPUTS: GRANT@23 TRISTATE@22 OS0 OS1 OS2 OS3 OS4 OS5 OS6

NETWORK:

OUT3 = CONF (OUT3,CK,VCC,VCC,VCC)

MACHINE: BUSARBITER

CLOCK: CLK

% STATES gives the output value mapping %

STATES: [GRANT TRISTATE OS0 OS1 OS2 OS3 OS4 OS5 OS6]
S0 [0 0 1 0 0 0 0 0 0]
S1 [1 1 0 1 0 0 0 0 0]
S2 [1 1 0 0 1 0 0 0 0]
S3 [1 1 0 0 0 1 0 0 0]
S4 [1 1 0 0 0 0 1 0 0]
S5 [0 1 0 0 0 0 0 1 0]
S6 [0 1 0 0 0 0 0 0 1]
% Transition Specifications follow%
S0:
IF REQUEST*/ACK THEN S1
IF ACK THEN S5
  S0
S1:
  S2
S2:
  IF /REQUEST*/ACK  + ACK THEN S6
  S2
S3:
  IF /REQUEST THEN S6
  IF REQUEST*/ACK THEN S2
  S3
S4:
  S3
S5:
  IF REQUEST THEN S4
  IF /REQUEST*/ACK THEN S0
  S5
S6:
  S5
END$
```

Figure 9. 68020 Bus Arbiter Design Report File (68020ARB.RPT)



```
SAM Design Processor Utilization Report
Version 1.01 7/28/87 01:57:09 38.1
***** Design implemented successfully

STEVE MCGRAY
WSI, INC.
3/18/88
68020 Bus Arbiter for SAM


                                   SAM448
                          .-------_-------.
             RESERVED :  1             28 : RESERVED
                  ACK :  2             27 : RESERVED
                  Gnd :  3             26 : RESERVED
                  Gnd :  4             25 : RESERVED
                  Gnd :  5             24 : RESERVED
                CLOCK :  6             23 : GRANT
                  Vcc :  7             22 : TRISTATE
               nRESET :  8             21 : GND
                  Gnd :  9             20 : RESERVED
                  Gnd : 10             19 : OS0
                  Gnd : 11             18 : OS1
              REQUEST : 12             17 : OS2
                  OS6 : 13             16 : OS3
                  OS5 : 14             15 : OS4
                          `---------------'



***** DESIGN LISTING

PART:
     SAM448

INPUTS:
     REQUEST@12, ACK@2

OUTPUTS:
     GRANT@23, TRISTATE@22, OS0@19, OS1@18, OS2@17, OS3@16,
     OS4@15, OS5@14, OS6@13

PINS:

DEFAULT:
     [000000000]

PROGRAM:
  0D:
       [001000000] JUMP S0;
  192D:
  S0:


68020arb.rpt


   IF REQUEST * ACK' THEN
       [110100000] JUMP S1;
   ELSEIF ACK THEN
       [010000010] JUMP S5;
   ELSE
       [001000000] JUMP S0;
```

Figure 9. 68020 Bus Arbiter Design Report File (68020ARB.RPT) (Cont.)



```
   1D:
   S1:
           [110010000] JUMP S2;
   193D:
   S2:
           IF REQUEST' +
               ACK THEN
                  [010000001] JUMP S6;
           ELSE
                  [110010000] JUMP S2;
   194D:
   S3:
           IF REQUEST' THEN
                  [010000001] JUMP S6;
           ELSEIF REQUEST * ACK' THEN
                  [110010000] JUMP S2;
           ELSE
                  [110001000] JUMP S3;
   2D:
   S4:
           [110001000] JUMP S3;
   195D:
   S5:
           IF REQUEST THEN
                  [110000100] JUMP S4;
           ELSEIF REQUEST' * ACK' THEN
                  [001000000] JUMP S0;
           ELSE
                  [010000010] JUMP S5;
   3D:
   S6:
           [010000010] JUMP S5;

END$

***** PART UTILIZATION

   4/192      Unconditional Branches (  2.08%)
   4/ 64      Conditional Branches   (  6.25%)

         0    Warnings
         0    Fatal errors
```

Design Simulation

Integral to the SAM+PLUS design system is the SAMSIM functional simulator. Once a design has been successfully processed, the user can specify input stimulus in a variety of formats and observe the device response quickly and effectively using this unit-delay simulator. As mentioned above, SAMSIM supports both hard-copy and virtual logic analyzer output formats. Split-window, multiple zoom-levels, and delta time display are a few of the capabilities of this interactive display mode.

SAMSIM supports both interactive and command file input. Shown in Figure 11 is a simple input stimulus command file for our design. Typically command files are given the design name with the extension .CMD (for example, 68020ARB.CMD). The first line specifies the source design JEDEC (or .JED) file. The next two lines illustrate logic sequences for the two machine inputs. The PATTERN CREATE command allows the user to specify a sequence of input logic levels to be applied to the indicated node or nodes. The notation ( )*n, where n is an integer, signifies hold the indicated logic value on the associated input for n clocks. SIMULATE 41 instructs SAMSIM to run the simulation for 41 clocks, and finally interactive display is invoked with the VIEW command.

Design Simulation (Cont.)

Some other representative SAMSIM commands, while not used in the example, include:

TRACE — Dumps entire state of machine (inputs, outputs, internal registers, etc.) for each clock executed.

GROUP — Specifies logical grouping of signals for easy observation or input vector specification.

SET — Modifies values of internal counter, stack, etc.

LINK — Logically links device pins for simulation purposes.

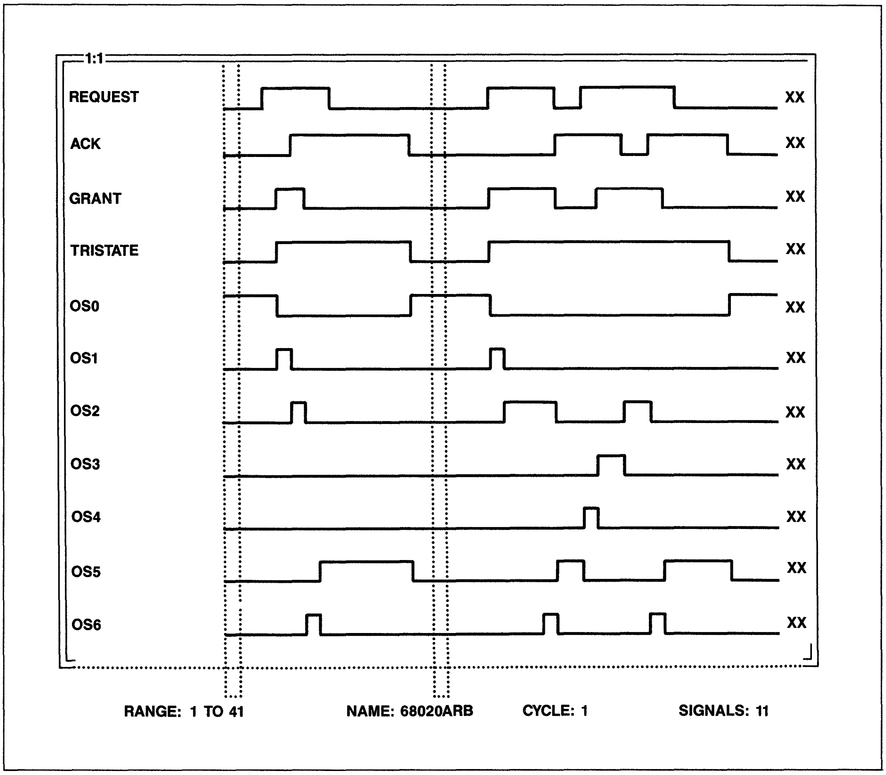
RADIX — Defines default radix for all SAMSIM commands.

Options are binary, hex, and decimal.

Running the SAMSIM simulator with this command file produces the results shown in Figure 10. Here, on the PC screen, is displayed the input stimulus to the SAM arbiter design, and the resulting state machine operation.

The initial input stimulus applied to the SAM design shows a straightforward bus exchange between the 68020 and another bus master. This corresponds to the first REQUEST/GRANT/ACK sequence. Upon detecting a REQUEST, the 68020 asserts its TRI-STATE line, and issues a GRANT pulse, allowing the new bus master to assume control. The alternate bus master asserts ACK when it detects the fact that the bus has been granted. When ACK finally drops, the 68020 knows it can resume control. The second such sequence involves not just a single initial REQUEST (bus master #1), but a second REQUEST from another bus master (#2) during the time bus master #1 has control. As a result, the 68020 must generate a new GRANT pulse (during S4–S2), and hand-over bus control to bus master #2 when bus master #1 is finished (ACK is dropped). When bus master #2 is finished, and no requests are pending, the 68020 finally retakes control of the bus (TRI-STATE goes low).

Figure 10. SAMSIM Interactive Output



