



PSD

Programmable MCU Peripherals

PSD Applications Handbook

Volume 2 of 2

1996

LORI STEINTHAL

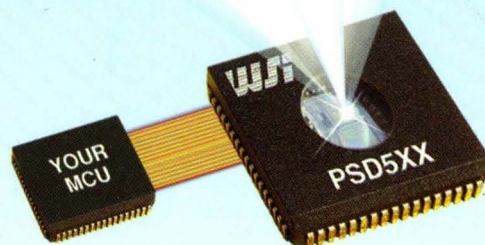
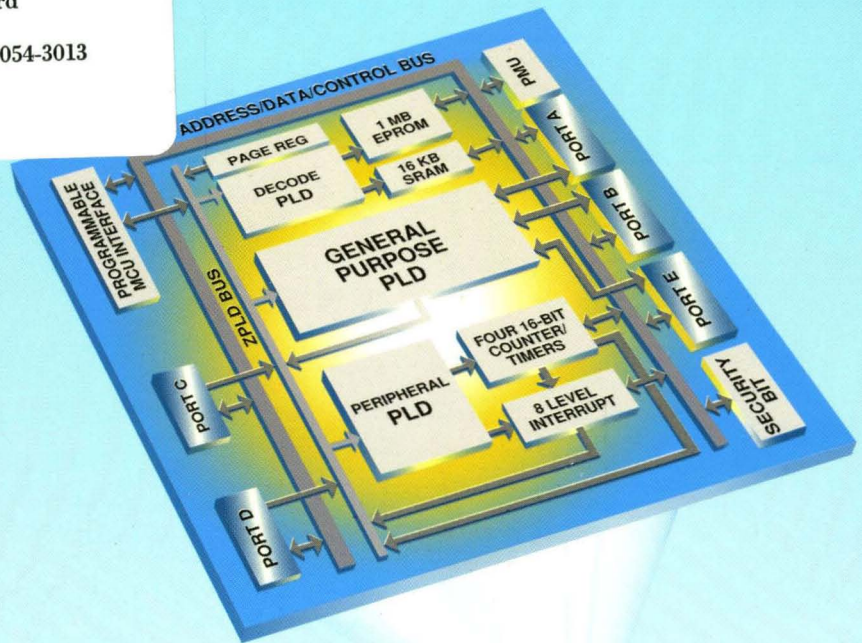


I² INCORPORATED

MANUFACTURERS REPRESENTATIVES

3255 Scott Boulevard
Suite 1-102
Santa Clara, CA 95054-3013

Tel: (408) 988-3400
Fax: (408) 988-2079





PSD
Programmable MCU Peripherals
PSD Applications Handbook
Volume 2 of 2

1996

*Copyright © 1996 WaferScale Integration, Inc.
(All rights reserved.)*

*47280 Kato Road, Fremont, California 94538
Tel: 510-656-5400 Facsimile: 510-657-5916*

*Printed in U. S. A.
on recycled paper.*





Table of Contents

PSD Application Notes

PSD3XX Family	Application Note 011	PSD3XX Device Description	1-1
	Application Note 013	The PSD301 Streamlines a Microcontroller-Based Smart Transmitter Design	1-55
	Application Note 014	Using the PSD3XX PAD for System Logic Replacement.....	1-67
	Application Note 015	Using Memory Paging with the PSD3XX	1-81
	Application Note 016	Power Considerations in the PSD3XX	1-95
	Application Note 017	Track Mode Implementation of PSD3XX	1-111
	Application Note 018	Security of Design in the PSD3XX	1-121
	Application Note 019	The PSD311 Simplifies an Eight Wire Cable Tester Design and Increases Flexibility in the Process	1-125
	Application Note 020	Benefits of 16-Bit Design with PSD3XX	1-145
	Application Note 021	Interfacing The PSD3XX To The MC68HC16 and The MC68300 Family of Microcontrollers	1-157
	Application Note 022	Using WSI's PSD3XX Programmable Microcontroller Peripheral Family with 80C31/80C51 Microcontrollers	1-167
	Application Note 023	PSD3XX Family – Programmable Microcontroller Peripheral Design Tutorial.....	1-179
	Application Note 024	Using the PSD311 with a High-Speed ADSP-2105 DSP	1-191
	Application Note 025	Interfacing The PSD3XX To The NEURON® 3150™ CHIP	1-201
	Application Note 026	PSD3XX Device Fit for PC Notebook Applications: Keyboard, Power Management and Auxiliary Peripherals Control	1-221
	Application Note 027	Simplification of Logic Networks in the PSD3XX PAD Using DeMorgan's Theorem	1-227
	Application Note 032	Use a ROM Emulator for Rapid Software Debug of a PSD3XX-Based Design	1-237
	Application Note 040	Three-Chip Feature Phone	1-251
	Application Note 041	Detailed Step-By-Step Design Implementation of an M68HC11 and PSD311 or PSD311R.....	1-269

ZPSD3XX Family	Application Note 034	ZPSD Power Consumption Calculations.....	2-1
-----------------------	----------------------	--	-----

PSD4XX/5XX Family	Application Note 028	PSD5XX Counter/Timers Operation.....	3-1
	Application Note 029	Interfacing PSD4XX/5XX To Microcontrollers	3-73
	Application Note 030	PSD4XX/5XX Power Calculations and Reduction	3-145
	Application Note 031	PSD4XX/5XX Design Tutorial	3-161
	Application Note 033	Keypad Interface to PSD4XX/5XX with Autoscanning	3-245
	Application 035	How To Design With The PSD4XX/5XX ZPLD	3-257
	Application 036	How To Fit Your Design Into The PSD4XX/5XX	3-265
	Application 037	How to Implement a Latch Function in Port A of PSD4XX/5XX that is Independent of the System Clock.....	3-271
	Application 038	How to Increase the Speed of the PSD5XX Counter/Timers.....	3-277
	Application 039	Encoder for Shaft Direction and Position Recognition Using the PSD5XX.....	3-287
	Application 042	Four Axis Stepper Motor Control Using a Programmable PSD5XX MCU Peripheral from WSI, Inc.	3-297

Motorola Application Notes	<i>The following are Motorola Application Notes and known as Application Notes 043 and 044 at WSI, Inc.</i>		
	Application Note 043	Using M68HC11 Microcontrollers with WSI Programmable Peripheral Devices	4-1
	Application Note 044	High Performance M68HC11 System Design Using The WSI PSD4XX and PSD5XX Families	4-9

Sales Representatives and Distributors	5-1
---	-----

PSD3XX Family	<i>Refer to PSD Products Data Book, Volume 1 of 2, for the Data Sheets listed below.</i> Field-Programmable Microcontroller Peripherals.....2-1
ZPSD3XX Family	Field-Programmable Microcontroller Peripherals.....3-1
PSD4XX Family	Field-Programmable Microcontroller Peripherals.....4-1
ZPSD4XX Family	Field-Programmable Microcontroller Peripherals.....5-1
PSD5XX Family	Field-Programmable Microcontroller Peripherals.....6-1
ZPSD5XX Family	Field-Programmable Microcontroller Peripherals.....7-1
PSD Development Systems and Accessories	PSDsoft8-1 PSD-Gold and PSD-Silver Development Systems8-5 WS6000 MagicPro® III Memory and Programmable Peripheral Programmer.....8-7 Electronic Bulletin Board8-11
Masked-PSD Ordering Information	MPSD Mask-Programmable PSD Ordering Information.....9-1



PSD3XX Family

1

ZPSD3XX Family

2

PSD4XX/5XX Family

3

Motorola Application Notes

4

***Sales Representatives
and Distributors***

5

Section Index

PSD3XX Family

Application Note 011	PSD3XX Device Description	1-1
Application Note 013	The PSD301 Streamlines a Microcontroller-Based Smart Transmitter Design	1-55
Application Note 014	Using the PSD3XX PAD for System Logic Replacement.....	1-67
Application Note 015	Using Memory Paging with the PSD3XX	1-81
Application Note 016	Power Considerations in the PSD3XX	1-95
Application Note 017	Track Mode Implementation of PSD3XX	1-111
Application Note 018	Security of Design in the PSD3XX	1-121
Application Note 019	The PSD311 Simplifies an Eight Wire Cable Tester Design and Increases Flexibility in the Process	1-125
Application Note 020	Benefits of 16-Bit Design with PSD3XX	1-145
Application Note 021	Interfacing The PSD3XX To The MC68HC16 and The MC68300 Family of Microcontrollers	1-157
Application Note 022	Using WSI's PSD3XX Programmable Microcontroller Peripheral Family with 80C31/80C51 Microcontrollers	1-167
Application Note 023	PSD3XX Family – Programmable Microcontroller Peripheral Design Tutorial.....	1-179
Application Note 024	Using the PSD311 with a High-Speed ADSP-2105 DSP	1-191
Application Note 025	Interfacing The PSD3XX To The NEURON® 3150™ CHIP	1-201
Application Note 026	PSD3XX Device Fit for PC Notebook Applications: Keyboard, Power Management and Auxiliary Peripherals Control	1-221
Application Note 027	Simplification of Logic Networks in the PSD3XX PAD Using DeMorgan's Theorem.....	1-227
Application Note 032	Use a ROM Emulator for Rapid Software Debug of a PSD3XX-Based Design	1-237
Application Note 040	Three-Chip Feature Phone	1-251
Application Note 041	Detailed Step-By-Step Design Implementation of an M68HC11 and PSD311 or PSD311R.....	1-269

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



Programmable Peripheral

Application Note 011

PSD3XX Device Description

Contents

Chapter 1: PSD301 Device Description

Introduction.....	1-3
WSI Software Support for the PSD Family.....	1-4
PSD3XX Architecture and Pin Nomenclature	1-4
Performance Characteristics	1-6
PSD3XX System Configuration for Port and I/O Options.....	1-6
Address Inputs	1-8
PSD3XX Programmable Array Decoder (PAD)	1-9
Microcontroller/Microprocessor Control Inputs.....	1-10
Input and Output Ports.....	1-11
PSD3XX General System Configuration	1-13
PSD3XX Configuration for Port Reconstruction	1-15

1

Chapter 2: Applications

8-Bit Microcontroller to PSD3XX Interface	1-17
Two PSD3XX Byte-Wide Interfaces to Intel 80C31	1-19
PSD3XX M68HC11 Byte-Wide Interface.....	1-21
8-Bit Non-Multiplexed PSD3XX Interface to M68008.....	1-23
16-Bit Non-Multiplexed Address/Data PSD3XX Interface to M68000	1-25
M68000/2X PSD3XX Applications	1-27
16-Bit Address/Data PSD3XX Interface to Intel 80186	1-29
16-Bit Address/Data PSD3XX to Intel 80196 Interface	1-31
Interfacing the PSD3XX to 8-Bit Microprocessors Z80 and M6809 Applications	1-33
PSD3XX Interface to the Intel 80286	1-36
External Peripherals to the PSD3XX/M68HC11 Configuration.....	1-38
Additional External SRAM.....	1-40
PSD3XX Used in Track Mode.....	1-44

Chapter 3: Software Support

Summary	1-53
---------------	------

Figures

Figure 1. PSD3XX Supports CPU as a Complete Peripheral, Memory, and Logic Subsystem	1-3
Figure 2. PSD3XX Architecture	1-5
Figure 3. 8-Bit Multiplexed Address/Data Mode	1-7
Figure 4. 16-Bit Multiplexed Address/Data Mode	1-7
Figure 5. Non-Multiplexed Mode 8-Bit Data Bus	1-7
Figure 6. Non-Multiplexed Mode 16-Bit Data Bus	1-8
Figure 7. PSD3XX Programmable Array Decoder.....	1-9
Figure 8. PSD3XX Port A Structure	1-11
Figure 9. PSD3XX Port B Structure.....	1-12

**Figures
(Cont.)**

Figure 10. Intel 80C31/PSD3XX Applications	1-18
Figure 11. Intel 80C31/2/PSD3XX Applications	1-20
Figure 12. M68HC11/PSD3XX Applications.....	1-22
Figure 13. M68008/PSD3XX Applications	1-24
Figure 14. M68000/PSD3XX Applications	1-26
Figure 15. M68000/2X PSD3XX Applications	1-28
Figure 16. Intel 80186/PSD3XX Applications.....	1-30
Figure 17. Intel 80196/PSD3XX Applications Open-Drain Drivers.....	1-32
Figure 18. Z80/PSD3XX Applications	1-34
Figure 19. 6809/PSD3XX Applications	1-35
Figure 20. Intel 80286/PSD3XX Applications.....	1-37
Figure 21. M68HC11/PSD3XX to M68230 Applications.....	1-39
Figure 22. M68HC11/PSD3XX to 16K SRAM Applications.....	1-41
Figure 23. SC80C451/PSD3XX to 16K SRAM Applications	1-43
Figure 24. Intel 80196/PSD3XX Track Mode to External SRAM.....	1-45
Figure 25. MAPLE Main Menu	1-47
Figure 26. MAPLE Menu with PARTNAME Submenu	1-48
Figure 27. CONFIGURATION Menu	1-49
Figure 28. Port C Configuration Menu	1-50
Figure 29. Port A Configuration Menu, Part 1	1-50
Figure 30. Port A Configuration Menu, Part 2	1-51
Figure 31. Port B Configuration Menu.....	1-51
Figure 32. Address MAP Menu	1-52
Figure 33. Port B Configuration Menu with Address Map	1-53

Tables

Table 1. Port Base Address Offset.....	1-10
Table 2. Non-Volatile Configuration Bits	1-14
Table 3. Small Controller System with One 80C31 and One PSD3XX.....	1-17
Table 4. 80C31 Interface to Two PSD3XX Devices with Power Economy Feature.....	1-19
Table 5. M68HC11 to PSD3XX Interface	1-21
Table 6. M68008 to PSD3XX Interface	1-23
Table 7. M68000 Microprocessor to One PSD3XX Interface	1-25
Table 8. M68000 Microprocessor to Two PSD3XX Devices in Parallel	1-27
Table 9. Intel 80196 to PSD3XX Configuration for CMOS Ports	1-29
Table 10. Intel 80196 to PSD3XX Configuration for LED Drivers	1-31
Table 11. Z80B to PSD3XX Interface	1-33
Table 12. M6809 to PSD3XX Interface	1-33
Table 13. Intel 80286 to PSD3XX Interface	1-36
Table 14. M68HC11/PSD3XX to External Peripheral M68230 Interface	1-38
Table 15. M68HC11/PSD3XX Configured to Address Additional SRAM	1-40
Table 16. SC80C451/PSD3XX Configured to Address Additional SRAM	1-42
Table 17. Intel 80196 to PSD3XX Used to Access External SRAM in Track Mode	1-44



Programmable Peripheral Application Note 011 PSD3XX Device Description

Chapter 1

Introduction

The PSD3XX family of products include flexible I/O ports, PLD, Page Register, 256K to 1M EPROM, 16K bit SRAM and "Glueless" Logic Interface to the micro controller. The PSD3XX is ideal for microcontroller based applications where fast time-to-market, small form factor and low power consumption are essential. These applications include disk controllers, cellular phones, modems, fax machines, medical instrumentation, industrial control, automotive engine control and many others.

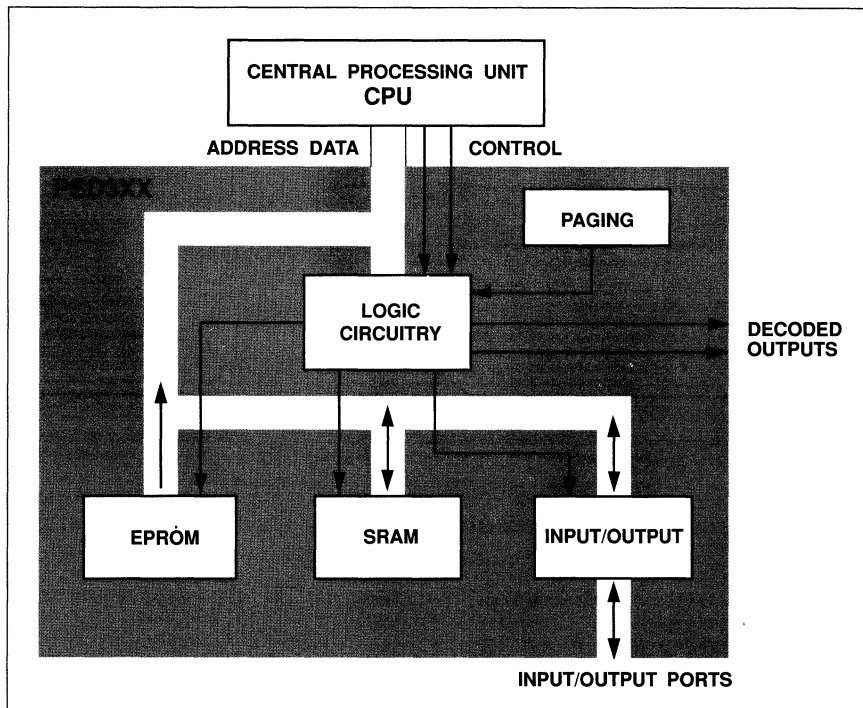
Traditionally, central processing units (CPUs) require the support of non-volatile memory for program storage, random access memory (RAM) for data storage, and some input/output (I/O) capability to communicate with external devices. The addition of general logic circuitry is necessary to 'glue' the parts of the system together. Figure 1 shows a

block diagram of such a system, configured with a CPU (or microprocessor). The typical microprocessor also has integrated into it on-board timers, a small amount of RAM and ROM, as well as a limited I/O capability.

The microprocessor (and often the microcontroller) requires additional external support EPROM and RAM memory, additional ports, memory mapping logic, and sometimes latches to separate address and data from a multiplexed address/data bus. Until very recently, designers had to create a discrete solution from a number of chips, or generate a full custom solution. Now, the PSD3XX integrates the different system support blocks into a single-chip solution. This relieves the designer from the constraint of thinking that memory mapping, ports, and address latch requirements should be developed from separate elements.