```
RANGE: 1 TO 41          NAME: 68020ARB     CYCLE: 1          SIGNALS: 11
```

Figure 11. SAMSIM Command File (68020ARB.CMD)

```
JEDEC 68020AARB
PATTERN CREATE REQUEST = (0)*3 1 1 1 (0)*12 1 1 1 1 1 0 0 (1)*7 (0)*5
PATTERN CREATE ACK =     (0)*5 (1)*8 (0)*10 (1)*6 (0)*2 (1)*6 (0)*4
SIMULATE 41
VIEW
```

Conclusion

State machine design is a straightforward process using the ASMILE input language in conjunction with the SAM device. Design entry and debug, using functional simulation, can be readily accomplished at the user's PC. When the design is debugged and complete, the SAM component may be programmed using PC-based hardware and software in seconds. Should design errors be detected after in-system test, a windowed SAM device may be erased, a design change compiled, and the device reprogrammed in minutes.

References

WSI 1990 Data Book

SAM448 Data Sheet

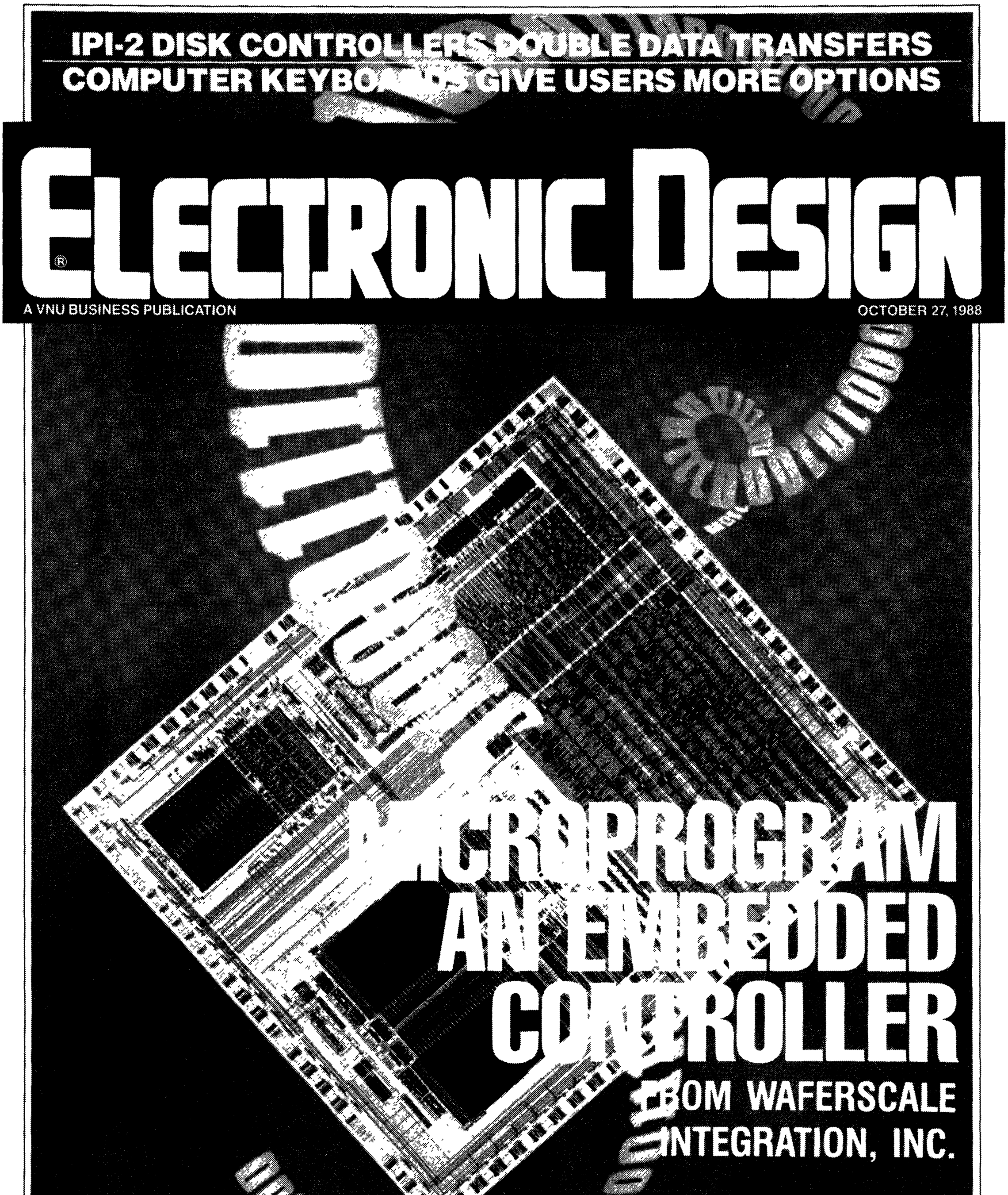
WSI Application Note #3: High-End SAM Applications Using Microassembler Design Entry

WAFERSCALE INTEGRATION, INC.

Section Index

Article Reprint

***For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.***



IPI-2 DISK CONTROLLERS DOUBLE DATA TRANSFERS

COMPUTER KEYBOARDS GIVE USERS MORE OPTIONS

ELECTRONIC DESIGN®

A VNU BUSINESS PUBLICATION

OCTOBER 27, 1988

MICROPROGRAM AN EMBEDDED CONTROLLER

FROM WAFERSCALE INTEGRATION, INC.

- ANALOG CAE GEARS UP FOR MIXED-MODE SIMULATION
- POWER-FACTOR CONTROL CIRCUITS
- RUN TWO STEPPERS FROM ONE BOARD

5

COVER FEATURE

Packing All The Major Blocks Of A Microprogrammable System, A CMOS IC Eases Embedded Controller Designs

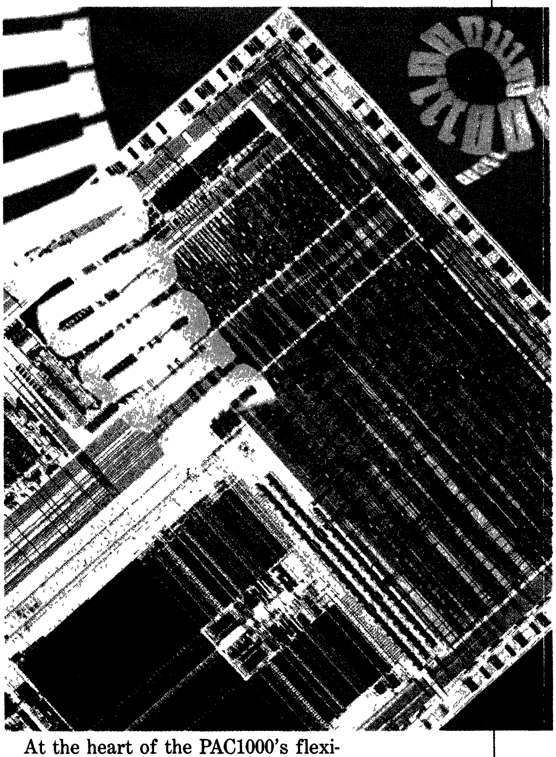
Configurable Chip Eases Control-System Design

Dave Bursky

Anyone who has ever designed a high-performance controller subsystem using high-speed microprogrammed building blocks, programmable logic devices, gate arrays, or discrete logic realizes the difficulties in integrating the complete solution. In such a system, the chip count escalates, the operating power rises, and the development schedule lengthens.

By integrating all these functions and resources onto one high-speed CMOS chip—the PAC1000 microcontroller—WaferScale Integration Inc. has drastically reduced the chip count from the typically required 50 or so ICs to just one. At the same time, the PAC1000 slashes the power consumption from tens of watts to less than 1.5 W and cuts development time.

The PAC1000 can solve many high-end embedded control applications and is the only available circuit that can tackle system, data, and event control tasks. A C-like language and PC-hosted system-development tools simplify the creation of the control software. Users can configure the circuit as a microprocessor peripheral or as a standalone controller to meet the unique requirements of high-performance system, data, or event controllers. Each of the chip's two bidirectional 16-bit buses, its individual I/O lines, and interrupt inputs can, if necessary, be redefined during each 50-ns instruction cycle.



At the heart of the PAC1000's flexibility lies an internal microprogrammable architecture, including a 16-bit CPU, a fast 10-bit microsequencer, a 32-word-by-16-bit register file, and a 1kword-by-64-bit high-speed EPROM. As product planning manager Yoram Cedar explains, since the circuit executes any of its instructions in one clock cycle, the controller delivers a raw throughput of

Reprinted with permission from **ELECTRONIC DESIGN** - October 27, 1988

COVER: USER-CONFIGURABLE CONTROLLER

20 MIPS.

Every instruction of the PAC1000 can perform as many as three simultaneous operations: program control, CPU functions, and output control, with all possible combinations allowed. Cedar claims the more powerful instruction format, combined with the higher clock speed, yields a five- to tenfold performance improvement, compared with other one-chip microcontrollers. The high throughput suits many tasks well. It has already found homes in radar, communications, video-graphics, I/O subsystems, bus and DMA controllers, and disk-drive-controllers.

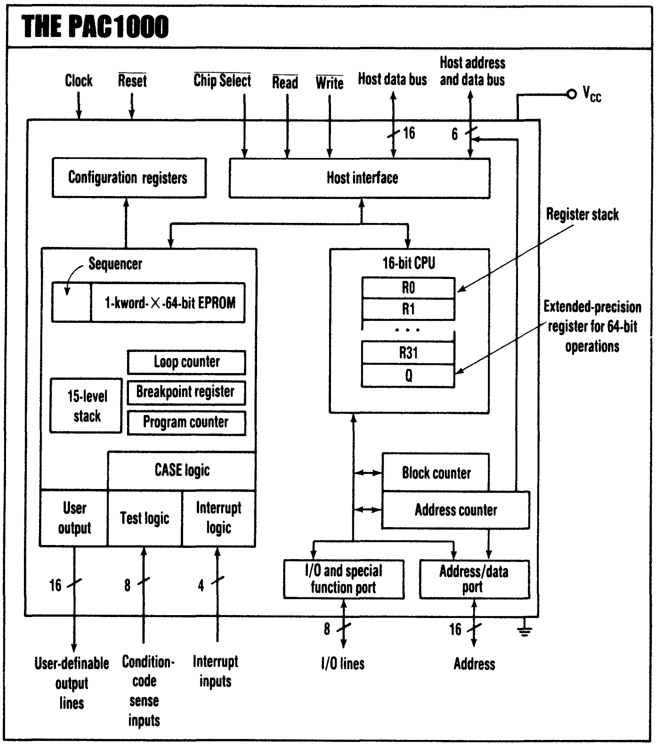
Besides the CPU, register file, and sequencer, the chip includes an auxiliary Q-register for double-word operations, an 8-input interrupt controller, 16 output control lines, 8 bidirectional I/O lines, scan-test and CASE program test logic, and a 22-bit external address bus *(Fig. 1, top)*.

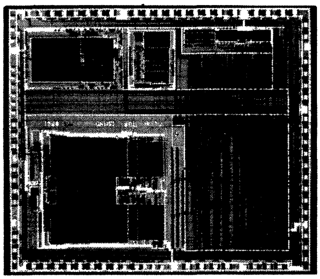
Also, Cedar emphasizes, the circuit deals much more rapidly with interrupts than most controllers do, and that serves embedded control applications well. The chip changes program flow in either of two ways. First, it has four user-definable interrupt input lines plus four dedicated internal interrupts that require just 100 ns, at most, to alter the program flow. Second, another set of input lines—22 condition-code inputs (8 external and 14 internal)—let the processor alter the program flow with condition calls and program jumps in just one 50-ns instruction cycle.

And if on-chip resources don't quite match an application's requirements, chip modifications can be done for large-volume users. The circuit was designed with the company's standard-cell library, and many of the chip's sections are actually cells in WaferScale's library *(Fig. 1, left)*. Noticeable on the chip's left side are the large cells that include the 64-kbit EPROM block on the bottom and the 16-bit CPU on the upper left. On the chip's right side, random logic performs the control and interface functions; small standard cells are used to create those circuits.