1

Figure 1.
PSD3XX supports
CPU as a Com-
plete peripheral,
memory, and
logic subsystem



**Introduction
(Cont.)**

This high integration of functionality into a single chip enables designers to reduce the overall chip count of the system. The result is increased system reliability, simpler PCB layout, and lower inventory and assembly costs. By integrating ports, latches, a Programmable Address Decoder (PAD), EPROM, and static RAM, the PSD3XX can bring the system solution down to only two chips: a microcontroller and a PSD3XX. The alternative solution would be discrete elements of RAM, EPROM, I/O mapped ports, and latches all mapped into the address scheme by a programmable logic device (PLD). This could escalate the chip count to 8-12 packages, depending on size and complexity.

For larger systems, multiple PSD3XX's can be configured. Due to its versatility and flexibility, two or more PSDs can be cascaded either horizontally (increasing bus width) or vertically (increasing sub-system depth). This proportionally increases the complement of memory, I/O ports, and chip-selects without the need for additional external glue logic.

An additional feature of the PSD3XX is its ability to support a wide range of microcontrollers or microprocessors because it has been designed with a wide range of configurable options. The designer can program any one of a number of different options to create specific compatibility with a host processor. Furthermore, this can be done without the need for external glue logic.

**WSI Software
Support for the
PSD Family**

The PSD family from WSI can be easily configured from a low-cost software support package called MAPLE. Designed to run in an IBM/PC environment, MAPLE makes design and configuration of the PSD3XX a

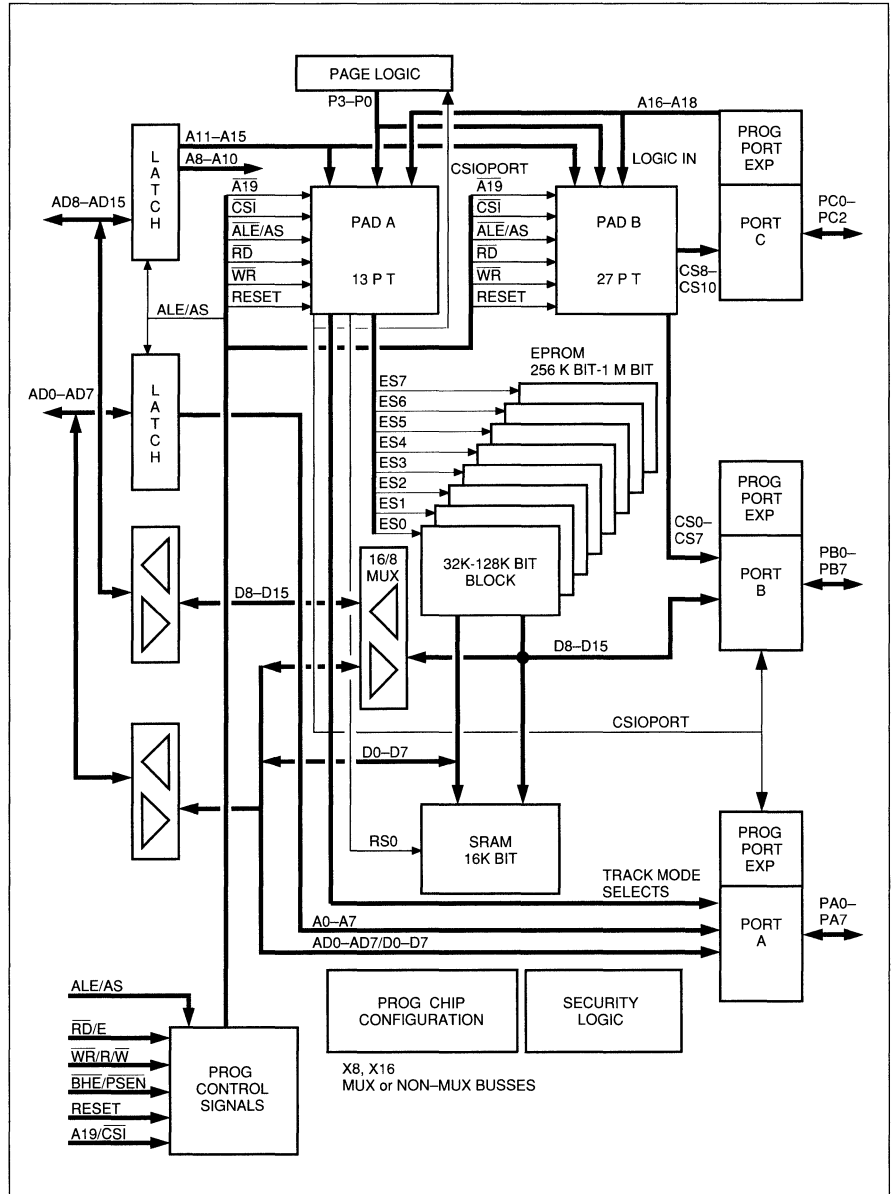
simple task. Memory mapping of EPROM and RAM blocks replaces PLD-like equations with user-friendly, high-level command entries.

**PSD3XX
Architecture and
Pin Nomenclature**

The PSD3XX is available in a variety of 44-pin packages (see the PSD3XX Data Sheet). Figure 2 is a functional block diagram of the

PSD3XX that shows the pin functions, internal architecture, and bus structure.

Figure 2.
PSD3XX
Architecture.



PSD3XX Architecture and Pin Nomenclature (Cont.)

Inputs AD0–AD15 enter the PSD through latches. These can be programmed to latch the address/data inputs, removing the need for such devices as the 74HCT373 or 573. Alternatively, in the transparent mode they simply buffer address inputs. The Address Latch Enable (ALE) signal is available to register a valid address input on the AD0–AD15 lines; its active polarity is programmable. Another name for this input is Address Strobe (AS). It provides the same function and the same timing as the ALE, but this pin name is more appropriate to Motorola-type systems. When either ALE or AS is valid, the latches are transparent; when inactive, the address/data inputs on AD0–AD15 remain latched.

The PSD3XX also contains a Programmable Address Decoder (PAD). Figure 2 shows that address inputs A11–A15 (and, possibly, inputs A16–A19) go directly to the PAD. Other inputs to the PAD include $\overline{RD}(E)$, $\overline{WR}(R/W)$, and ALE(AS). Programming of the PAD enables the designer to internally select the EPROM banks via internal chip-select lines ES0–ES7. An additional chip-select for the internal SRAM is available through RS0. Port C conveys either $\overline{CS8}$ – $\overline{CS10}$ to external

devices or receives A16–A18 inputs, directing them to the PAD. Also, A19 can be programmed to go directly into the PAD. Note that these lines are not necessarily dedicated to address inputs; they can be used as general purpose logic inputs. Thus, the PAD can be programmed to perform general combinational logic without adding any 'glue logic' to the overall system design. Address inputs A16–A19 can be used as general inputs to the PAD for implementing logic equations, and not for address decoding. If they are not used, A16–A19 are "don't care" conditions in memory map allocation. (See Figure 7 for a more detailed diagram of the PAD.)

The internal port options (Ports A and B) are both 8 bit-wide and can be programmed to act as traditional I/O ports. Port C is a 3-bit port designed to output logic functions from the PAD, receive address inputs A16–A18, or a combination of both. Ports A and B, however, are more complex because a number of different options can be selected with regard to system configuration. Figures 3, 4, 5, and 6 show the variety of configurations that are available to these ports.

Performance Characteristics

Two key timing parameters associated with the device are the EPROM/SRAM access times and the propagation delay through the PAD. The worst-case delay from valid address input to valid data output is 120 ns whether the address input is multiplexed or not. The cycle time of the system is virtually 120 ns with a small margin for address switching. This gives a system clock rate of

about 8.3 MHz. Considering the power-down option, it takes 100 ns for active power input enabled through the $\overline{CS1}$ to valid data output. If the chip-select output option is chosen for either Port B or Port C, the propagation delay for address and control input through the PAD to valid chip-select output is 35 ns.

PSD3XX System Configuration for Port and I/O Options

In this section, the EPROM and SRAM are treated as separate entities and the four options available for configuring the PSD301 in a processor system are detailed. Figure 3 shows an 8-bit data configuration for systems that multiplex 8-bits of data (D0–D7) with the corresponding address inputs (A0–A7). Lines A8–A15 are dedicated to higher-order address inputs. Ports A and B are then available for data I/O and Port C is available for additional inputs, A16–A18 or chip-select outputs $\overline{CS8}$ – $\overline{CS10}$. Port A also has the option of passing any one or all of the internally latched lower-order addresses (A0–A7) to the output. Another mode supported by

Port A is called "track mode." In this mode, the PSD301 can be programmed to pass the I/Os AD0–AD7 through the device enabling a shared memory or peripheral resource to be accessed. Port B has an additional mode to the general port mode. The PSD301's on-chip PAD can be programmed to generate chip-select signals which can be routed to Port B's output for external chip selection as $\overline{CS0}$ – $\overline{CS7}$. Port C can be programmed for inputs A16–A18 or as additional chip-select outputs $\overline{CS8}$ – $\overline{CS10}$. Although labeled as address inputs, A16–A18 can be used for general Boolean inputs to the PAD array.

Figure 3.
8-bit multiplexed address/data mode

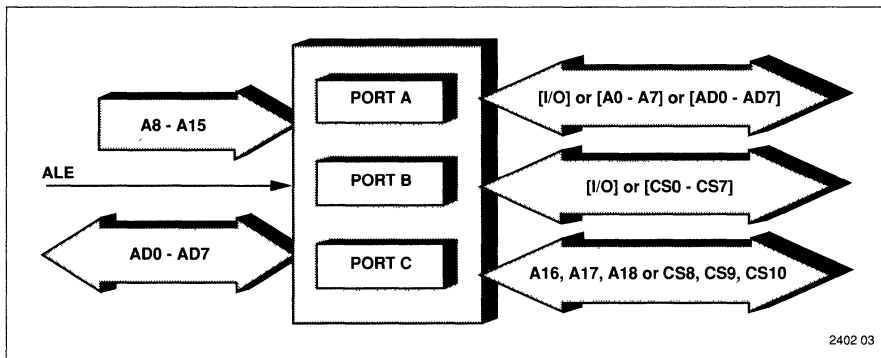
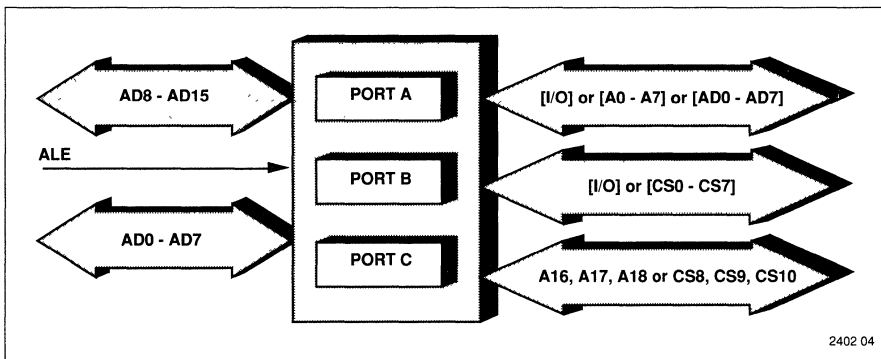


Figure 4 extends the option offered in Figure 3 to a 16-bit multiplexed bus. AD8-AD15 convey address and data I/O. The port options remain the same as for Figure 3; thus,

these two configurations are suitable for multiplexed address/data systems of 8 or 16 bits.

Figure 4.
16-bit multiplexed address/data mode.



Figures 5 and 6 show options for a non-multiplexed host processor or controller. Figure 5 is suited to byte-wide systems and Figure 6 to 16-bit word-wide configurations. In Figure 5, Port A is used for data D0-D7 but

Port B is still available for general I/O operations or chip-select outputs. This configuration is suitable for processors such as the M68008.

Figure 5.
Non-multiplexed Mode 8-bit Data Bus

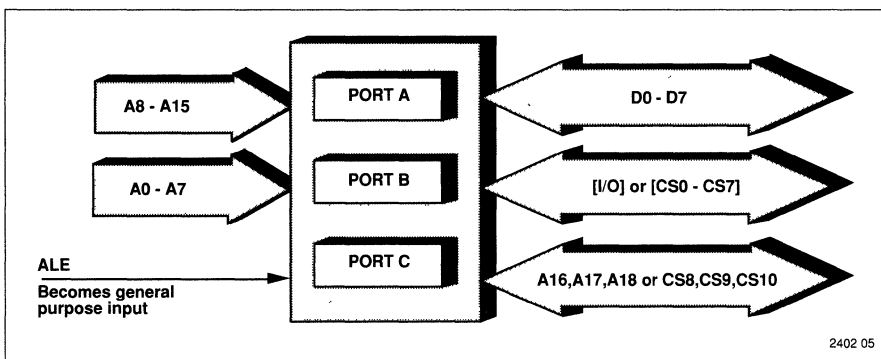
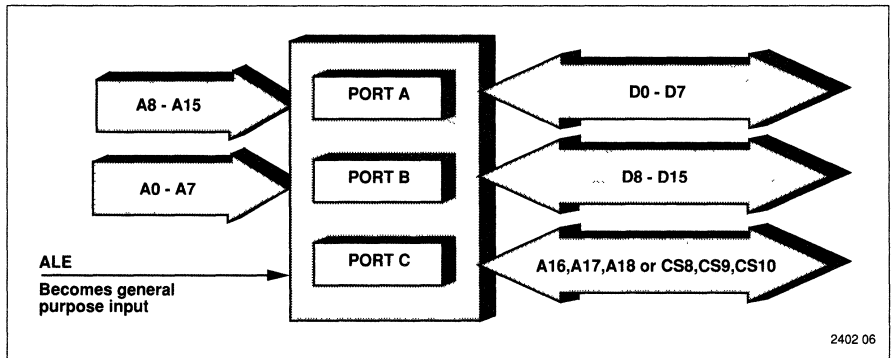


Figure 6.
Non-multiplexed
Mode 16-bit Data
Bus



The function of Port C is the same in all of the four modes of operation. For 16-bit data transfers, an additional 8 bits of data is required. Figure 6 shows Port B as the data bus for the higher-order data byte D8–D15.

With D0–D7, this configuration is suitable for 16-bit microprocessors such as the M68000. Port C is available for address inputs or chip-select outputs.

Address Inputs

The processor interface has 16 address inputs: A0–A15. The device can be programmed to accept either address inputs or multiplexed address/data inputs. The address lines can be latched into the one or two octal latches for multiplexed byte or word-wide buses respectively. The device is initially programmed with a word configuration setting the PSD3XX to a specific mode; for example, one configuration bit selects whether the address input is multiplexed with data or is a non-multiplexed dedicated address. In the non-multiplexed scheme, the input latches are held as transparent. When the address inputs are valid on the chip as A0–A15, they can be subdivided into two buses: as lower-order addresses (A1–A11), and as higher-order addresses (A12–A15). A1–A11 go directly to the EPROM and inputs A1–A10 go to the SRAM (see Figure 2). The EPROM blocks are selected through the PAD via outputs ES0–ES7 as shown in Figure 2; and the SRAM is selected by the RS0 output.

The address input lines A11–A15, along with possible additional address inputs A16–A19, go into the PAD array. These address inputs are available for mapping the blocks of memory into the map scheme of the system. One option is to program the additional address inputs as valid higher-order address inputs for memory addressing ranges above 64K bytes or 32K words. If A16–A18 are not required, these PAD inputs can be ignored. Only microprocessors and microcontrollers with a large addressing range use these higher-order address lines. A second option is to disregard these address inputs to the

chip in favor of additional chip-select outputs. A third option is available if the designer does not need additional chip-select outputs or high-order address inputs. The inputs A16–A18 can be used as general-purpose logic inputs. Examples of this are illustrated in some of the following applications.

An interface with the Z80B microprocessor uses inputs A16, A17, and A18 for signals $\overline{M1}$, \overline{MREQ} , and \overline{IORQ} , respectively. In the M68008 application, two of these pins are programmed as \overline{DTACK} and \overline{BERR} from the PSD301 to the M68008. A wired-OR function can be implemented on the \overline{DTACK} or \overline{BERR} input if the user takes advantage of Port B's open-drain feature. If two PSD3XX devices are used together, the \overline{DTACK} and \overline{BERR} lines can be wired together and the external pull-up resistors can be used to tie these lines HIGH. It is also possible to use the internal PAD of one PSD3XX to gate these lines together and produce composite \overline{DTACK} and \overline{BERR} inputs to the M68008.