For every instruction, a dedicated field specifies the bit pattern on the output lines. Also, designers can individually program eight I/O lines as inputs or outputs or to perform special functions under the control of the chip's mode and I/O registers. The special functions turn the I/O lines into control signals that allow various features and flags to indicate several status conditions. In addition to the eight I/O lines, the circuit has two 16-bit bidirectional buses that go on and off the chip: One links with the host; the other is the upper 16 bits of the address/data bus. Another 16 lines are dedicated, user-programmable latched output lines. These can be changed on a cycle-by-cycle basis.

Thanks to all its buses and control signals, the PAC1000 microcontroller operates as either a memory-





1. PACKING A 16-bit microprogrammable central processor with a 32-word register file, a 1-kword-by-64-bit microcode UV EPROM, sequencer, and other configurable resources, the PAC1000 user-configurable microcontroller from WaferScale Integration delivers a raw instruction throughput of 20 MIPS at 20 MHz (top). Designers can add or alter various blocks to customize versions for high-volume users (left).

COVER: USER-CONFIGURABLE CONTROLLER

mapped peripheral to a microprocessor to offload the CPU *(Fig. 2a)* or as a standalone controller running from its own internally or externally stored program *(Fig. 2b)*. As a peripheral, the chip ties into the host with a straightforward bus interface—a 16-bit data bus and a 6-bit address bus to access the internal resources of the PAC1000—and the standard Chip Select, Read, and Write control lines. In the standalone mode, the chip typically runs the application program from its internal memory and uses its 16-bit output bus and 8-bit I/O port to control the application and communicate to a host system.

To handle multiple operations in parallel, the chip internally takes advantage of a long—64-bit—microcode word so that each word can control multiple sections of the circuitry. The on-chip microcode storage area consists of a fast, reprogrammable UV EPROM, organized as 1 kword by 64 bits. Since the EPROM is read only by the on-chip logic, it doesn't need high-current output buffers, which slow down the memory access. Thus, the EPROM contents can be read very quickly—the chip's 20-MHz version accesses memory in just 30 ns, well within the CPU's 50-ns instruction cycle time. The memory is also secure. Users can program a security bit to prevent an external system from extracting the code from the memory array.

Besides its own program memory, the chip also has a separate address/data bus that can be programmed for either 16 or 22 address lines (with 64-kword or 4-Mword off-chip addressing ranges, respectively). The address generator for the bus is separate from the sequencer that addresses the program memory. The PAC1000 can therefore execute a program while it's using the address bus to move data from memory into the on-chip register file or to an externally controlled device.

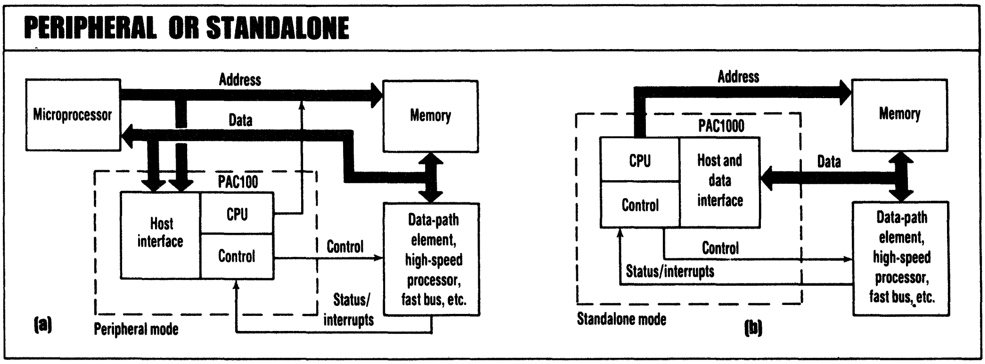
The address bus, in fact, can serve as a simple direct-memory-access controller when used with the on-chip 22-bit address counter and 16-bit block counter. This DMA controller can transfer data from external memory to the on-chip register file or to an external device.

An eight-word FIFO register lets a host microprocessor asynchronously load commands or data into the controller. The 22-bit word length in the FIFO register is employed, so that if data values are to be loaded into the register file, the lower 16 bits of the 22-bit word sent over the host data bus represent the data, and the next five bits—the lower five bits of the host-interface address bus—represent the register location into which the data will be loaded (R0 to R31). The sixth bit of the host-interface address bus signifies whether the word loaded into the FIFO register is a command or data word. If it's a command, the lower 10 bits of the host-data bus are used as a branch address to one of the 1024 memory locations in the EPROM.

The 10-bit sequencer addresses the 1,024 words of program memory and has a 15-level stack that permits multiple subroutine calls to occur without forcing the program to go back to a higher level before calling the next subroutine. Besides having more levels in the stack than WaferScale's 5910 microsequencer, the enhanced sequencer block has a 10-bit loop counter that cuts overhead in programs for loops and nested loops. The application program can load the counter with a constant or a value calculated in the CPU.

Because programming fast, embedded controllers can get complicated, the company includes on-chip programming and test features to ease system development. For starters, a 10-bit breakpoint register simplifies real-time debugging. It can be loaded from either of two sources—a value stored in a CPU register or a constant value specified in the program memory. When the program memory address matches the register contents, the register issues an interrupt, which a service routine in memory could then react to.

Test and CASE logic on the chip also aids program and hardware testing. The condition-code logic responds to 22 different program test conditions that can be tested for true



2. MULTIPLE BUSES, AN ON-CHIP ADDRESS GENERATOR, **and sequencer blocks let the microcontroller operate as a memory-mapped peripheral to offload the host microprocessor (a). Or it can be operated as a standalone controller (b).**

COVER: USER-CONFIGURABLE CONTROLLER

SAMPLE PROGRAM FOR PAC1000 MICROCONTROLLER