Internally, the memory blocks are arranged word-wide with a byte-wide isolation buffer separating the lower and upper bytes. This buffer is controlled from the configuration section of the PSD3XX. When the PSD3XX is configured to operate in word-wide mode, this buffer isolates the two buses into D0–D7 and D8–D15. In word-wide mode, the control of the data flow through this buffer is determined by BHE, A0, and the device's current configuration mode. Accessing byte-wide data can be thought of as accessing bytes on even and odd word boundaries or as two separate

**Address Inputs
(Cont.)**

banks of byte-wide data. The total complement of EPROM is shown as eight banks. The chip-select outputs ES0-ES7 come from the PAD. These are program-

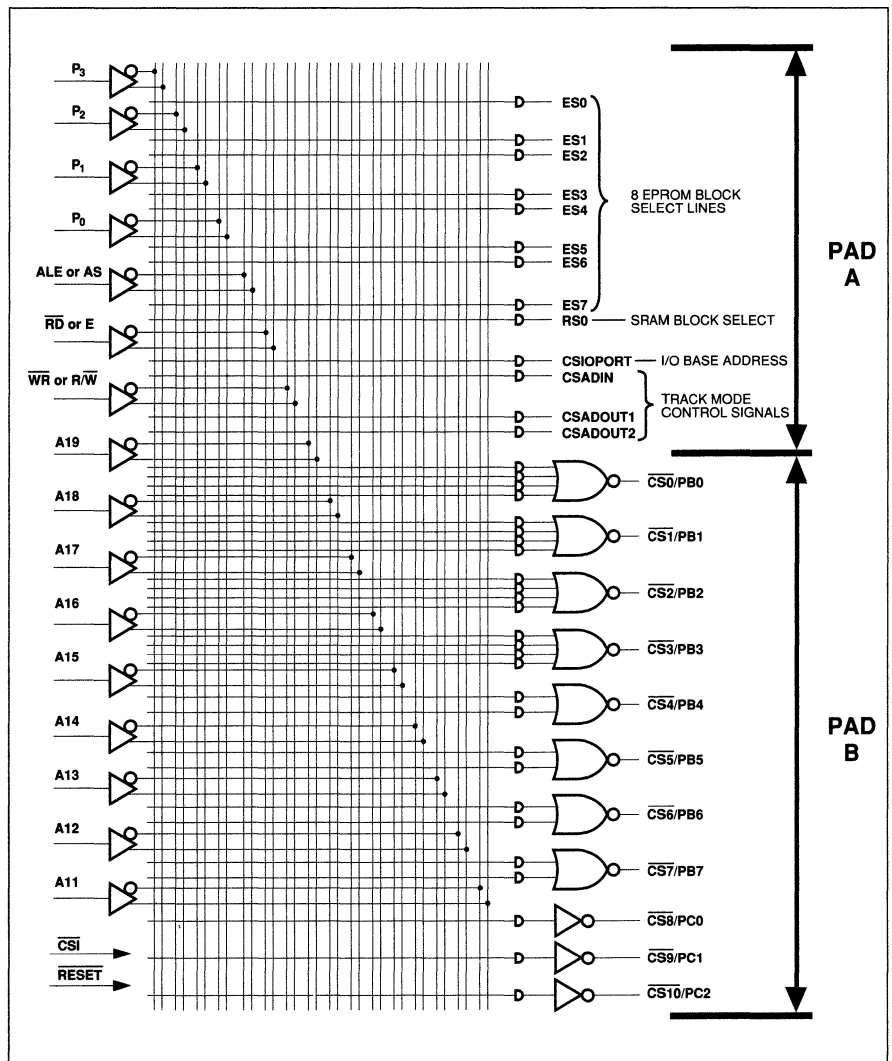
mable address and control decode signals from the PAD inputs. Figure 7 provides a detailed schematic diagram of the PAD in terms of a traditional PLD.

**PSD3XX
Programmable
Array Decoder
(PAD)**

The PAD is an EPROM-based reprogrammable logic fuse array with sum-of-product outputs. For Intel-type configurations, inputs to the PAD are A11-A19, ALE, RD, and WR. For Motorola type configurations, they are R/W, AS, and E. The CSi and RESET inputs are used to deselect the PAD for power-down configurations and initialization, respectively. Internal to the PSD301 are the ES0-ES7

EPROM select lines. There is one product term dedicated to each EPROM block, and a single product term (RSO) for the SRAM selection. Address and control for each EPROM bank can be programmed to a resolution of a 4K word boundary and positioned anywhere in the mapping scheme of the designer's system. Similarly, the SRAM can be positioned on 2K word boundaries.

**Figure 7.
PSD3XX
Programmable
Array Decoder**



PSD3XX Programmable Array Decoder (PAD) (Cont.)

Other internal product term outputs from the PAD are the CSIOPORT, CSADIN, CSADOUT1, and CSADOUT2 lines. A single product term generates the CSIOPORT signal; this provides a base address for Ports A and B. The registers relevant to these ports are addressed as a base offset (see Table 1). The CSADIN signal is used to control the input buffer in the track mode. It can be enabled to read data in a programmed address space from Port A through the

PSD3XX. CSADOUT1–2 are used to control the multiplexed address and write data through the PSD3XX to the Port A pads. The address range is programmed into the PAD qualifying the address space, but CSADOUT1 is qualified by the ALE signal outside of the PAD. This automatically lets the design distinguish between address and write data. To qualify valid write data, the PSD3XX automatically includes the CSADOUT2 product term with the \overline{WR} or R/\overline{W} signal.

Table 1.
Port Base Address
Offset

Register Name	Offset From The CSIOPORT Base Address
Pin Register of Port A	+2 (Accessible only during Read)
Pin Register of Port B	+3 (Accessible only during Read)
Direction Register Port A	+4
Direction Register Port B	+5
Data Register Port A	+6
Data Register Port B	+7
Pin Register of Ports A and B	+2 (Accessible only during Read)
Direction Register of Ports A and B	+4
Data Register of Ports A and B	+6

The PAD structure enables additional chip-selects to be routed to the Port B output pins. The four chip-select outputs ($\overline{CS0}$ – $\overline{CS3}$) are supported by four product terms per output. $\overline{CS4}$ – $\overline{CS7}$ have two product terms per output. The ability to use more than one product term from a chip-select enables the mapping of additional devices to be distributed through the address space, rather than selecting memory as a block. Sacrificing Port B terminals for chip-selects could occur in systems requiring a larger EPROM, RAM, or

I/O space. Additional PSD3XX devices can be designed into a system by using the chip-select outputs from Port C or B of one master PSD3XX. This is required for addressing a space greater than 1M. Finally, the outputs of the sum-of-product terms are inverted to be consistent with active LOW chip-select inputs for additional external RAM, EPROM, peripherals, or busses. Port C has the capability of providing three additional external chip-selects, each supporting one product term per output.

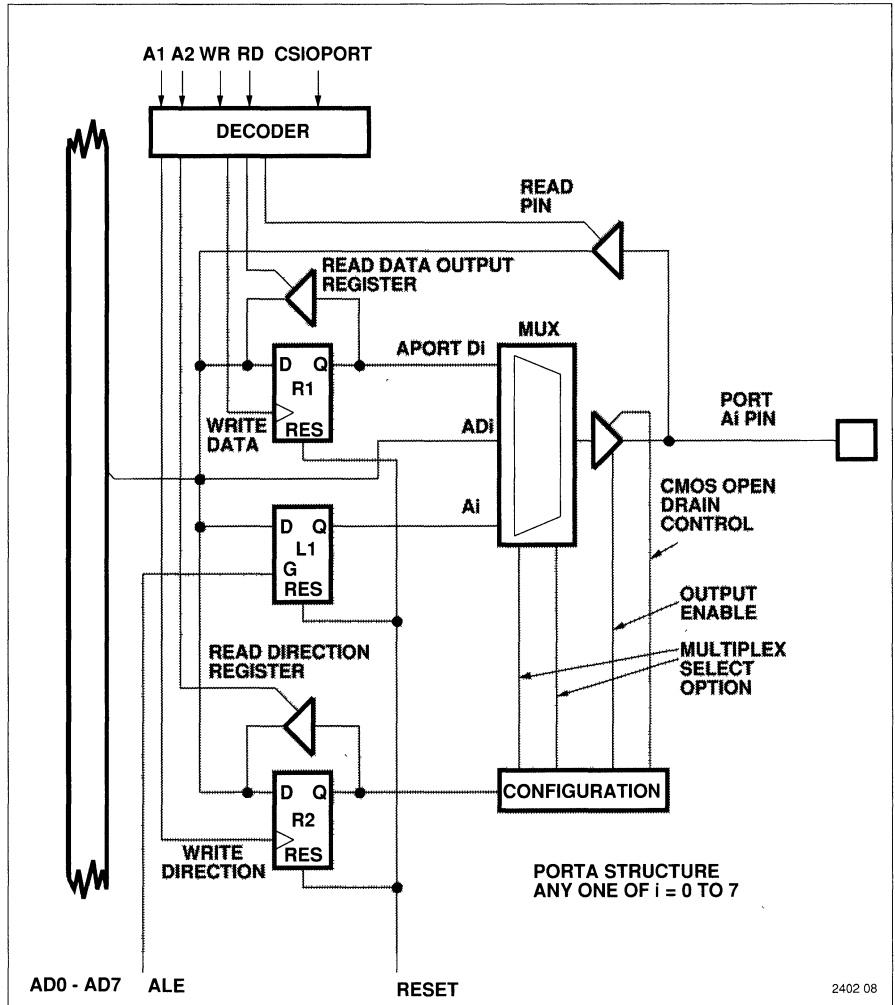
Microcontroller/ Microprocessor Control Inputs

The control inputs are also programmable: \overline{WR} or R/\overline{W} and \overline{RD} or E are used for read/write control of the internal EPROM, RAM, and I/O capability. Other control inputs are a programmable option for Bus High Enable or Program Store Enable (BHE/ \overline{PSEN}) and Address Latch Enable or Address Strobe (ALE/AS). These pins are selected to suit the bus protocol of the host processor or, where not applicable, they can be ignored. The $\overline{CS1}$ /A19 input is available either for a power-down chip-select enable or as a higher-order address input without the power-down feature. The final control input is the RESET input; this also is a programmable option. Its active polarity can be chosen to be compat-

ible with the host system. The function of the RESET input is to clear and initialize the PSD301 at start-up. All I/Os are set up as inputs and all outputs are either in a non-active or three-state condition.

Consequently, the PSD3XX is prevented from actively driving outputs during start-up. This feature was incorporated to prevent potential bus conflicts. In Figure 2, the $\overline{CS1}$ and RESET inputs are shown also as PAD inputs. $\overline{CS1}$ is a hardware option into the PAD that powers down the internal circuitry and is used in power-sensitive applications. Neither signal is available as a programmable option.

Figure 8.
PSD3XX Port A
Structure

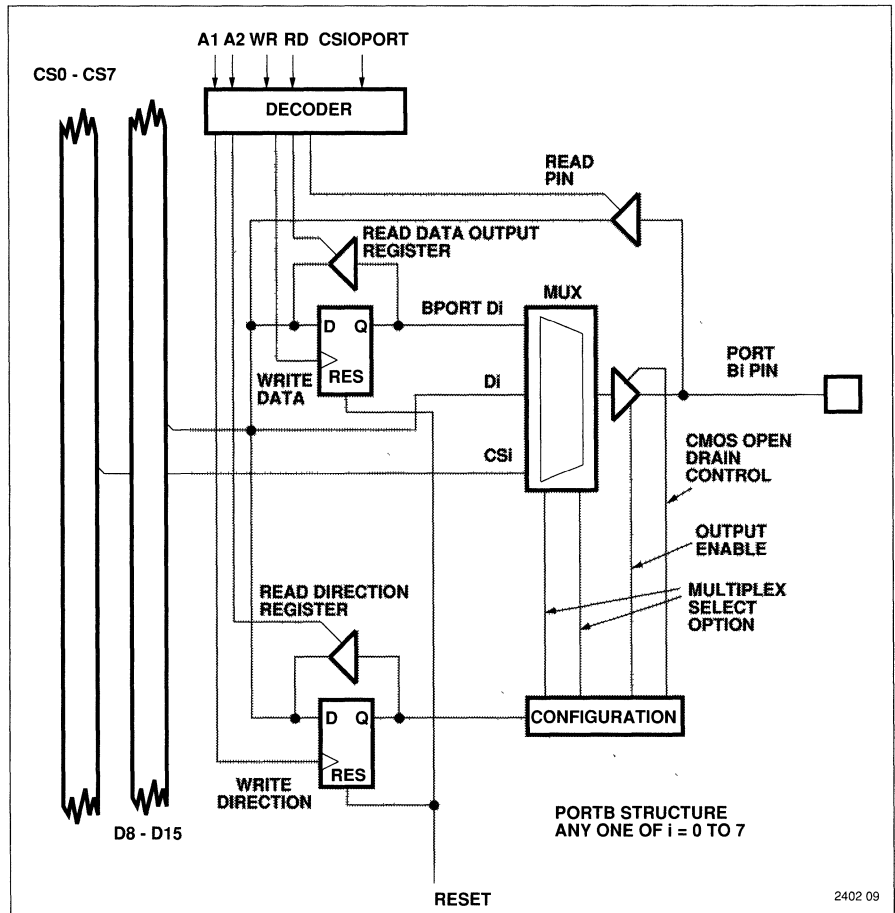


Input and Output Ports

The port section comprises Port A (8 bits), Port B (8 bits), and Port C (3 bits). These support the many different I/O operations. For port expansion, Ports A and B can be configured as general I/O ports, each to

convey eight bits of digital data to and from an external device. Figure 8 shows a single cell of Port A, Figure 9 shows a single cell of Port B.

Figure 9.
PSD3XX Port B
Structure



Writing data to a port is similar to writing data to a RAM location. If a port is programmed as an output, data is loaded into the output register as if it were a RAM location. Although the ports are not bit addressable, individual bits can be selected as either input or output. Thus, PA0-5 can be set as data outputs while PA6 and PA7 can be configured as inputs. Any mix of I/Os is possible giving the ports additional flexibility.

The direction of data flow through the port is

determined by the data direction register. This register is dynamically programmable so that the I/O direction through Ports A and B can be altered during the microcontroller program execution. The data direction register initializes with logic zeros after an active RESET and causes each port bit to be set as an input. This state of initialization guarantees that the ports are prevented from driving the output lines at start-up. If the user requires all the Port A or Port B bits to be

Input and Output Ports (Cont.)

inputs, the data direction register can be left in this default state. To enable it as an output, logic ones can be written into the data direction location.

Due to the internal design, it is possible to program Port A or Port B bit lines as inputs and still write data to the port locations. This is because both ports have on-chip latches and can hold data. These registers are hidden or buried; i.e., they exist in the port and their condition can be read back at any time. However, these outputs do not drive the output pins because the port has been enabled as an input.

To access the port as a memory mapped location, the initial selection is made through the PAD's CSIOPORT. This provides a base address from which the locations shown in Table 1 give access to the various ports or their options. The configuration support software automatically ensures that there is no conflict between an SRAM location and I/O port in the case of memory mapped peripherals. It is also possible for the PSD3XX to distinguish between I/O and memory mapped locations. The user can input memory and

I/O control signals to the PAD through the A16–A18 inputs and program an active CSIOPORT output by decoding these signals. This can be achieved with Intel- and Zilog-type processors which have separate memory and I/O controls. Signal input through pins A16–A18 is made possible through Port C. This 3-bit port is responsible for either PAD chip-select outputs or address/logic inputs. CSIOPORT points to a base address at which Ports A and B reside. Table 1 provides the offset from the base address and the associated port function. Figure 2 shows that Port A is driven by a multiplexed address/data bus of AD0–AD7 and the selection of address/data is made from the configuration memory and internal control functions.

The other options available to the user are selecting 1) the shared resource or track mode where AD0–AD7 is routed directly through to the Port A output, or 2) the latched address A0–A7. In track mode, AD0–AD7 inputs to the PSD3XX are used to access local or private memory and peripherals and the outputs AD0–AD7 through Port A are used to access a public resource.

PSD3XX General System Configuration

The PSD3XX family devices consists of two byte-wide configurable I/O ports (Ports A and B), 256K to 1M bits of EPROM, 16K bits of RAM, and the PAD. Additional I/O capability to and from the PAD is through a 3-bit I/O (Port C). There are also on-chip latches to support processors and controllers that multiplex address and data on the same bus. The EPROM memory section of the device is programmable just like a standard EPROM device. However, unlike the single-chip EPROM, the PSD3XX must also be configured to function into one of its many possible modes of operation: This is done by programming a non-volatile EPROM memory location with 45 configuration bits. These bits select the mode of operation and are programmed into the EPROM along with the hexadecimal microprocessor/microcontroller assembly language object code. When using MAPLE software,

the assignment of logic conditions to the configuration bits locations is transparent to the user; the resultant word is merged with the EPROM code and the data map for the PAD.

Table 2 shows the the configuration locations and their functional assignment. For example, one of the configuration bits enables the device architecture to be compatible for either byte- or word-wide data buses. This is the configuration data or CDATA bit. The 256 Kbits of EPROM can be configured as a 32K byte-wide bus for applications with an 8031 microcontroller or as a 16K word-wide bus for applications with an M68000 microprocessor. These configuration bits are discussed in detail as each feature is covered in this application note.

Table 2.
Non-volatile
Configuration
Bits

Configuration Bits	Number of Bits	Function
CDATA	1	CDATA. 0 = eight bits, 1 = sixteen bits
CADDRAT	1	ADDRESS/DATA Multiplexed. 0 = Non-multiplexed, 1 = Multiplexed
CRRWR	1	CRRWR. 0 = \overline{RD} and \overline{WR} , 1 = R/\overline{W} and E
CA19/ \overline{CS} I	1	A19 or \overline{CS} I. 0 = Enable power-down, 1 = Enable A19
CALE	1	ALE Polarity. 0 = Active HIGH, 1 = Active LOW
CRESET	1	CRESET. 0 = Active LOW RESET, 1 = Active HIGH RESET
\overline{COMB}/SEP	1	Combined or Separate Address Space for SRAM and EPROM. 0 = Combined, 1 = Separate
CPAF2	1	Port A Track Mode or Port Mode. 0 = Port or Address, 1 = AD0-AD7 Track Mode
CPAF1	8	Port A I/O or A0-A7. 0 = Port A pin is I/O, 1 = Port A pin is Address
CPBF	8	Port B I/O or CS. 0 = Port B pins are CS _i (i = 0-7), 1 = Port B pins are I/O
CPCF	3	Port C A16-A18 or \overline{CS} 8- \overline{CS} 10. 0 = Port C pins are Address, 1 = Port C pins are Chip-select
CPACOD	8	Port A CMOS or Open Drain. 0 = CMOS drivers, 1 = Open Drain
CPBCOD	8	Port B CMOS or Open Drain. 0 = CMOS Drivers, 1 = Open Drain
CADDHLT	1	A16-A18 Transparent or Latched. 1 = Address latched, 0 = Address transparent
CSECURITY	1	CSECURITY On/Off. 0 = Off, 1 = On

In addition to bus width, the polarity and mode of the bus control signals are programmable. There are two types of read/write control: one is consistent with either a Motorola and Texas Instruments control bus standard; the other is consistent with the Intel/National Semiconductor/Zilog control bus standard. The configure read and write bit (CRRWR), distinguishes between one of two conventions: either an Intel (8031) or Motorola (M68HC11) convention can be selected by programming this single bit in the configuration memory. The Intel device requires the PSD3XX to be programmed with an active LOW \overline{RD} and \overline{WR} controls (CRRWR = 0). For applications with the Motorola microprocessor, select the R/ \overline{W} and E option (CRRWR = 1). In addition to a choice of two READ/WRITE controls, the user can select either a multiplexed Address/Data Bus or separate address and data lines.