```
/* control memory read/write based on CC0 */
segment memcon ,
  enmem   equu   h'0002' ,                          /* output control constants          */
  dismem  equ    h'0040' ,
  wr      equ    h'0000' ,
  rd      equ    h'1000' ,
start
  IF CC0 , OUT enmem ,                              /* enable memory                     */
     FOR 6 , AOR  = R0 + R1 , OUT wr ,              /* store begin addr in AOR and loop  */
             AOR  = AOR + 4 , OUT rd ,              /* inc addr by 4 and do rd/wr        */
     ENDFOR , OUT wr ,                              /* end loop body                     */
  ELSE , OUT dismem ,                               /* disable mem if CC0 is not true    */
  ENDIF ,
end ,
```

3. THE HIGH-LEVEL LANGUAGE developed by WaferScale employs C-language-like structures to let designers easily develop complex configuration microcode.

or not-true results. Up to four conditions can be tested simultaneously. Tests can check for the state of various flags or register contents.

The processor handles two types of CASE operations: standard and priority. A CASE group consists of a combination of four test conditions that can be tested in a single cycle. In that same cycle, the PAC1000 branches to any one of 16 locations, depending on the status of the four inputs to the CASE group being tested. The priority CASE instruction operates on internal and external interrupt conditions and treats interrupts as prioritized test conditions. The priority encoder generates a branch to the highest-priority condition.

Thanks to all its on-chip resources, the PAC1000 is a powerful one-chip controller, housed in a windowed, 88-lead pin-grid-array package or an 84-lead ceramic leaded chip carrier. An 84-lead plastic leaded chip carrier package (the one-time-programmable version) is also available. Because the chip employs an EPROM to hold the program, revisions to the code are no more difficult than reprogramming a standard EPROM. Prototype systems and production products can benefit from the ability to revise the code at the last minute.

To alleviate the complexity of microcode program development, WaferScale has assembled a series of PC-hosted system-development tools (PAC-SDT). These make the PAC1000 as easy to program as any one-chip microcontroller. A simple example of a multiple-command expression in the C-like language lets designers combine operations such as FOR6,AOR=R0+R1,OUT WR (loop for six cycles, add the contents of registers R0 and R1 and store the result in the AOR register, output the value WR) in one word *(Fig. 3)*.

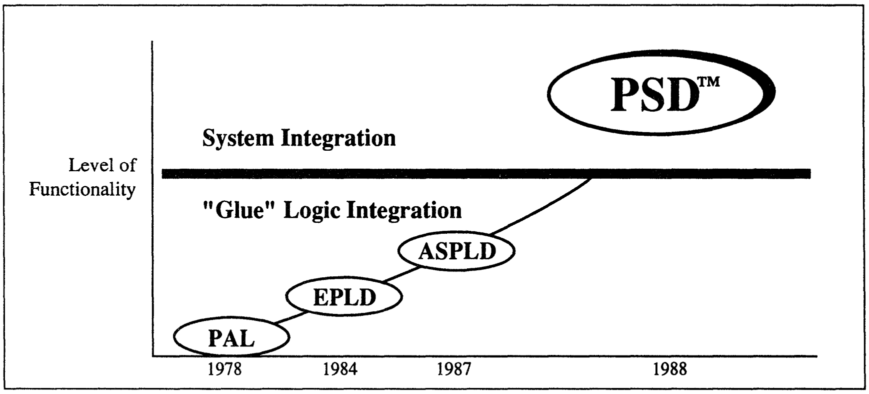
The toolset has a system-entry language, a functional simulator, and a device programmer (MagicPro). The system-entry language software is the most critical part. The high-level language uses a structure similar to C's and practically eliminates writing routines in machine or assembly code. But designers who are more comfortable working on that level can write machine-code routines.□

WSI Launches The Programmable System Device™

A new class of user-configurable products; a higher standard in functionality, integration, and performance.

PSD:™ *n* Programmable System Device.™
1) A user-configurable system-on-a-chip, integrating high-performance EPROM, SRAM, and Logic; **2)** User configurable with a menu-driven, familiar "C"-like language and IBM-PC®-hosted system development tools; **3)** A standard product first launched in 1988 by WSI.

WSI's PSD™ Products: **A Major Advance in user-configurability**



Not just programmable logic, but programmable logic *and* memory—***programmable systems.***

WAFERSCALE INTEGRATION, INC.

47280 Kato Road
Fremont, California 94538
800/331-1030, extension 234
In California call:
800/323-3939, extension 234

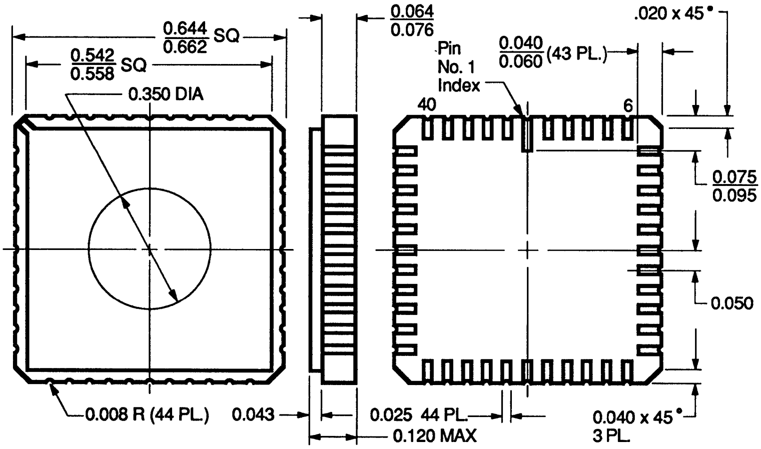
WAFERSCALE INTEGRATION, INC.

Section Index

For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.

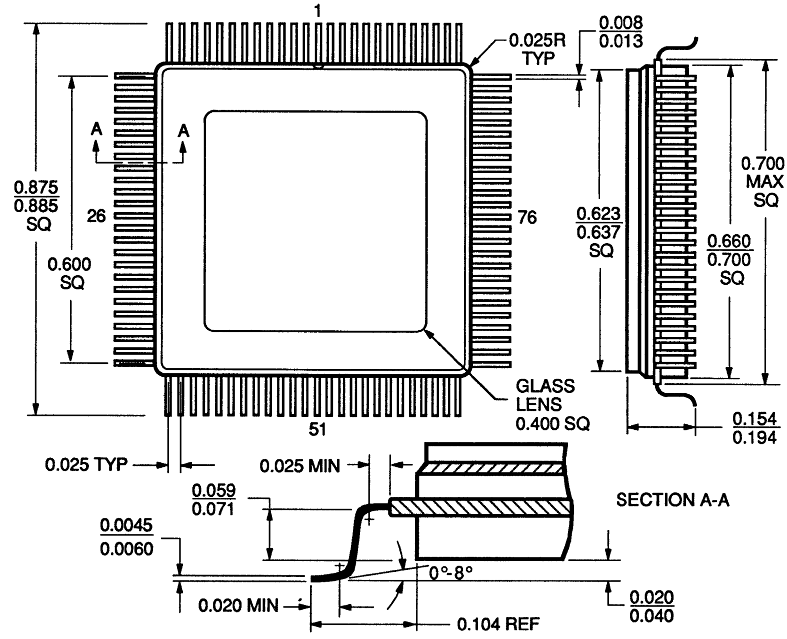
Drawing C3

44 Pad Ceramic Leadless Chip Carrier (CLLCC) (Package Type C)



Drawing F3

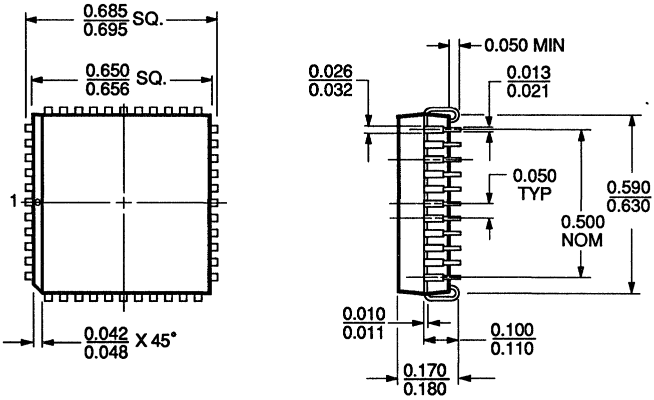
100 Pin Ceramic Quad Flatpack (with Window), Gullwing, Fine Pitch (Package Type F)



Package Information (Cont.)

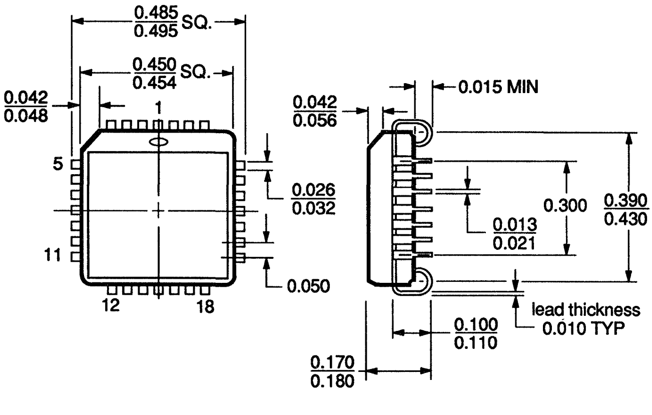
Drawing J2

44 Pin Plastic Leaded Chip Carrier (PLDCC) (Package Type J)



Drawing J3

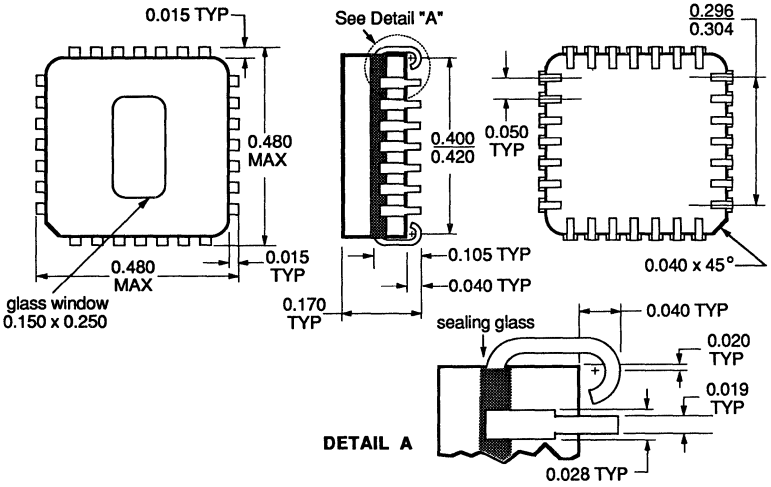
28 Pin Plastic Leaded Chip Carrier (PLDCC) (Package Type J)



Package Information (Cont.)

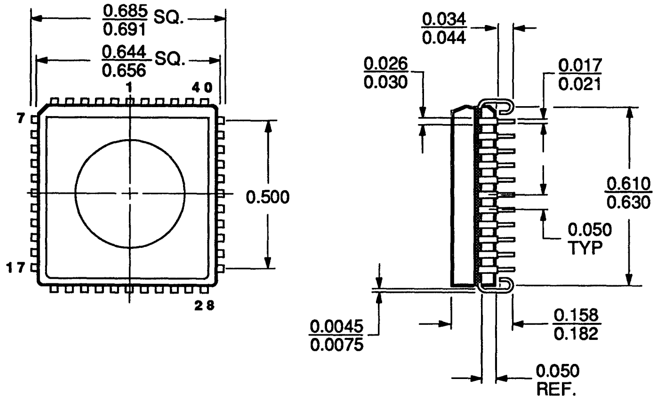
Drawing L2

28 Pin Ceramic Leaded Chip Carrier (CLDCC) with Window (Package Type L)



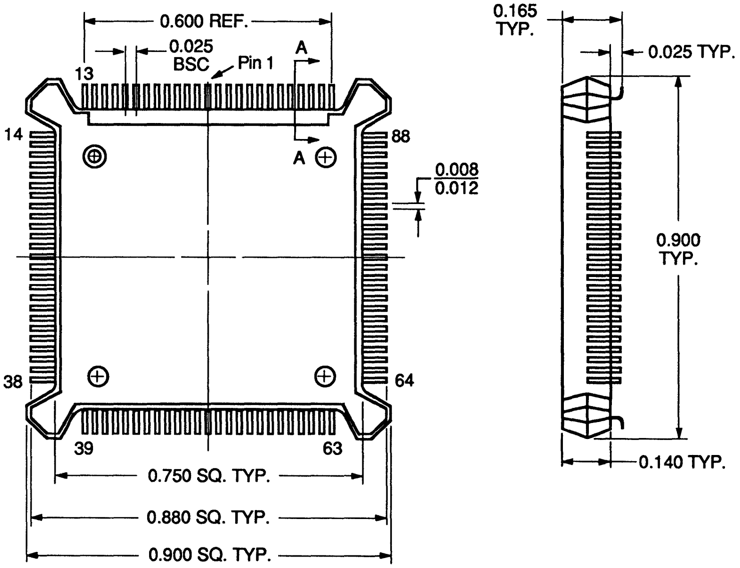
Drawing L4

44 Pin Ceramic Leaded Chip Carrier (CLDCC) with Window (Package Type L)

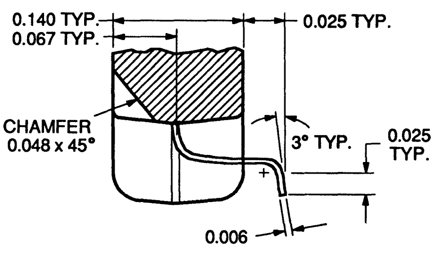


Package Information (Cont.)

Drawing Q1 100 Pin Plastic Quad Flatpack (PQFP), Gullwing, Fine Pitch



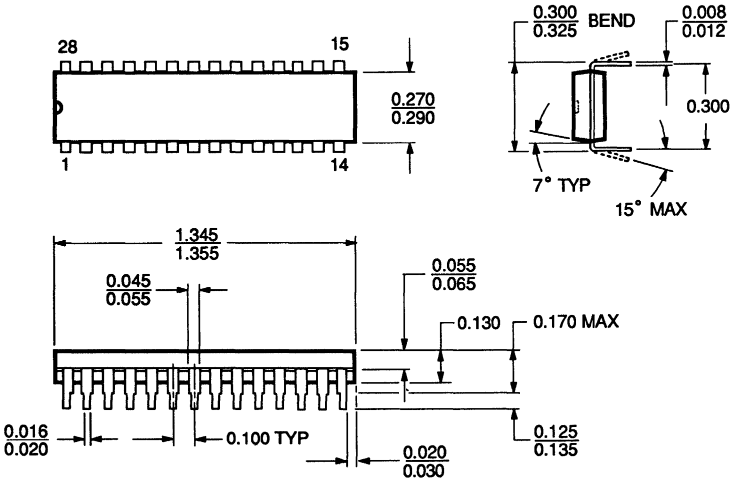
SECTION A–A



Package Information (Cont.)

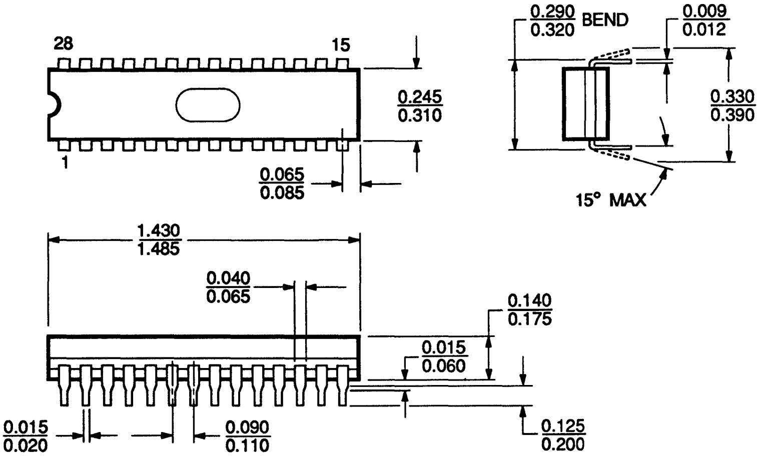
Drawing S2

28 Pin Plastic .300 DIP (Package Type S)



Drawing T2

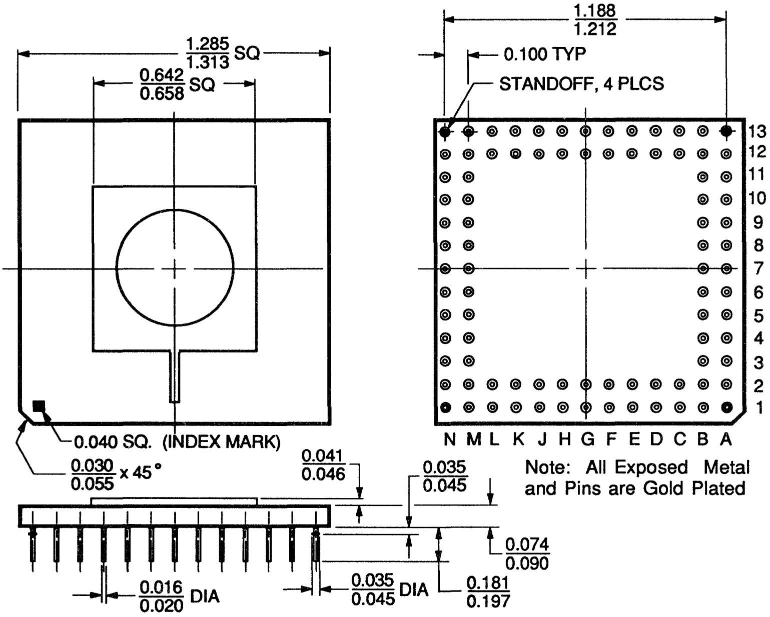
28 Pin CERDIP (Package Type T)



Package Information (Cont.)

Drawing X1

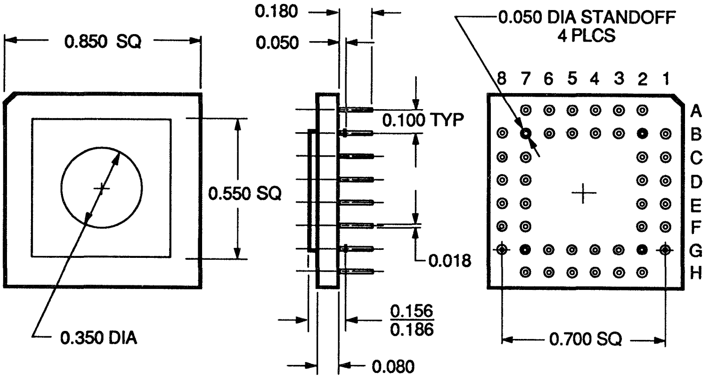
88 Pin Ceramic PGA (Package Type X)



Note: All Exposed Metal and Pins are Gold Plated

Drawing X2

44 Pin Ceramic PGA (Package Type X)



Section Index

For additional information,
call 800-TEAM-WSI (800-832-6974).
In California, call 800-562-6363.

Sales Representatives and Distributors

WAFERSCALE INTEGRATION, INC.

Domestic Representatives

ALABAMA

Southern Tech. Sales
Huntsville
Tel: (205) 539-4789
Fax: (205) 539-7449

ARIZONA

Summit Sales
Scottsdale
Tel: (602) 998-4850
Fax: (602) 998-5274

CALIFORNIA

Bager Electronics Inc.
Fountain Valley
Tel: (714) 957-3367
Fax: (714) 546-2654

Bager Elecronics Inc.
Woodland Hills
Tel: (818) 712-0011
Fax: (818) 712-0160

Earle Assoc. Inc.
San Diego
Tel: (619) 278-5441
Fax: (619) 278-5443

Criterion
Santa Clara
Tel: (408) 988-6300
Fax: (408) 986-9039

Technology Sales
Kentfield
Tel: (415) 459-2661
Fax: (415) 459-3341

CANADA

Har-Tech Electronics, Ltd.
Toronto
Tel: (416) 665-7773
Fax: (416) 665-7290

Har-Tech Electronics, Ltd.
Montreal
Tel: (514) 694-6110
Telex: 05-822679
Fax: (514) 694-8501

Har-Tech Electronics, Ltd.
Ottawa
Tel: (613) 726-9410
Fax: (613) 726-8834

COLORADO

Waugaman Associates, Inc.
Wheat Ridge
Tel: (303) 423-1020
Fax: (303) 467-3095

CONNECTICUT

Advanced Tech Sales
Wallingford
Tel: (203) 284-0838
Fax: (203) 284-8232

FLORIDA

Sales Engineering
Concepts, Inc.
Fort Lauderdale
Tel: (305) 426-4601
Fax: (305) 427-7338

Sales Engineering
Concepts, Inc.
Altamonte Springs
Tel: (407) 682-4800
Fax: (407) 682-6491

Sales Engineering
Concepts, Inc.
Tampa
Tel: (407) 682-4800
Fax: (407) 854-3127

ILLINOIS

Sieger Associates
Schaumburg
Tel: (708) 310-8844
Telex: 206248
Fax. (708) 310-9530

INDIANA

Giesting & Associates
Carmel
Tel: (317) 844-5222
Fax: (317) 844-5861

IOWA

Gassner & Clark Co.
Cedar Rapids
Tel: (319) 393-5763
Twx: 62950087
Fax: (319) 393-5799

KANSAS

C. Logsdon & Assoc.
Prairie Village
Tel: (913) 381-3833
Fax: (913) 381-9774

MARYLAND

Logical Technology, Inc.
Glen Burnie
Tel: (301) 766-7444
Fax: (301) 760-2054

MASSACHUSETTS

Advanced Tech Sales, Inc.
North Reading
Tel. (508) 664-0888
Fax: (508) 664-5503

MICHIGAN

Giesting & Associates