Figure 3 shows the configuration that is best suited for the 8031 microcontroller; Figure 4 shows the configuration for an 80196 microcontroller with a 16-bit multiplexed addressed/ data bus. For the non-multiplexed modes:

Figure 5 applies to M6809 microprocessors, while Figure 6 shows the mode applicable to the M68000. Selection of multiplexed or non-multiplexed buses is a programmable option that can be invoked through the configure address/data multiplex (CADDRAT) bit. With the 8031 controller, address outputs A0-A7 are multiplexed with the data D0-D7 input/output lines to create a composite AD0-AD7 bus.

The PSD3XX's input latches can be programmed to catch a valid address when the microcontroller's ALE signal transitions from active HIGH to inactive LOW. The polarity of the ALE signal is also a programmable feature in the CALE field of the configuration table. Address latching can be programmed to occur on either an active HIGH or an active LOW ALE signal. With Intel devices, the address is valid when ALE is HIGH. Once latched, data or code can be read from, or written to, the PSD3XX. The CALE active HIGH or LOW ALE configuration bit only applies to addresses A0-A15. A separate configuration bit, (CADDHLT), exists for the control of the higher-order address inputs

PSD3XX General System Configuration (Cont.)

(A16–A19). If necessary, these addresses can also be latched by the host system.

The highest address input is A19 but this signal can be omitted in favor of a power-down chip-select input (\overline{CS} I). A19/ \overline{CS} I is selected by the CA19/ \overline{CS} I configuration bit. When the \overline{CS} I input is selected and the pin is driven HIGH, the device can be powered-down consuming only standby power. When configured with other CMOS devices, the standby power is in the 80–250 μ A range. Many CMOS microcontrollers do not need a large memory address space; thus, address inputs A16–A19 would be unnecessary. The CA19/ \overline{CS} I input can be programmed with a logic LOW to enable a power-down option for power sensitive applications.

The address/data multiplexed scheme also supports the 16-bit processors. In this case, AD0–AD15 convey a 16-bit address qualified by ALE (or AS for the Motorola convention) and 16-bits of data I/O. This feature is shown in Figure 4. A microcontroller that would use this scheme is the 80C196. The M68HC11, like the 8031, uses the 8-bit multiplexed scheme but with the Motorola convention for bus control.

Another control pin used for 80C31 applications used to distinguish between program

and data memory is the \overline{PSEN} output. The $\overline{COMB/SEP}$ configuration bit should be programmed HIGH if data and memory are separate and LOW to configure a combined memory space in the PSD3XX. This is a useful feature for systems that require program memory and data memory to be in separate blocks.

For systems that use separate data and address buses, the address latches can be set into a transparent mode by clearing the CADDRDAT bit location. Thus, the PSD3XX is suitable for multiplexed or non-multiplexed bus structures employing 8- or 16-bit bus widths.

The RESET input to the PSD3XX enables the device to be initialized at start-up. RESET can be either active HIGH or active LOW depending on the processor type. The CRESET configuration bit selects the polarity of the RESET input: LOW for active LOW and HIGH for active HIGH RESET. Normally, memory systems do not require a RESET input; however, the PSD3XX contains data direction registers for the ports that must be initialized at start-up. Note that all port I/O buffers are automatically programmed as inputs during start-up.

PSD3XX Configuration for Port Reconstruction

A key feature of the PSD3XX is the concept of port reconstruction. When using microcontrollers with additional off-chip memory, port I/O address lines are sacrificed for address, data, and memory control lines. With a multiplexed address/data scheme, two 8-bit controller ports could be lost to address and data. Furthermore, in some control applications, many port I/O bits are required to send actuating signals to solenoids, instrument displays, etc., and receive data through sensors and switch panels. In many control environments, a large amount of I/O capability is required; also, additional external memory is needed for microcontroller instructions to perform data manipulation. Without the PSD3XX, the supplement of extra ports as discrete latches addressed through logic decoders can add a number of chips to the final design. By using the PSD3XX, additional EPROM, RAM, and ports are all provided on one chip. Port reconstruction lets the designer reclaim the two ports sacrificed for the microcontroller's address and data.

Port configuration is achieved through the configuration register bits. CPAF1 configura-

tion of Port A contains eight bits; programming a logic LOW assigns the selected bit with I/O capability as if it were a conventional port. If programmed HIGH, the internally latched address inputs A0–A7 are routed to Port A lines PA0–PA7. This feature enables other on-card peripherals to use A0–A7 as latched addresses. Without this feature, external peripherals to the PSD3XX would require an external octal latch to catch the multiplexed address when it becomes valid at the microcontroller's output. Configuration of Port A as general I/O or address/data is on a bit-wise basis; thus, the choice of port or address/data assignment can be mixed. For example, configuration code 11100000B programmed into location CPAF1 passes addresses A0–A2 to outputs PA0–PA2 and enables PA3–PA7 as conventional port lines.

Configuration bit CPAF2 is a 1-bit location. When programmed LOW, it selects the port/address option, as described above. If CPAF2 is programmed HIGH, port bits PA0–PA7 are set into track mode. Activity on the PA0–PA7 outputs follow logic transitions on inputs AD0–AD7. The multiplexed ad-

PSD3XX Configuration for Port Reconstruction (Cont.)

dress/ data input is tracked through PA0–PA7. Track mode enables the host microcontroller to access a shared memory and peripheral resource through the PSD3XX while maintaining the ability to access its own (private) memory/peripheral resource directly from the microcontroller's address/data outputs. In this mode, the address/data AD0–AD7 passes through the PSD3XX logically unaltered. In summary, PA0–PA7 can be programmed as port I/O or latched address outputs A0–A7 (each bit being programmed on an individual basis), or as AD0–AD7 outputs (track mode).

Port B bits PB0–PB7 can be programmed either as regular port I/Os, or as chip-select outputs $\overline{CS}0$ – $\overline{CS}7$ encoded from the PAD outputs. Figure 7 shows the PAD structure as a conventional PLD. Eight bits are programmed into CPBF. Logic LOW indicates that a port pin is a chip-select output derived from the PAD. Programming a logic HIGH sets the appropriate pin as an I/O function. The bit pattern 11111000B programmed into the CPBF location sets up PB0–PB4 as I/O ports and PB5–PB7 as chip-selects. The typical applications, where Port B is programmed as bi-directional, would be with microcontroller chips that need additional port bits. This would be in applications where port reconstruction is needed to drive many indicators,

solenoids, read switches, sensors, etc. In large microprocessor-based systems, the chip-select option would probably be chosen; in this case, the PAD outputs select other PSD devices, DRAM memory chips, and peripherals such as timers, UARTs, etc.

The three bits comprising Port C can be programmed by the CPCF configuration bits. This group of three bits define whether Port C is used for inputs (typically A16–A18) or whether the pins are used as chip-select outputs from the PAD. Although labeled as A16–A18, the nomenclature of these pins does not constrain the designer to using these inputs as dedicated higher-order address inputs. In fact, they can be general-purpose inputs to the PAD for processors that do not have an address capability above 64K locations. When the PSD3XX is used with the Z80B microprocessor, the Port C inputs have been programmed as \overline{MREQ} , \overline{IORQ} , and $\overline{M1}$. In the case of an interface to the M6809B, two inputs of Port C have been converted to chip-select outputs for other memory devices and one output has been used to feedback a READY input to the M6809B. Port C can be used as a general I/O from the PAD in the form of address, control, and chip-select bits. A logic LOW programs a port bit as an input; a HIGH programs it as an output.



Programmable Peripheral Application Note 011 Applications

Chapter 2

8-Bit Microcontroller to PSD3XX Interface

Figure 10 illustrates the minimum configuration of one controller and one PSD3XX. The application illustrates port reconstruction through the device's Port A and Port B I/O, reconstituting port 2 and port 0 of the microcontroller. Table 3 gives the configuration information that would be programmed in the configuration section of the PSD. Table 3 shows that both port I/Os have been programmed with CMOS load and drive characteristics. A feature of the 8051/8031 family is the PSEN signal, which determines whether the memory selection is active for executable

code or data. This family of controllers has separate memory locations for code and data. To maintain full compatibility, the PSD3XX is also capable of being programmed to respond to the PSEN signal. When A16–A18 are programmed as inputs but not driven, they should be tied active HIGH or LOW. Unused inputs to the PSD3XX must not be permitted to float. Tying can be avoided on unused A16–A18 lines if these are programmed as 'dummy' $\overline{CS}8$ – $\overline{CS}10$ outputs. A19/ $\overline{CS}1$ cannot be programmed as an output; thus, it must be tied if not used.

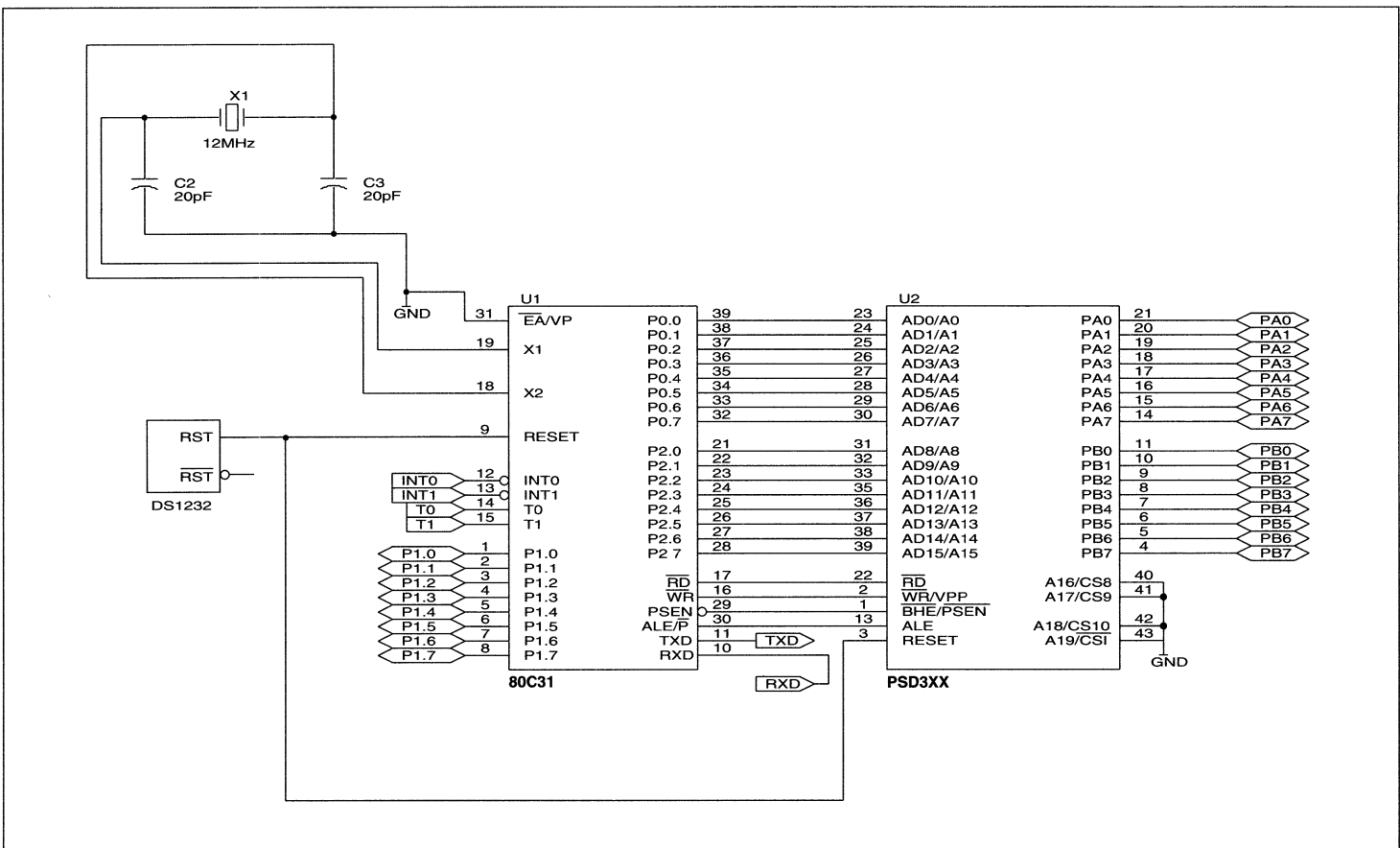
1

Table 3.
Small Controller
System with One
80C31 and One
PSD3XX

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	0	Set \overline{RD} and \overline{WR} mode
CA19/ $\overline{CS}1$	0	Set $\overline{CS}1$ input power-down mode
CALE	0	Active HIGH ALE
CRESET	1	Active HIGH RESET
\overline{COMB}/SEP	1	Code and data memory separate
CPAF2	0	Input/Output Port A
CPAF1	00H	Input/Output Port A (0–7)
CPBF	FFH	Input/Output Port B
CPCF	000B	Port C programmed for inputs
CPACOD	00H	Configure CMOS outputs Port A
CPBCOD	00H	Configure CMOS outputs Port B
CADDHLT	0	Transparent inputs A16–A19
CSECURITY	0	No security



Figure 10.
80C31/PSD3XX
Applications



Two PSD3XX Byte-Wide Interfaces to the Intel 80C31

Figure 11 illustrates an extension to the previous design in that two PSD3XX devices have been used, doubling the memory and port resources of the system solution. In this application, the power-down capability has been used so that one PSD3XX can be active while the other device is in power-down mode. The mean power consumption is reduced, so this configuration can be considered for power-sensitive applications.

The configuration Table 4 indicates that Port C has been configured as outputs. Provided one PSD3XX is powered up for the whole address range, its PAD can decode an address range to select and deselect the second PSD3XX device through the $\overline{CS10}$ output. In Figure 11, the PAD output A18/ $\overline{CS10}$ on PSD3XX U2 can be used to power-down the second PSD3XX through the A19/ $\overline{CS1}$ input.

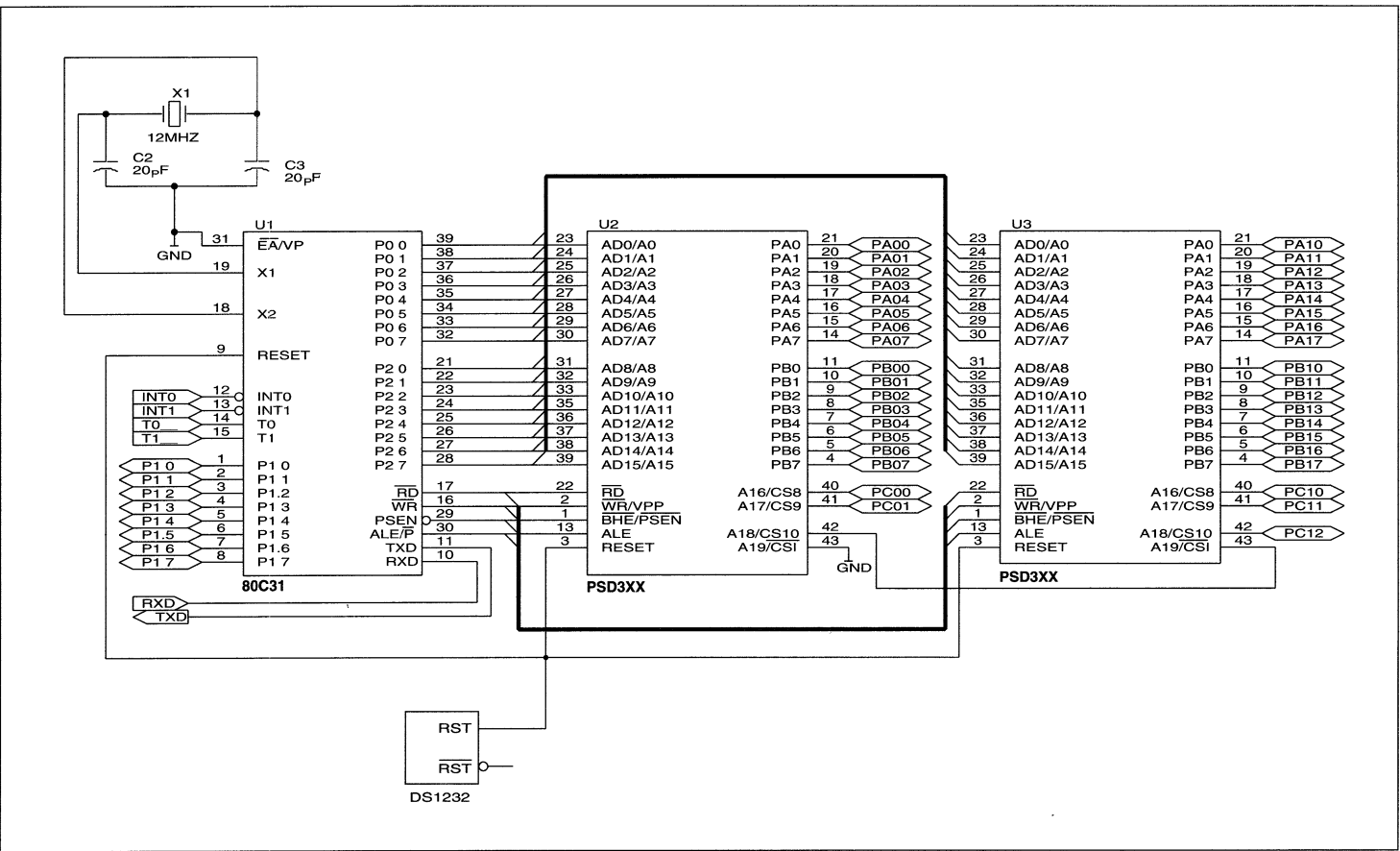
Table 4.
80C31 Interface
to Two PSD3XX
Devices with
Power Economy
Feature

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	0	Set \overline{RD} and \overline{WR} mode
CA19/ $\overline{CS1}$	0	Set $\overline{CS1}$ input power-down mode
CALE	0	Active HIGH ALE
CRESET	1	Active HIGH RESET
$\overline{COMB/SEP}$	1	Code and data memory separate
CPAF2	0	Input/Output Port A
CPAF1	00H	Input/Output Port A (0-7)
CPBF	FFH	Input/Output Port B
CPCF	111B	Outputs $\overline{CS8}$ - $\overline{CS10}$
CPACOD	00H	Configure CMOS outputs Port A
CPBCOD	00H	Configure CMOS outputs Port B
CADDHLT	X	"Don't care" for latched A16-A19
CSECURITY	0	No security

It is not recommended that the two PSD3XX devices select each other because the PAD section of a PSD device is powered down with the rest of the device. At least one PAD

decoder must be kept active to select and deselect others. Port C outputs $\overline{CS16}$ - $\overline{CS18}$ can power-down as many as three other PSD3XX devices.

Figure 11.
80C31/2/PSD3XX
Applications



PSD3XX M68HC11 Byte-Wide Interface

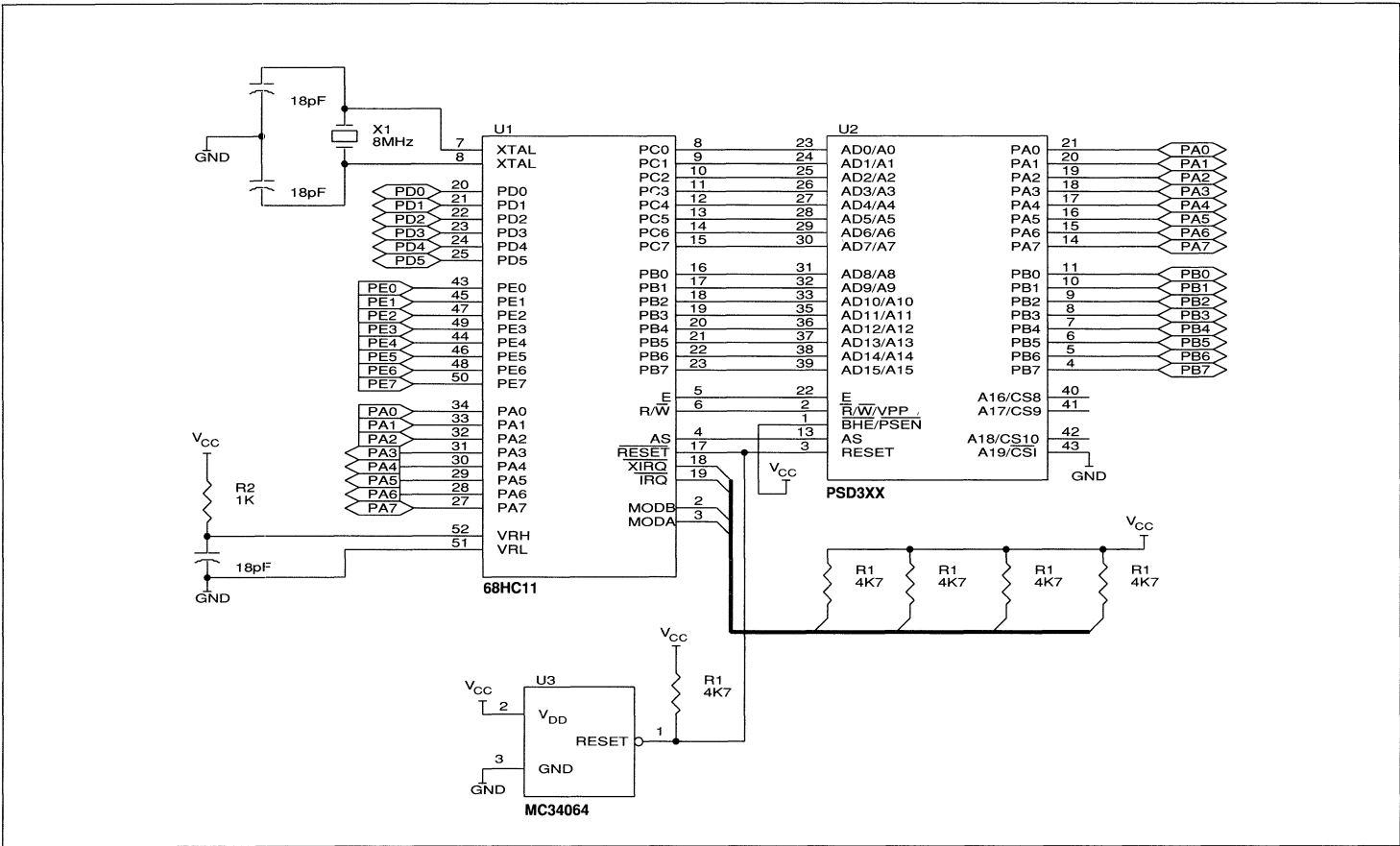
Figure 12 illustrates the configuration of an M68HC11 microcontroller which also uses the 8-bits wide multiplexed address/data bus. The application is similar to that given in Figures 6 and 7 except that the R/W and E control lines have been invoked to establish compatibility with the Motorola device. The address strobe output from the M68HC11 is HIGH so the AS(ALE) input is set HIGH. The SRAM and EPROM section are programmed as combined and both Ports A and B are enabled as I/Os with CMOS drives. Port C is programmed with chip-select outputs $\overline{CS8}$ – $\overline{CS10}$. Other PSD3XX devices can be mapped into the addressing scheme or the lines can be programmed to transition as strobes in defined mapping areas. The latch enable bit for the higher-order address lines A16–A19 is not used establishing a don't care condition. The CADDHLT condition must be selected if any one of A16–A19 lines is selected as input to the PSD.

In this design, the security bit is programmed. This bit prevents the reading of the PAD configuration by an unauthorized user. Furthermore, if the security bit has been programmed, standard programming machines can not read the internal code of a PSD3XX. However, data can always be read from the EPROM, RAM, and ports. This provides normal use of the device. If the address map in the PAD cannot be interpreted, the actual location of data within the address and I/O space is difficult to determine. Besides programming the CSECURITY bit, added security can be applied by scrambling the sequence of address and data inputs. A short PASCAL or 'C' program can be written to reorganize the original Intel MCS code to be aligned with the scrambled pins. Table 5 indicates the configuration for the M68HC11/PSD3XX interface.

Table 5.
M68HC11 to
PSD3XX Interface

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	1	Set R/W and E mode
CA19/CS \overline{I}	0	Enable \overline{CSI} input
CALE	0	Active HIGH AS (ALE)
CRESET	0	Active LOW RESET
\overline{COMB} /SEP	0	Combined memory mode
CPAF2	0	Input/Output Port A
CPAF1	00H	Input/Output Port A
CPBF	FFH	Input/Output Port B
CPCF	111B	Output $\overline{CS8}$ – $\overline{CS10}$
CPACOD	00H	CMOS drivers
CPBCOD	00H	CMOS drivers
CADDHLT	X	"Don't care" A16–A19 not used
CSECURITY	1	Security on

Figure 12.
68HC11/PSD3XX
Applications



8-BIT Non-Multiplexed PSD3XX Interface to M68008

Figure 13 illustrates an application in which the address and data are not multiplexed. The M68008 has an 8-bit data bus and 20-bit address bus. The PSD3XX can be programmed to support the microprocessor by providing data I/O through Port A. The address lines from the microprocessor go to inputs A0–A19. Port B outputs are used for external chip-selects to other MAP devices or other memory resources. The configuration has been set for compatibility with Motorola control signals. There are six chip-select outputs ($\overline{CS0}$ – $\overline{CS5}$) and an address decode for \overline{DTACK} and \overline{BERR} . The PAD decodes an address range which is fed back to the microprocessor through these inputs. Using the open-drain configuration has been implemented in Port B bits 6 and 7. The two pull-up resistors enable external memory and peripherals to access the \overline{DTACK} and \overline{BERR} inputs as a wired-OR function.

If other PSD3XX devices are mapped into the M68008 system, no additional glue logic is

needed to avoid possible bus contention on these lines. In this application, ALE(AS) can be used as a general-purpose logic input to the PAD because the function of ALE becomes redundant in a non-multiplexed address/data bus. Also shown in Figure 13 is a method of inverting the active LOW \overline{DS} (Data Strobe) M68008 output. The A19 input is enabled to the PSD internal PAD and inverted at the output of $\overline{CS10}$ to drive the PSD3XX E input. The E input must be active HIGH but \overline{DS} is active LOW and qualifies a valid data transfer. Thus, the PAD must perform a signal inversion. The E signal output from the M68008 is used to interface to Motorola 8-bit peripherals. However, with Motorola microcontroller families such as the M68HC11, the E signal output can drive the E input to the PSD3XX. Table 6 gives the configuration information associated with the design given in Figure 13.

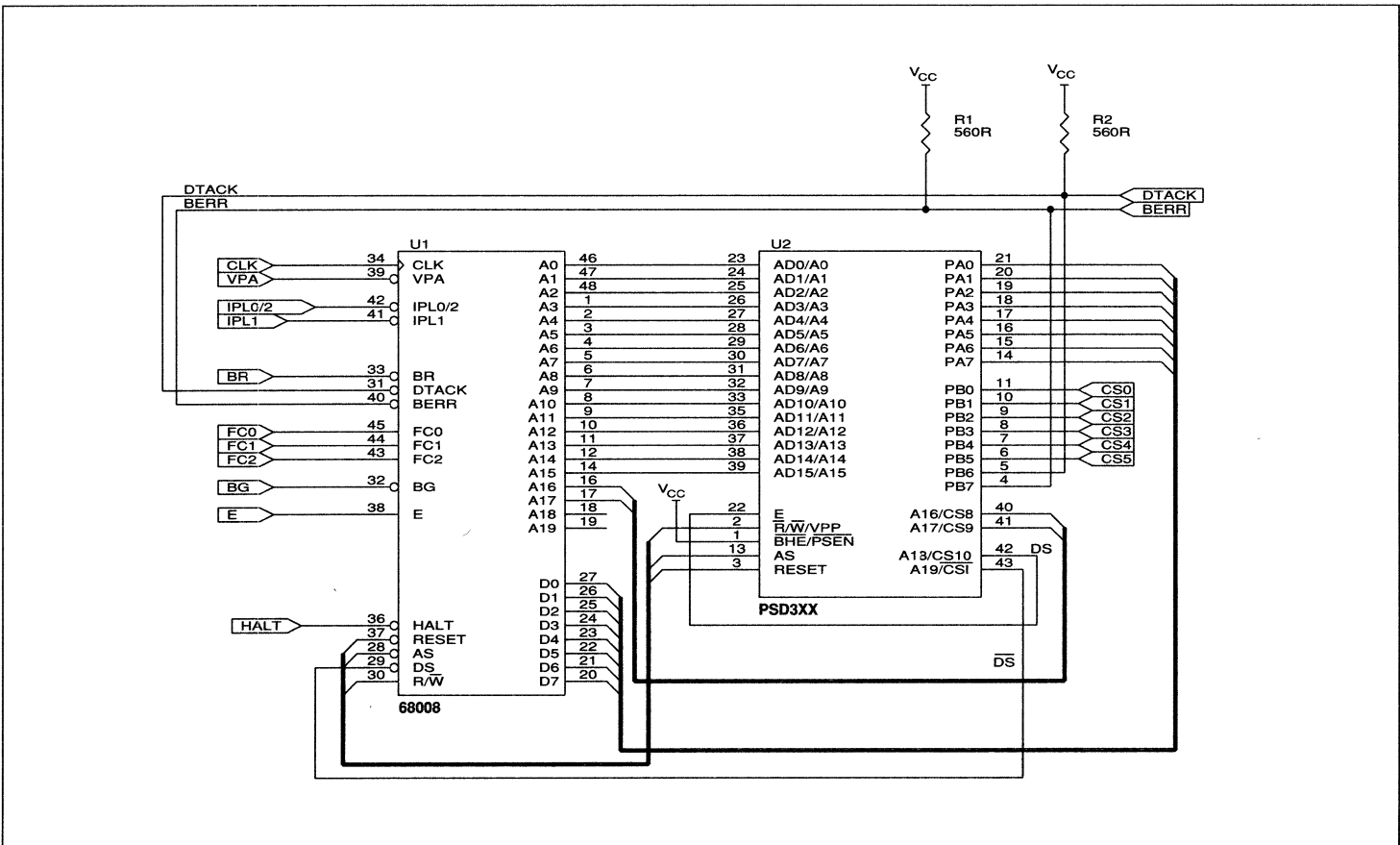
Table 6.
M68008 to
PSD3XX
Interface

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	0	Non-Multiplexed address/data
CRRWR	1	Set R/\overline{W} and E mode
CA19/ $\overline{CS1}$	1	Enable A19 input ¹
CALE	X	"Don't care" non-multiplexed mode
CRESET	0	Active LOW RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	X	"Don't care" Port A used for data
CPAF1	XXH	"Don't care" Port A used for data
CPBF	00H	Port B used for chip-selects
CPCF	001B	Configure A16 and A17 In, $\overline{CS10}$ Out ²
CPACOD	00H	CMOS drivers
CPBCOD	C0H	CMOS drivers, PB6, PB7 open drain
CADDHLT	0	Address latch transparent A16–A19
CSECURITY	1	Security on

1 The \overline{DS} output from the M68008 drives the A19 input to the PSD3XX

2 The internal PAD of the PSD3XX inverts the \overline{DS} input to drive its own E input from the $\overline{CS10}$ PAD output. A16 and A17 are programmed as PSD inputs

Figure 13.
M68008/PSD3XX
Applications



***This page
intentionally left blank***

***This page
intentionally left blank***

M68000/ 2X PSD3XX Applications

With the circuit design given in Figure 15, two PSD3XX devices are used in a byte-wide mode. One PSD stores the upper data byte and one the lower data byte of a 16-bit word. By using the devices in this way, two 6-bit wide ports can be created in Port B of each device. PB6 and PB7 are programmed as open-drain outputs and wired-OR giving

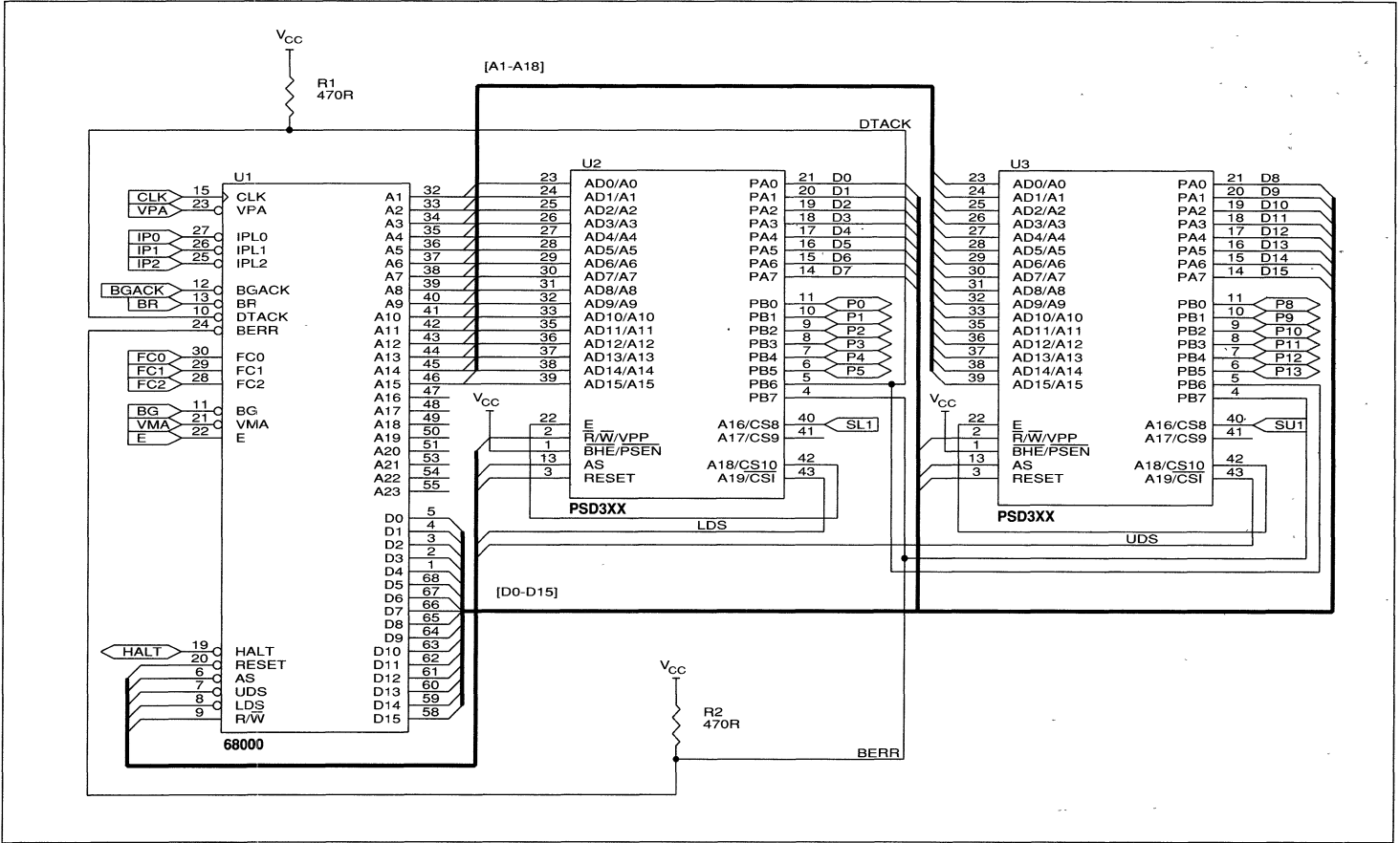
composite DTACK and BERR feedback signals to the M68000. The generation of the E signal for both PSD devices is achieved in the same way it was in the M68008. The \overline{LDS} and \overline{UDS} inputs (to U2 and U3 respectively) are inverted by the PAD and drive the relevant E inputs. Table 8 gives the configuration information relevant to both PSD devices.