Livonia
Tel: (313) 478-8106
Fax: (313) 477-6908

Giesting & Associates
Coloma
Tel: (616) 468-4200
Fax: (616) 468-6511

MINNESOTA

HMR
Minneapolis
Tel: (612) 988-2122
Fax: (612) 884-4768

MISSOURI

John G. Macke Company
St. Louis
Tel: (314) 432-2830
Fax: (314) 432-1456

NEW JERSEY

Strategic Sales, Inc.
Teaneck
Tel: (201) 833-0099
Fax: (201) 833-0061

S.J. Associates, Inc.
Mt. Laurel, NJ 08084
Tel: (609) 866-1234
Fax: (609) 866-8627

NEW MEXICO

S & S Technologies
Albuquerque
Tel: (505) 255-5599
Fax: (505) 255-5944

NEW YORK

Tri-Tech Electronics, Inc
East Rochester
Tel: (716) 385-6500
Twx: 62934993
Fax: (716) 385-7655

Tri-Tech Electronics Inc.
Endwell
Tel: (607) 754-1094
Twx: 5102520891
Fax: (607) 785-4557

Tri-Tech Electronics Inc.
Fayetteville
Tel: (315) 446-2881
Twx: 7105410604
Fax: (315) 446-3047

Tri-Tech Electronics Inc.
Fishkill
Tel: (914) 897-5611
Twx: 62906505
Fax: (914) 897-5611

NORTH CAROLINA

Rep, Inc.
Morrisville
Tel: (919) 469-9997
Twx: 821765
Fax: (919) 481-3879

Rep, Inc.
Charlotte
Tel: (704) 563-5554
Twx: 821765
Fax: (704) 535-7507

OHIO

Giesting & Associates
Cincinnati
Tel: (513) 385-1105
Fax: (513) 385-5069

Giesting & Associates
Cleveland
Tel: (216) 261-9705
Fax: (216) 261-5624

OREGON

Thorson Company
Northwest
Beaverton
Tel: (503) 644-5900
Telex: 294835
Fax: (503) 644-5919

PENNSYLVANIA

Giesting & Associates
Pittsburgh
Tel: (412) 828-3553
Fax: (412) 828-5861

PUERTO RICO

G & A Associates
Milaville, Rio Piedras
Tel: (809) 758-7001
Fax: 809-754-0421

TEXAS

Southwestern
Technical Sales
Dallas
Tel: (214) 369-0977
Fax: (214) 369-2903

Southwestern
Technical Sales
Austin
Tel: (512) 440-0499

Southwestern
Technical Sales
Houston
Tel: (713) 440-9200

UTAH

Butterworth Marketing
West Valley
Tel: (801) 972-5566
Fax: (801) 972-5573

WASHINGTON

Thorson Company
Northwest
Bellevue
Tel: (206) 455-9180
Twx: 9104432300
Fax: (206) 455-9185

Domestic Distributors

ALABAMA

Schweber Electronics
Huntsville
Tel: (205) 895-0480

ARIZONA

Schweber Electronics
Tempe
Tel: (602) 431-0030

Time Electronics
Tempe
Tel: (602) 967-2000

Wyle Laboratories
Phoenix
Tel: (602) 431-0030

CALIFORNIA

Schweber Electronics
Calabasas
Tel: (818) 880-9686

Schweber Electronics
Irvine
Tel: (714) 863-0200

Schweber Electronics
Sacramento
Tel: (916) 364-0222

Schweber Electronics
San Diego
Tel: (619) 495-0015

Schweber Electronics
San Jose
Tel: (408) 432-7171

Time Electronics
Torrance
Tel: (213) 320-0880

Time Electronics
Sunnyvale
Tel: (408) 734-9888

Time Electronics
Chatsworth
Tel: (818) 998-7200

Time Electronics
San Diego
Tel: (619) 586-1331

Time Electronics
Anaheim
Tel: (714) 937-0911

Wyle Laboratories
Santa Clara
Tel: (408) 727-2500

Wyle Laboratories
Rancho Cordova
Tel: (916) 638-5282

Wyle Laboratories
Irvine
Tel: (714) 863-9953

Wyle Laboratories
Irvine
Tel: (714) 851-9953

Wyle Laboratories
Calabasas
Tel: (818) 880-9001

Wyle Laboratories
San Diego
Tel: (619) 565-9171

CANADA

Time Electronics
2798 Thamesgate Drive, #5
Mississauga,
Ontario L4T 4E8
Tel: (416) 672-5300

COLORADO

Schweber Electronics
Englewood
Tel: (303) 799-0258

Time Electronics
Englewood
Tel: (303) 799-8851

Wyle Laboratories
Thornton
Tel: (303) 457-9953

CONNECTICUT

Schweber Electronics
Oxford
Tel: (203) 264-4700

FLORIDA

Schweber Electronics
Altamonte Springs
Tel: (305) 331-7555

Schweber Electronics
Largo
Tel: (813) 541-5100

Schweber Electronics
North Pompano Beach
Tel: (305) 997-7511

Time Electronics
Ft. Lauderdale
Tel: (305) 974-4800

Time Electronics
Orlando
Tel: (305) 841-6565

GEORGIA

Schweber Electronics
Norcross
Tel: (404) 449-9170

Time Electronics
Norcross
Tel: (404) 448-4448

KANSAS

Schweber Electronics
Overland Park
Tel: (913) 492-2922

ILLINOIS

Schweber Electronics
Elk Grove Village
Tel: (708) 364-3750

Time Electronics
Wooddale
Tel: (708) 350-0610

INDIANA

Schweber Electronics
Indianapolis
Tel: (317) 843-1050

IOWA

Schweber Electronics
Cedar Rapids
Tel: (319) 373-1417

MARYLAND

Schweber Electronics
Columbia
Tel: (301) 596-7800

Time Laboratories
Columbia
Tel: (301) 964-3090

Vantage Components
Columbia
Tel: (301) 720-5100
Tel: (301) 621-8555

MASSACHUSSETTS

Schweber Electronics
Bedford
Tel: (617) 275-5100

Time Electronics
Peabody
Tel: (508) 532-9900

Wyle Laboratories
Burlington
Tel: (617) 272-7300

MICHIGAN

Schweber Electronics
Livonia
Tel: (313) 525-8100

MINNESOTA

Schweber Electronics
Edina
Tel: (612) 941-5280

Time Electronics
Edina
Tel: (612) 835-1250

MISSOURI

Schweber Electronics
Earth City
Tel: (314) 739-0526

Time Electronics
St. Louis
Tel: (314) 391-6444

NEW HAMPSHIRE

Schweber Electronics
Manchester
Tel: (603) 625-2250

NEW JERSEY

Schweber Electronics
Pinebrook
Tel: (201) 227-7880

Time Electronics
Pinebrook
Tel: (201) 882-4611

Vantage Components
Clifton
Tel: (201) 777-4100

NEW YORK

Schweber Electronics
Rochester
Tel: (716) 424-2222

Schweber Electronics
Hauppauge
Tel: (516) 231-2500

Schweber Electronics
Westbury
Tel: (516) 334-7555

Time Electronics
Hauppauge
Tel: (516) 273-0100

Time Electronics
Fairport
Tel: (716) 383-8853
Fax: (716) 383-8863

Time Electronics
East Syracuse
Tel: (315) 432-0355

Vantage Components
Smithtown
Tel: (516) 543-2000

NORTH CAROLINA

Schweber Electronics
Raleigh
Tel: (919) 876-0000

Time Electronics
Charlotte
Tel: (704) 522-7600

OHIO

Schweber Electronics
Beachwood
Tel: (216) 464-2970

Schweber Electronics
Dayton
Tel: (513) 439-1800

Time Electronics
Dublin
Tel: (614) 761-1100

OKLAHOMA

Schweber Electronics
Tulsa
Tel: (918) 622-8000

OREGON

Time Electronics
Portland
Tel: (503) 684-3780

PENNSYLVANIA

Schweber Electronics
Horsham
Tel: (215) 441-0600

Schweber Electronics
Pittsburgh
Tel: (412) 963-6804

Time Electronics
King of Prussia
Tel: (215) 337-0900

TEXAS

Schweber Electronics
Austin
Tel: (512) 339-0088

Schweber Electronics
Dallas
Tel: (214) 247-6300

Domestic Distributors (Cont.)

TEXAS

Schweber Electronics
Houston
Tel: (713) 784-3600

Time Electronics
Carrollton
Tel: (214) 241-7441

Wyle Laboratories
Richardson
Tel: (214) 235-9953

Wyle Laboratories
Houston
Tel: (713) 879-9953

Wyle Laboratories
Austin
Tel: (512) 834-9957

UTAH

Time Electronics
West Valley
Tel: (801) 973-8181

Wyle Laboratories
West Valley
Tel: (801) 974-9953

WASHINGTON

Time Electronics
Redmond
Tel: (206) 882-1600

Wyle Laboratories
Redmond
Tel: (206) 881-1150

WISCONSIN

Schweber Electronics
New Berlin
Tel: (414) 784-9020

International Distributors

AUSTRALIA

Energy Control
Brisbane
Tel: 61-7-376-2955
Fax: 61-7-376-3286
Tlx: 43778

BELGIUM

Inelco
Brussels
Tel: 32 2 216 0160
Tlx: 84-22090
Fax: 32 2 2164606

DENMARK

Distributoren Intereiko, A/S
DK-2690 Karlslunde
Tel: 45-53-140700
Tlx: 85543507
Fax: 45-53-146805

ENGLAND

Mıcro Call Ltd.
Thame, Oxon 0X9 3XD
Tel: 44 84 426 1939
Fax: 44 84 426 1678

FINLAND

OY Comdax AB
SF-00210 Helsingfors
Tel: 358 067 02 77
Tlx: 857125876
Fax: 358 06922326

FRANCE

MICROEL
Imeuble MICRO
Cedex
Tel: 33 (1) 69.07.08.24
Tlx: 692493F
Fax: 33 (1) 69.07.17.23

GERMANY

Topas Electronic GmbH
3000 Hannover 1
Tel: (0511) 13 12 17
Tlx: 9218176
Fax: (0511) 13 12 16

Scantec GmbH
D-33 Planegg
Tel: (089) 859-8021
Tlx: 5213219
Fax: (089) 857-6574

HOLLAND

Maxtronix
Savannahweg 60
3542 AW UTRECHT
Tel: (31) 30-420340
Fax: (31) 30-422440

HONG KONG

Components Agent Ltd.
New Territories
Tel: 0-499-2688
Tlx: 78030398
Fax: 852 0-4136080

INDIA

Pamir Electronics Corp.
400 West Lancaster