Table 8.
M68000 Micro-
processor to Two
PSD3XX Devices
in Parallel

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	0	Non-multiplexed address/data
CRRWR	1	Set R/ \overline{W} and E control inputs
CA19/ $\overline{CS1}$	1	Enable A19 input ¹
CALE	X	"Don't care" not used
CRESET	0	Active LOW RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	X	"Don't care" Port A used for data
CPAF1	XXH	"Don't care" Port A used for data
CPBF	FFH	Port B used for I/O
CPCF	111B	Configure $\overline{CS8}$ - $\overline{CS10}$ ²
CPACOD	00H	CMOS drivers
CPBCOD	00H	CMOS drivers
CADDHLT	0	Transparent A19
CSECURITY	0	No security

1. A19 input to the PSD3XX's is used to receive \overline{UDS} and \overline{LDS} from the M68000 microprocessor. These signals are inverted by the PAD of each PSD3XX and fed back to the E input of each device.
2. $\overline{CS10}$ of each PSD3XX drives the inverted \overline{UDS} and \overline{LDS} back to E input. Port C is programmed to output $\overline{CS8}$ and $\overline{CS9}$. Additional byte-wide peripherals can be configured to the system and selected by these signals.

Figure 15.
M68000/
2X PSD3XX
Applications



16- Bit Address/ Data PSD3XX Interface to Intel 80186

Figure 16 and Table 9 give the configuration of the PSD3XX in an Intel 80186 system. This device has a 16-bit multiplexed address/data bus. Ports A and B are used for data I/O functions, so this design can take advantage of the port expansion capability. To distinguish between memory and I/O functions, it is necessary to decode the S2 output from the 80186. This output line goes directly to the PAD through Port C bit zero. When LOW, this signal qualifies a memory access; when HIGH, it indicates that an I/O operation is in progress. Programming the PAD can use this input to differentiate between I/O and memory access.

Two additional signals from the 80186 are UCS and LCS (upper chip-select and lower

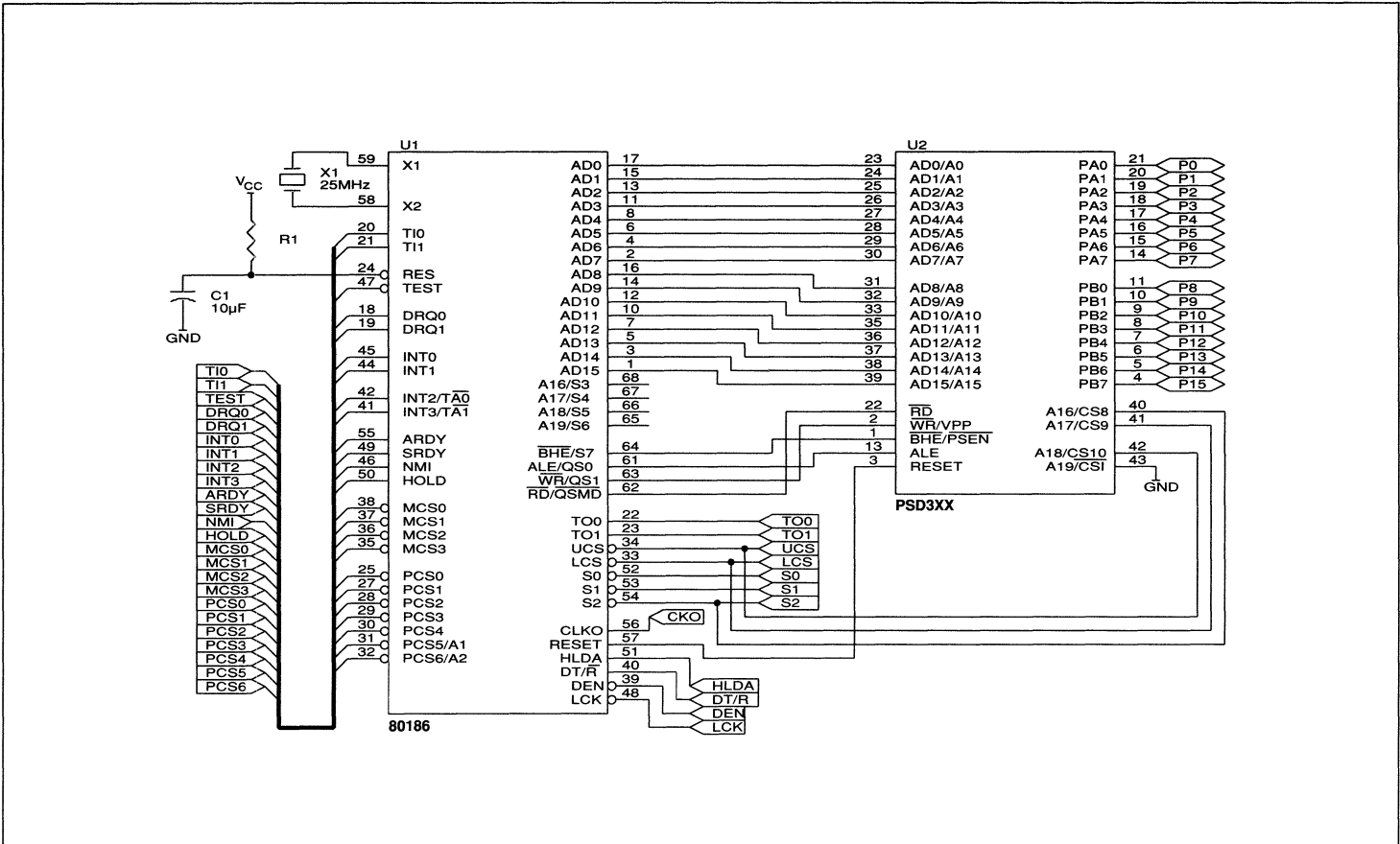
chip-select, respectively). The signals have been included in the system to help minimize the requirement for additional glue logic. Both can be used in the PAD decoder to position sections of EPROM and RAM. The UCS is designed to decode addresses FFFFH to a programmable limit. The 80186 begins executing from memory location FFFF0H after a system reset; thus, this signal should be used to select EPROM that contain a system initialization sequence. The LCS has been designed to program from 00000H up to a programmable limit. In this example, the RESET line from the 80186 is active HIGH and drives the RESET input of the PSD301 which is programmed to respond to a HIGH level.

1

Table 9.
Intel 80186 to
PSD3XX
Configuration
for CMOS Ports

Configuration	Bits	Function
CDATA	1	16-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	0	Set \overline{RD} and \overline{WR} mode
CA19/ \overline{CS} I	0	\overline{CS} I input to PAD
CALE	0	Active HIGH ALE
CRESET	1	Active High RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	0	I/O Port A
CPAF1	0	I/O Port A
CPBF	FFH	I/O Port B
CPCF	000B	Input A16-A18
CPACOD	00H	CMOS drivers
CPBCOD	00H	CMOS drivers
CADDHLT	0	Latched A16-A19
CSECURITY	0	No security

Figure 16.
Intel 80186/
PSD3XX
Applications



16-Bit Address/ Data PSD3XX to Intel 80196 Interface

In Figure 17, the PSD3XX is connected to an Intel 80196 microcontroller. In many microcontroller applications it is necessary to illuminate indicators (such as LEDs). Here, the PSD3XX is used to drive LED indicator

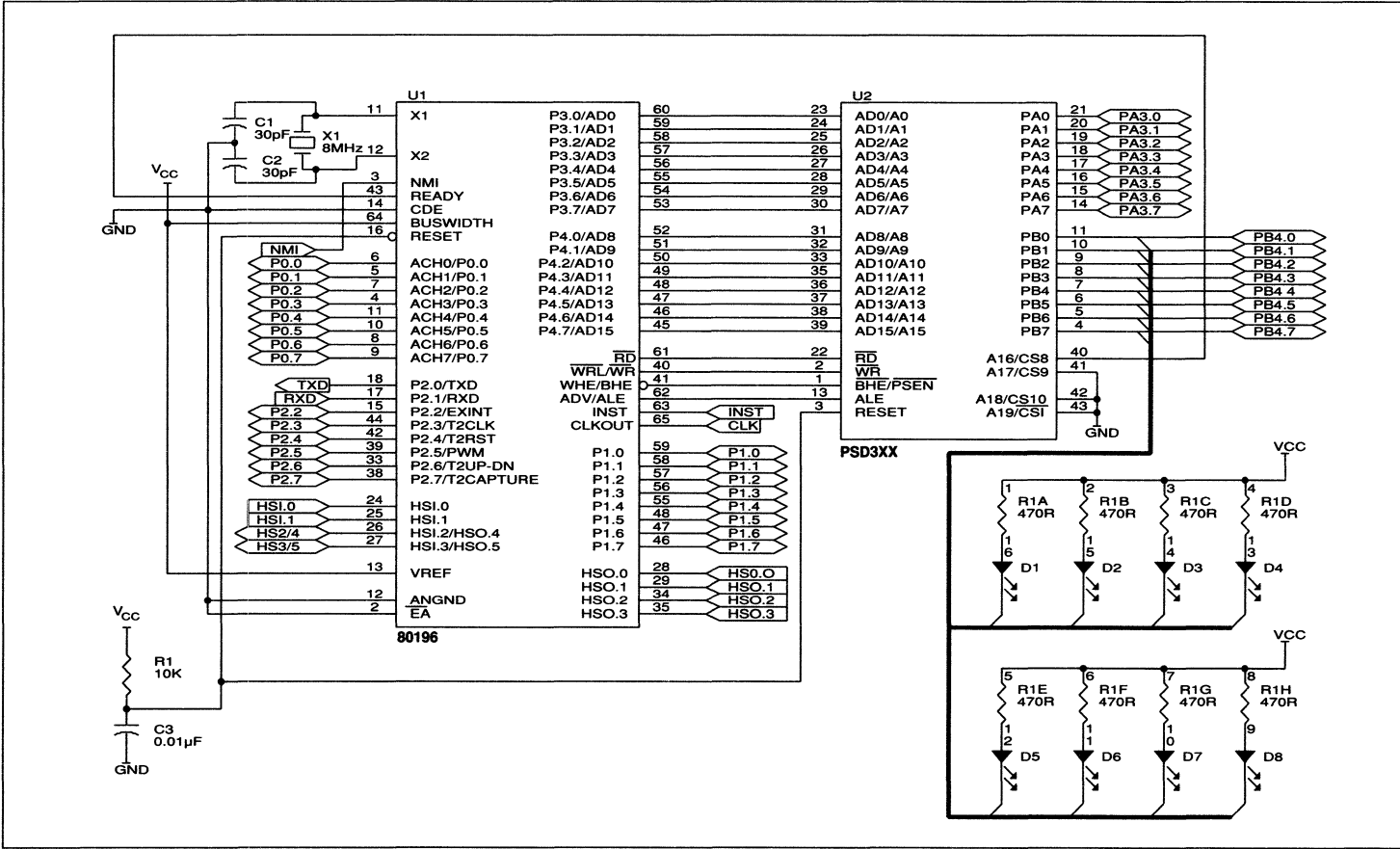
displays. High-efficiency LEDs can be illuminated through the open drain outputs of Port B. The configuration information in Table 10 indicates that Port B has open drain drivers to sink LED illumination current.

Table 10.
Intel 80196 to
PSD3XX
Configuration
for LED Drivers

Configuration	Bits	Function
CDATA	1	16-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	0	Set \overline{RD} and \overline{WR} mode
CA19/ \overline{CS} _I	0	"Don't care" A19/ \overline{CS} _I
CALE	0	Active HIGH ALE
CRESET	0	Active LOW RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	0	I/O Port A
CPAF1	00H	I/O Port A
CPBF	FFH	I/O Port B
CPCF	000B	Output A16–A18
CPACOD	00H	CMOS drivers
CPBCOD	FFH	Open drain drivers
CADDHLT	X	"Don't care" (not used)
CSECURITY	0	No security

1

Figure 17.
**Intel 80196/
 PSD3XX Applica-
 tion Open Drain
 Drivers**



Interfacing the PSD3XX to 8-Bit Microprocessors Z80 and M6809 Applications

Figures 18 and 19 illustrate the PSD3XX used with 8-bit microprocessors, such as the Z80B and M6809B. Tables 11 and 12 reflect the configuration of each design, respectively. The mode of operation is 8-bit data bus with a non-multiplexed address/data input. In the case of the Z80B, $\overline{CS8}$ – $\overline{CS10}$ inputs are tied to $\overline{M1}$, \overline{MREQ} , and \overline{IORQ} respectively. Since

the PAD can be programmed to distinguish between memory and I/O operations, the Z80B system has access to an 8-bit data port Port B. With the M6809B system, $\overline{CS8}$ is used to respond to the MRDY input of the microprocessor and $\overline{CS9}$ and $\overline{CS10}$ are available for external chip-select.

Table 11.
Z80B to PSD3XX Interface

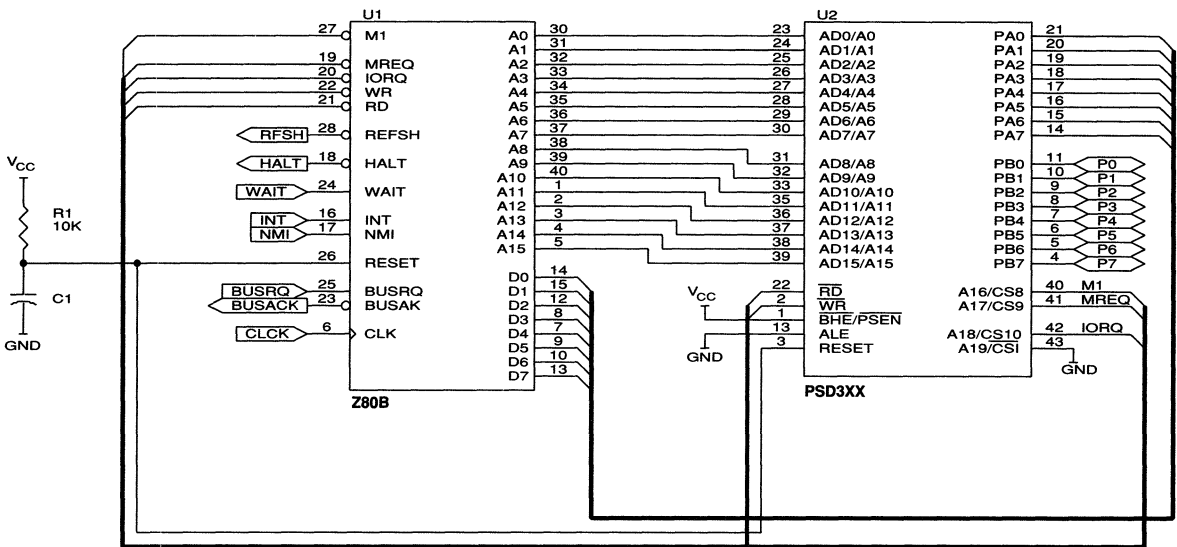
Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	0	Non-multiplexed address/data
CRRWR	0	Set \overline{RD} and \overline{WR} mode
CA19/ $\overline{CS1}$	0	$\overline{CS1}$ input
CALE	X	"Don't care" (not used)
CRESET	0	Active LOW RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	X	"Don't care" Port A used for data
CPAF1	XXH	"Don't care" Port A used for data
CPBF	FFH	I/O Port B
CPCF	000B	Configure A16–A18 as inputs
CPACOD	00H	CMOS drivers
CPBCOD	00H	CMOS drivers
CADDHLT	0	A16–A18 transparent ¹
CSECURITY	0	No security

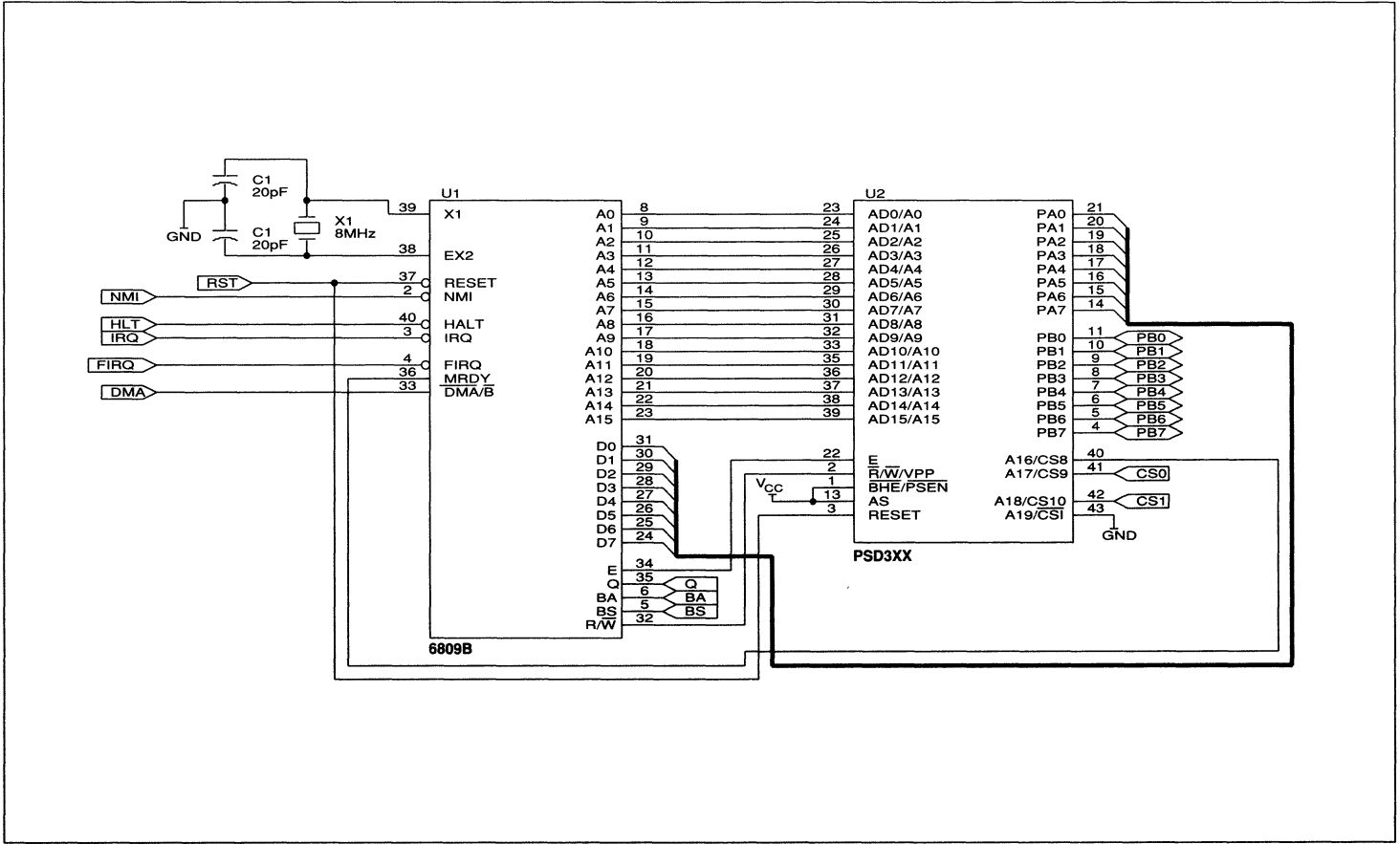
1. A16–A18 inputs are used as $\overline{M1}$, \overline{MREQ} , and \overline{IORQ} inputs to the PAD from the Z80B output. Use the ALIAS command in the support software.

Table 12.
M6809 to PSD3XX Interface

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	0	Non-multiplexed address/data
CRRWR	1	Set R/\overline{W} and E mode
CA19/ $\overline{CS1}$	0	Enable $\overline{CS1}$ input
CALE	X	"Don't care" non-multiplexed mode
CRESET	0	Active LOW RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	X	"Don't care" Port A used for data
CPAF1	XXH	"Don't care" Port A used for data
CPBF	FFH	Port B used for I/O
CPCF	111B	$\overline{CS8}$ – $\overline{CS10}$ outputs
CPACOD	00H	CMOS drivers
CPBCOD	00H	CMOS drivers
CADDHLT	0	"Don't care"
CSECURITY	0	No security

Figure 18
Z80B/PSD3XX
Applications





PSD3XX Interface to the Intel 80286

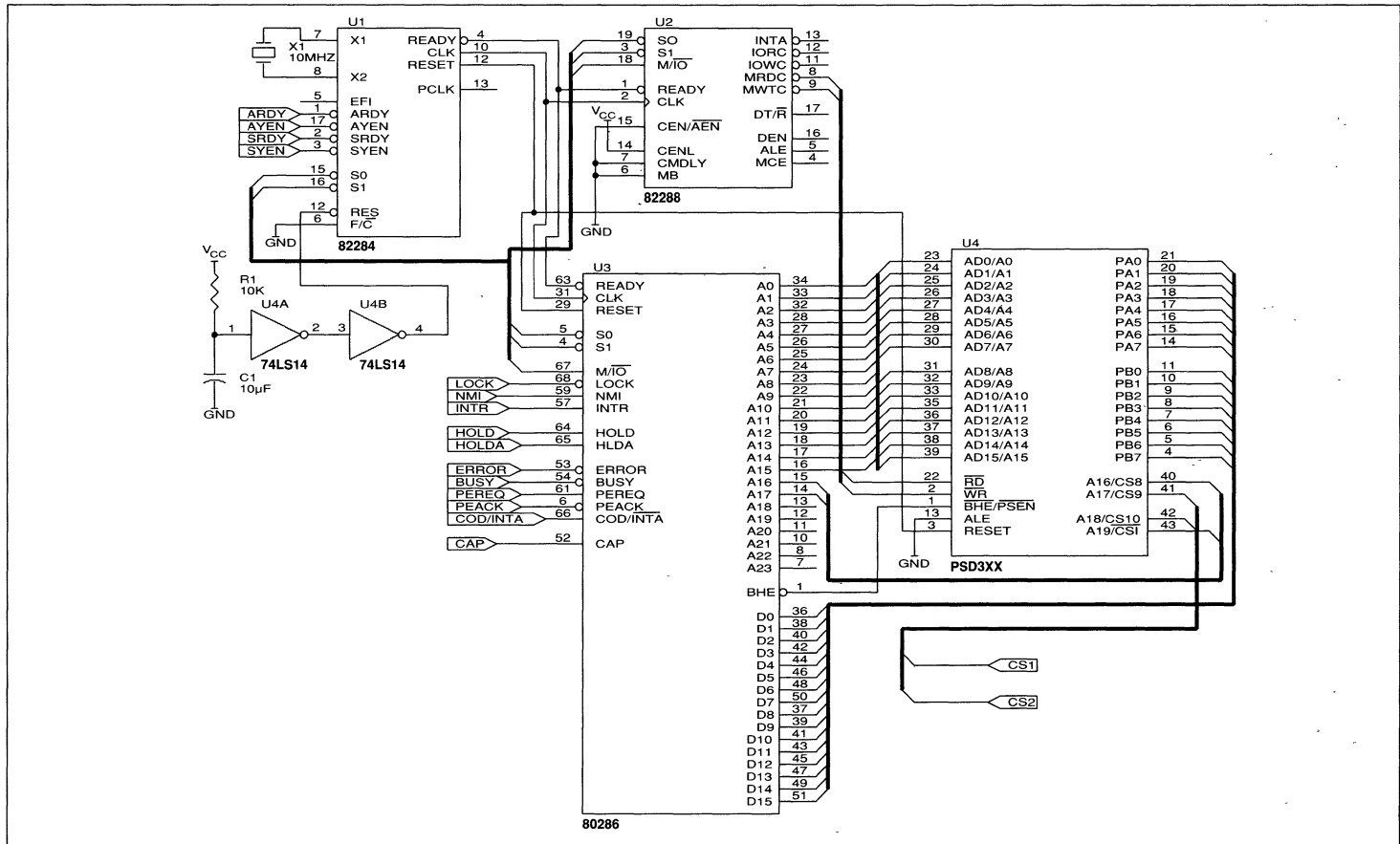
Figure 20 provides a schematic of the PSD3XX interface to an 80286. The device is configured for a 16-bit data bus in the non-multiplexed mode. Ports A and B are converted automatically for use as a bi-directional data path into the PSD3XX. (This was also

the case for the M68000 microprocessor). To eliminate (or lessen) glue logic, $\overline{CS1}$ and $\overline{CS2}$ are generated from the internal PAD. This is programmed as an address decoder. Table 13 provides configuration information relevant to this system design.

Table 13.
Intel 80286 to
PSD3XX Interface

Configuration	Bits	Function
CDATA	1	16-bit data bus
CADDRDAT	0	Non-multiplexed address/data
CRRWR	0	Set \overline{RD} and \overline{WR} control inputs
CA19/ $\overline{CS1}$	1	Enable A19 input
CALE	X	"Don't care" non-multiplexed mode
CRESET	1	Active HIGH RESET
$\overline{COMB/SEP}$	0	Combined memory mode
CPAF2	X	"Don't care" Port A used for data
CPAF1	XXH	"Don't care" Port A used for data
CPBF	XXH	"Don't care" Port B used for data
CPCF	011B	A16 input; $\overline{CS9}$ and $\overline{CS10}$ outputs
CPACOD	00H	CMOS drivers
CPBCOD	00H	CMOS drivers
CADDHLT	0	Transparent A16-A19 input
CSECURITY	0	No security

Figure 20
Intel 80286/
PSD3XX
Applications



External Peripherals to the PSD3XX/M68HC11 Configuration

The configuration in Figure 21 illustrates how the user can feed address outputs from the internal latch to Port A. Addresses A0–A7, derived from a multiplexed address/data bus, can go directly to an additional peripheral without the need for an additional octal latch such as the 74HC373 or 74HC573. Port A can be used for address outputs A0–A7 while PB0–PB7 can be used as chip-selects. Lines A0–A4 of the PSD3XX drive the RS1–RS5 register select inputs of the M68230. For the M68HC11, the eight bits of address and data come from its PC port PC0–PC7 (AD0–AD7) and are latched by the AS input. Configured in this mode, the PSD3XX can address and map additional peripheral chips. Port A of the PSD3XX conveys the internally latched

address outputs A0–A7 to the output and can be used to address registers in the peripheral chips while Port B outputs can place individual peripherals at peripheral or memory-mapped boundaries. Thus, a number of additional chips can be selected through Port B. This effectively can increase the port density of the system design. The general I/O capability can then be extended to extra ports, timers, UARTs, serial communications channels, keyboard interface devices, CRT controllers, etc. without the need for additional glue logic. Table 14 highlights the configuration information programmed into the PSD3XX when configuring the M68HC11 to a M68230 peripheral.

Table 14.
M68HC11/PSD3XX
to External
Peripheral
M68230
Interface

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	1	Set R \bar{W} and E mode
CA19/ \bar{CS} I	0	Set power-down mode
CALE	0	Active HIGH AS
CRESET	0	Active LOW RESET
\bar{COMB} /SEP	0	Combined memory mode
CPAF2	0	Port A = address A0–A7
CPAF1	FFH	Port A set for address
CPBF	00H	Port B set for chip-select
CPCF	111B	Port C set for chip-select
CPACOD	00H	CMOS buffers
CPBCOD	00H	CMOS buffers
CADDHLT	X	"Don't care"
CSECURITY	0	No security

Additional External SRAM

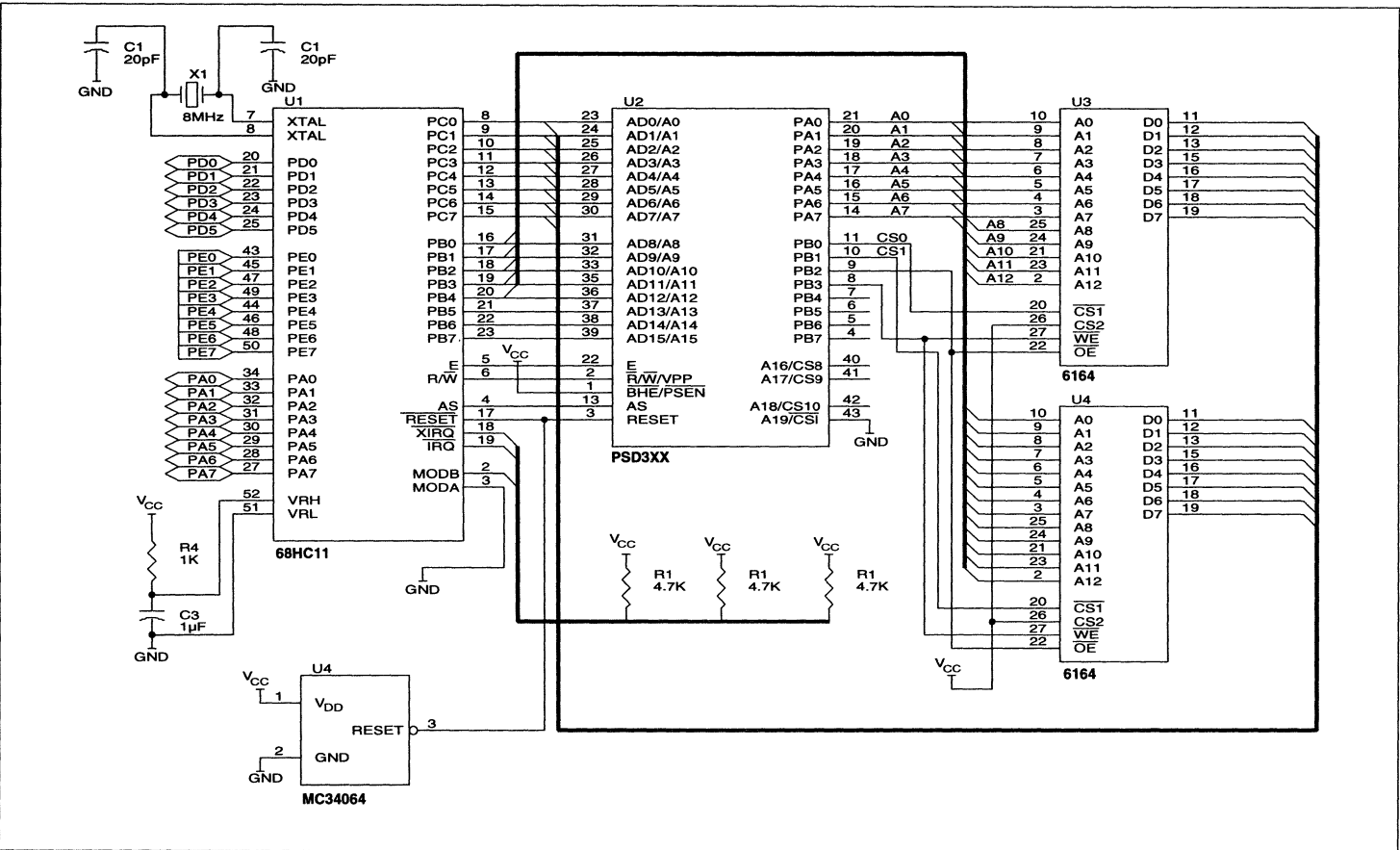
Figure 22 illustrates how additional SRAMs can be configured into a system. This PSD3XX configuration is not limited to external peripheral expansion; it can also be used to add additional memory without the need for external glue logic. With an 8-bit address/data multiplexed scheme, the higher-order addresses (A8–A15) are non-multiplexed. These address lines are fed directly to the

external SRAM from the microcontroller and do not need to go through the PSD3XX. These lines can drive the RAM chip directly. Thus the M68HC11 system, which is highly memory-intensive and requires more RAM than the microcontroller and PSD3XX can supply, can take advantage of the configuration shown in Figure 23 which is detailed in Table 15.

Table 15.
**M68HC11/
PSD3XX
Configured to
Address
Additional
SRAM**

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	1	Set R/ \overline{W} and E mode
CA19/ \overline{CS} I	0	Set power-down mode
CALE	0	Active HIGH AS
CRESET	0	Active LOW RESET
\overline{COMB} /SEP	0	Combined memory mode
CPAF2	0	Port A = address A0–A7
CPAF1	FFH	Port A set for address
CPBF	00H	Port B set for chip-select
CPCF	111B	Port C set for chip-select
CPACOD	00H	CMOS buffers
CPBCOD	00H	CMOS buffers
CADDHLT	X	Latched A16–A19 "don't care"
CSECURITY	0	No security

Figure 22.
M68HC11/PSD3XX
to 16K SRAM
Applications



**Additional
External SRAM
(Cont.)**

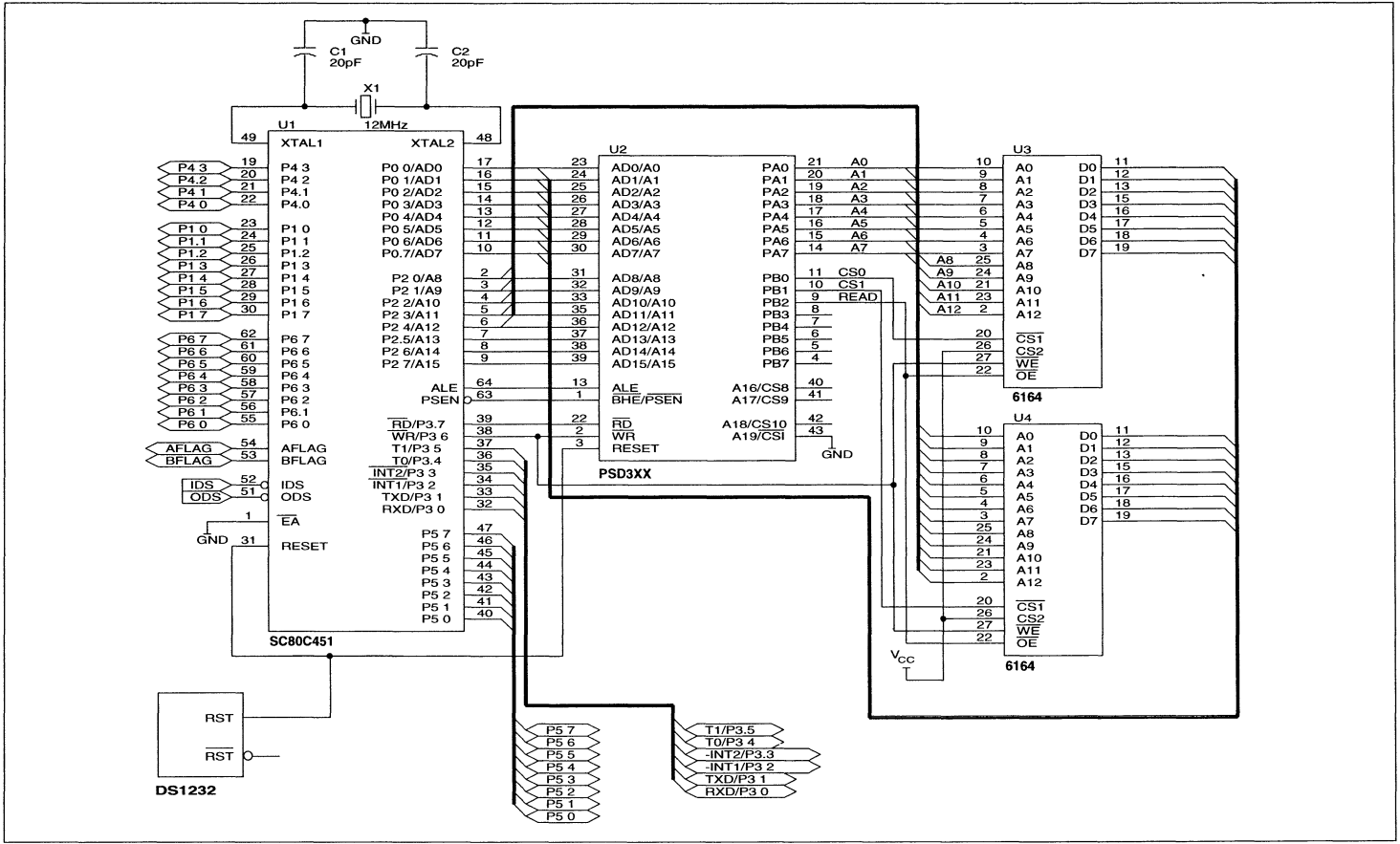
Figure 23 illustrates, and Table 16 details, a similar system using the Signetics SC80C451. This microcontroller has many ports and some SRAM but requires off-chip EPROM to store programmed instructions. This device is similar to the 8051/31 family which uses the active LOW PSEN signal to differentiate between executable code and

data. Since it is a multiplexed 8-bit machine, it can use the on-chip latches. In highly RAM-intensive applications, an additional two 8K x 8 SRAM chips can be included and selected through Port B. If additional SRAM chips are not needed, Ports A and B can recreate Ports 0 and 2 which are lost in addressing external memory.