Suite 202
Devon, PA 19333 USA
Tel: 215-688-5299
Fax: 215-688-5382
Tlx: 210656 Pamir UR

ISRAEL

Vectronics
60 Medinat Hayehudim St.
P.O. Box 2024
Herzlia B 46120, Israel
Tel: 972 52 556070
Tlx: 922342579
Fax: 972 52 556508

ITALY

Silverstar
20146 Milano
Tel: 39 2 661251
Tlx: 843332189
Fax: 39-2-66101359

JAPAN

Nippon Imex Corporation
Setagaya-ku, Tokyo
Tel: 321 4415
Tlx: 781 23444
Fax: 81 3 325 0021

Kyocera Corporation
Setagaya-ku, Tokyo
Tel: 3-708-3111
Tlx: 7812466091
Fax: 81-3-708-3864

KOREA

Eastern Electronics, Inc.
Sungdong-Ku, Seoul
Tel: 82 2 463-2266
Tlx: 78727381
Fax: 82 2 465-2607

NORWAY

OTE A/S
N-0617 Oslo 6
Tel: 47 2 306600
Tlx: 85678955
Fax: 47 2 321360

SPAIN

Unitronics, S.A.
28008 Madrid
Tel: 34 1 542 5204
Tlx: 83122596
Fax: 34 1 248 4228

SWEDEN

Traco AB
S-123 22 Farsta
Tel: 468 930011
Tlx: 85410689
Fax: 468 947732

SWITZERLAND

Laser & Electronic
Equipment
8053 Zürich
Tel: 41 (1) 55 3330
Tlx: 816801
Fax: 41 (1) 55 3458

TAIWAN

Sertek International, Inc.
Taipei, 10479, Taiwan
Tel: 2-501-0019
Tlx: 78523756

WSI Direct Sales Offices

REGIONAL SALES

Northeast
North Andover, MA
Tel: (508) 685-6101
Fax: 508/685-6105

Midwest
Hoffman Estates, IL
Tel: (708) 490-5318
Fax: 708/882-1881

Southwest
Huntington Beach, CA
Tel: (714) 848-6968
Fax: 714/848-5648

Mid-Atlantic
Trevose, PA
Tel: (215) 638-9617
Fax: 215/638-7326

Southeast
Huntsville, AL
Tel: (205) 539-7406
Fax: 205/539-7449

Northwest
Fremont, CA
Tel: (415) 656-5400
Telex: 289255
Fax: 415/657-5916

EUROPE SALES

Excelsiorlaan 53
1930 Zaventem
Belgium
Tel: 32-2-725-0546
Fax: 32-2-725-1146

For additional information or assistance, call 800-TEAM-WSI (800-832-6974). In California, call 800-562-6363.

LIFE SUPPORT POLICY:
WaferScale Integration, Inc. (WSI) products are not authorized for use as critical components in life support systems or devices without the express written approval of the President of WSI. As used herein:

A) Life support devices or systems are devices or systems which 1) are intended for surgical implant into the body, or 2) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury or death to the user,

B) A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Information furnished herein by WaferScale Integration, Inc. (WSI) is believed to be accurate and reliable. However, no responsibility is assumed for its use. WSI makes no representation that the use of its products or the interconnection of its circuits, as described herein, will not infringe on existing patent rights. No patent liability shall be incurred by WSI for use of the circuits or devices described herein. WSI does not assume any responsibility for use of any circuitry described, no circuit patent rights or licenses are granted or implied, and WSI reserves the right without commitment, at any time without notice, to change said circuitry or specifications. The performance characteristics listed in this book result from specific tests, correlated testing, guard banding, design and other practices common to the industry. Information contained herein supersedes previously published specifications. Contact your WSI sales representative for specific testing details or latest information.

Products in this book may be covered by one or more of the following patents. Additional patents are pending.

USA: 4,328,565; 4,361,847; 4,409,723; 4,639,893; 4,649,520; 4,795,719; 4,763,184; 4,758,869
West Germany: 3,103,160
Japan: 1,279,100
England: 2,073,484; 2,073,487

Patents Pending

Rev. 1.3

WAFERSCALE INTEGRATION, INC.

47280 Kato Road, Fremont, CA 94538-7333
415-656-5400 FAX: 415-657-5916 TELEX: 289255
800-TEAM-WSI (800-832-6974)
IN CALIFORNIA 800-562-6363

Printed in U.S.A.