**Table 16.
SC80C451/
PSD3XX
Configured to
Address
Additional
SRAM**

Configuration	Bits	Function
CDATA	0	8-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	0	Set RD and WR mode
CA19/CSI	0	Set power-down mode
CALE	0	Active HIGH ALE
CRESET	1	Active HIGH RESET
COMB/SEP	1	Separate data/program memory
CPAF2	0	Port A = address A0-A7
CPAF1	FFH	Port A set for address
CPBF	00H	Port B set for chip-select
CPCF	111B	Port C set for chip-select
CPACOD	00H	CMOS buffers
CPBCOD	00H	CMOS buffers
CADDHLT	0	"Don't care" (not used)
CSECURITY	0	No security

Figure 23.
**SC80C451/
 PSD3XX
 to 16K SRAM
 Applications**



PSD3XX Used in Track Mode

Figure 24 illustrates a design that utilizes the track mode of operation that has been discussed but not illustrated in an application. Here, Port A passes or tracks through the multiplexed address and data of the 80196. Address and data outputs AD0-AD7 from the 80196 appear on the PSD3XX Port A pins. In this mode, the SRAM, shown in Figure 24 as U4, can be accessed either by the 80196 (used in byte mode) or by a second processor in the host system. The SRAM in the design can be used as a common resource. An example would be a system in which the host uses the memory to pass parameters to the local 80196. Table 17 gives the configuration data for an 80196/PSD301 interface to SRAM using Track Mode.

A Direct Memory Access can transfer data to the common memory via a BUSRQ/BUSGR handshake. Note that the PAD in the PSD3XX controls the three-state condition of the octal latch U3 74HCT373 enabling the host system to control SRAM addresses A0-A7. Port A of the PSD3XX is also put into

a three-state condition during host-to-SRAM activity. In the design given in Figure 24, Port B outputs PB0, PB1, and PB2 are used to control the SRAM inputs CE, OE, and WR respectively. Also, A8, A9, and A10 are fed through the PAD as identity functions to the open drain drivers of PB3, PB4, and PB5 respectively. There is no track-through feature for these address lines; however, if they are fed through the PAD, they can drive the external memory resource as if they were tracked through.

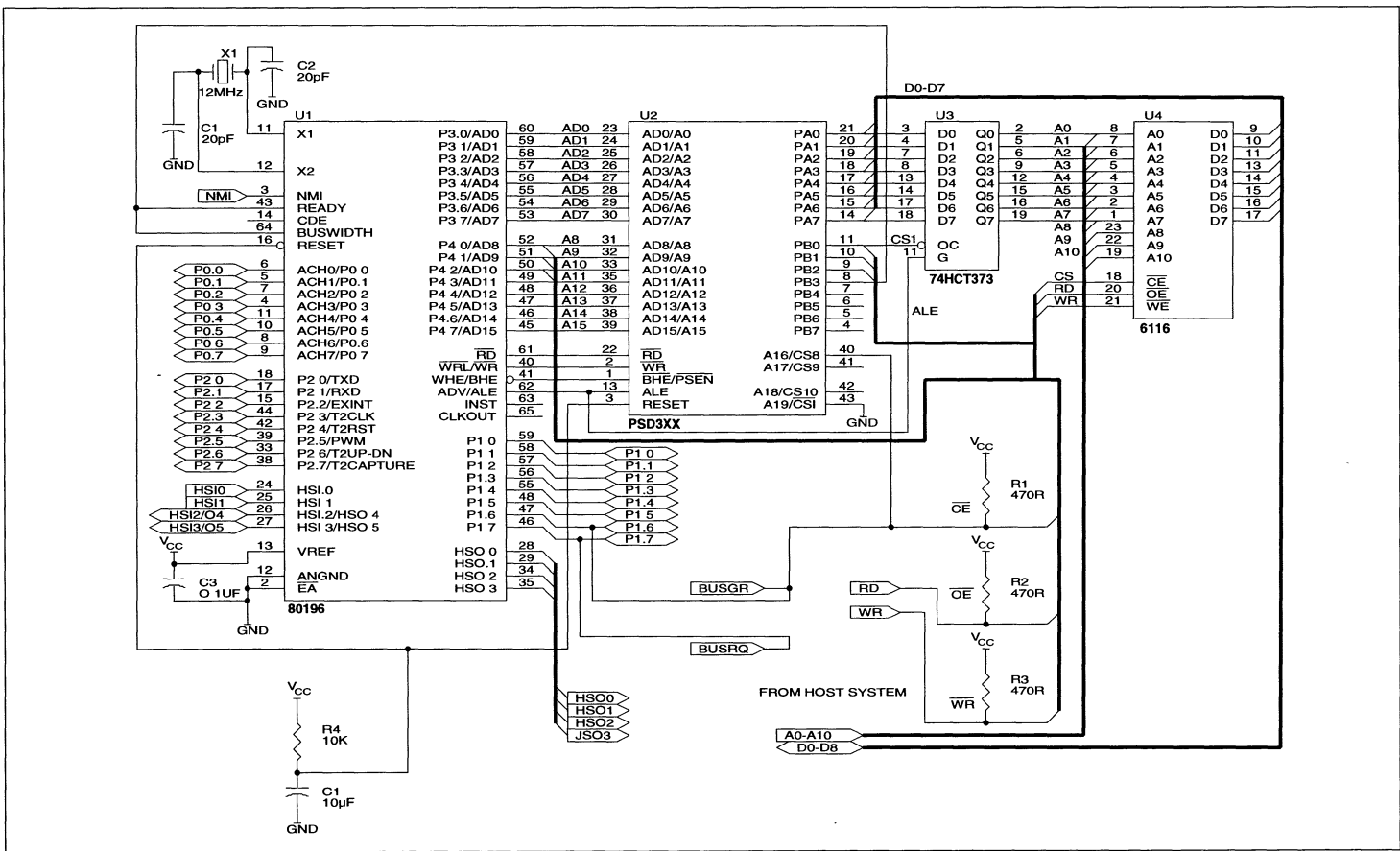
The M80196 can operate in either byte- or word-wide mode controlled by its BUSWIDTH input. In this application, the PB6 output drives the BUSWIDTH line to switch between the byte-wide bus of the external SRAM and the word-wide interface of the PSD3XX. All Port B outputs, with the exception of PB6, are configured as open-drain. Provided the host system also has open drain/ collector drivers, both systems can access the SRAM without bus conflict. The only additional circuitry required would be the pull-up resistors.

**Table 17.
Intel 80196 to
PSD3XX Used to
Access External
SRAM in Track
Mode**

Configuration	Bits	Function
CDATA	1	16-bit data bus
CADDRDAT	1	Multiplexed address/data
CRRWR	0	Set RD and WR mode
CA19/CSI	0	Set power-down mode
CALE	0	Active HIGH ALE
CRESET	0	Active LOW RESET
COMB/SEP	0	Combined memory mode
CPAF2	1	Address/data (Track Mode)
CPAF1	XXH	"Don't care" in Track Mode
CPBF	00H	Port B set for chip-select outputs
CPCF	111B	Port C set for logic outputs
CPACOD	00H	CMOS buffers
CPBCOD	FFH	Open drain buffers
CADDHLT	X	Latched A16-A19 "don't care"
CSECURITY	0	No security



Figure 24.
**Intel 80196/
 PSD3XX Track
 Mode to External
 SRAM**



Programmable Peripheral Application Note 011 Software Support

Chapter 3

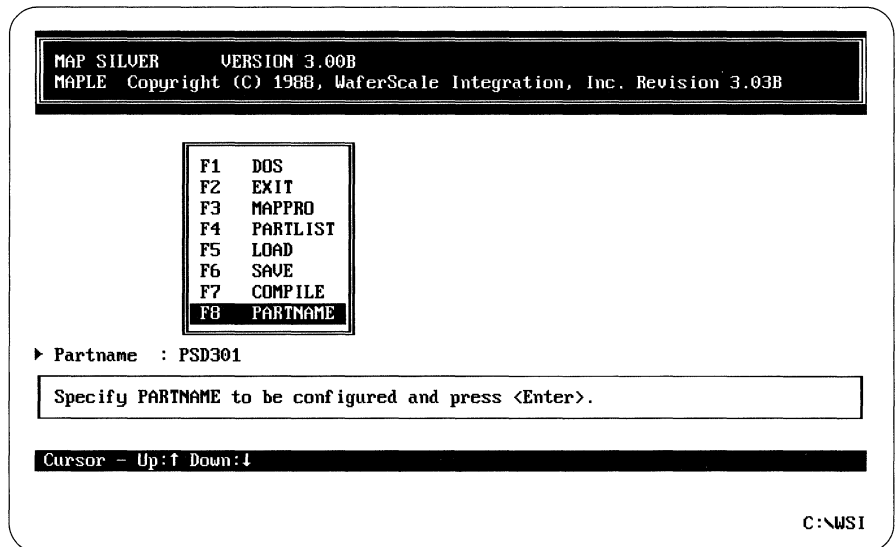
The support software for both PSD3XX family and MAP168 memory-mapped peripheral devices is designed to run on IBM PC XT/AT or 100% compatible systems. It is menu-driven and very user-friendly. In many cases it has the capability of preventing the user from creating invalid configurations. For example, in a non-multiplexed system with a 16-bit data bus, Ports A and B are used for data I/O. The software recognizes this and prevents the

user from inadvertently programming Ports A and B as regular ports.

When running in the IBM PC environment, the PSD development software creates the menu shown in Figure 25. Initially, the designer selects the part type with the user key F8 or moves the screen cursor to PARTNAME. In the example shown, the selection for the part type is PSD301.

1

Figure 25.
MAPLE Main
Menu



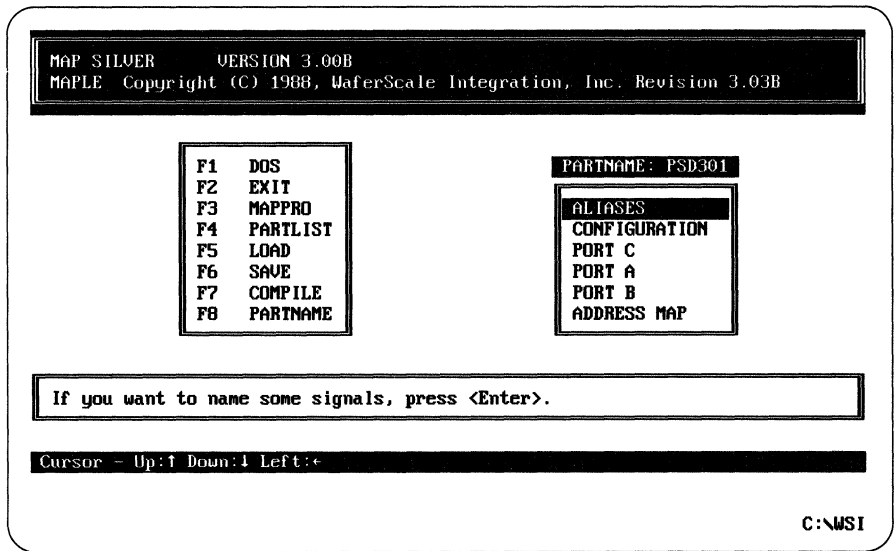
2402 25

The menu listed to the left of Figure 25 links the function keys and their association. F1 suspends the MAPLE software to DOS for file editing or updating. F2 exits the program and returns the user to the DOS environment. F3 selects the programmer option so the user can program the compiled object file into the PSD301 device provided a programmer is connected to the system. The LOAD selection (F5), loads an existing program into the MAPLE environment for editing and compiling. F6 saves that program under a user-

defined name. F7 compiles the user-generated file into an object file that can be transferred to the programmer. F8 provides part type selection, either PSD301 or MAP168.

Figure 26 illustrates a second menu to the right of the main menu. The list shows ALIASES, CONFIGURATION, PORT C, PORT A, PORT B, and ADDRESS MAP. The designer selects each choice, starting from ALIASES, and moves down through the list configuring each option.

Figure 26.
MAPLE Menu
with PARTNAME
Submenu



2402 26

ALIASES Menu

The ALIASES selection lets the user individually define the port pins with user-relevant names. The circuit diagram shown in Figure 13 uses an M68008 processor, with BERR and

DTACK signals coming from the PAD, as well as the remaining CS0, CS5 chip-select outputs.

Figure 27. CONFIGURATION Menu

Figure 27 gives the CONFIGURATION menu. In this case, the PSD301 has been configured for the system shown in Figure 10: interfacing to an 80C31; the 8-bit data/address bus is multiplexed. The chip-select input is chosen over the A19 input. The RESET and ALE polarity is set as active HIGH with \overline{RD} and \overline{WR} control inputs enabled. The inputs A16-A19 are transparent and separate strobes are enabled for SRAM and EPROM.

This feature activates the \overline{PSEN} input. In this configuration it is possible for the SRAM and EPROM to share the same address space. After the device is configured, Ports A, B, and C can be set up. If the main menu is invoked by selecting F1 (Figure 28), Port C can be selected as shown in Figure 26. Here, the individual selection of \overline{CS}/A_i configures the three pins as outputs.

CONFIGURATION

Address/Data Mode (Multiplexed: MX, Non-Multiplexed: NM)	MX
Data Bus Width (8/16 bits)	8
CSI (Power-Down/Chip Enable) or A19	CSI
Reset Polarity (Active Low: LO, Active High: HI)	HI
ALE Polarity (Active Low: LO, Active High: HI)	HI
\overline{WR} and \overline{RD} (\overline{WRD}) or $\overline{R/W}$ and \overline{E} (\overline{RWE})?	\overline{WRD}
A19-A16 Transparent or Latched by ALE (Trans: T, Latched: L)	T
Using different READ Strobes for SRAM and EPROM ? (Y/N)	Y
Separate SRAM and EPROM address spaces ? (Y/N)	Y

If SRAM and EPROM share the same Address space, press SPACEBAR.

F1-Return to Main Menu F2-Temporary exit to Dos Cursor- Up:↑ Down:↓

C:\NSI

2402 27

Figure 30.
Port A
Configuration
Menu, Part 2.

Port A can be programmed to be either address I/O or track mode, as illustrated in Figure 30. Track mode is selected if the

designer wants to program the device as shown in Figure 24.

PORT A

Prev Config:→ ADDRESS/I/O
TRACK MODE

If you want to configure PORT A pins individually as Address or I/O bits, press <Enter>.

F1 - Return to Main Menu F2 - Temporary exit to Dos.
Cursor - Up:↑ Down:↓

C:\NWSI

2402 30

Figure 31.
Port B
Configuration
Menu

Figure 31 gives the configuration of Port B. This is similar to the configuration pattern for the M68008 shown in Figure 13. Here, CS6

and CS7 have been programmed as open-drain outputs connected to the micro-processor's DTACK and BERR, respectively.

PORT B

PIN	CS/I/O	CMOS/OD
PB0	CS0	CMOS
PB1	CS1	CMOS
PB2	CS2	CMOS
PB3	CS3	CMOS
PB4	CS4	CMOS
PB5	CS5	CMOS
PB6	CS6	OD
PB7	CS7	OD

If you have CMOS output for PB7 press SPACEBAR.

F1 - Return to Main Menu F2 - Temporary exit to Dos
F3 - Goto CS Definition Cursor - Up:↑ Down:↓ Left:← Right:→

C:\NWSI

2402 31

Figure 33.
Port B
Configuration
Menu with
Address Map

PORT B													
PIN	CS/IO	CMOS/OD	CHIP SELECT DEFINITION CS0										
PB0	CS0	CMOS	A18	A17	A16	A15	A14	A13	A12	A11	ALE	RD	WR
PB1	CS1	CMOS	0	0	0	1	1	1	1	1	X	X	X
PB2	CS2	CMOS											
PB3	CS3	CMOS											
PB4	CS4	CMOS											
PB5	CS5	CMOS											
PB6	CS6	CMOS											
PB7	CS7	CMOS											

CS definition is the NOR of the product terms(rows). Enter 1 to select Active High signal, 0 to select Active Low signal, X to mean 'don't care', SPACEBAR to erase. Enter values in columns relevant to your application; other blank columns will be treated as 'don't care's.

F1 - Return to PORT B Cursor - Up:↑ Down:↓ Left:← Right:→

C:\MSI

2402 31

Summary

The PSD3XX microcontroller peripheral with memory, supported with low-cost software and programming capability form WSI, greatly simplifies the overall design of microcontroller based systems. The key advantage is the extensive condensing of glue logic, latches, ports, and discrete memory elements into a single-device,

enhancing the reliability of the final product. Applications for the device extend to practically any area that uses microcontrollers or microprocessors, from modems and vending machines to disc controllers and high-end processor systems.



Programmable Peripheral

Application Note 013

The PSD301 Streamlines a Microcontroller-based Smart Transmitter Design

By Seyamak Keyghobad – Bailey Controls,
and Karen Spesard – WSI

Abstract

A smart transmitter design is described which takes advantage of the integration capabilities and flexibility of WSI's PSD301 microcontroller peripheral. The following discussion illustrates how the

PSD301, in effect, was responsible for eliminating an extra 2.5 inch diameter board in a system where real estate is at a premium by reducing the number of components from 12 down to 5.

1

Introduction

Designers of systems using microcontrollers and microprocessors often face the problem of how to integrate peripheral logic and memory functions into their designs without using many discrete chips and large areas of board space. For example, when external EPROM and SRAMs are configured into systems with ROMless microcontrollers, general I/O ports are typically sacrificed for address, data input/output, and control functions. When these I/O ports are depleted, the total chip count of the system is increased by requiring the use of additional external ports and steering logic. Designers, who have limited board space, such as found in the disk drive,

modem, cellular phone, industrial/process control, and automotive industries, find this a critical problem.

The PSD301 programmable peripheral device from WSI solves this problem by integrating all SRAM, EPROM, programmable decoding and configurable I/O port functions needed in 8 or 16-bit microcontroller designs into a single-chip user-configurable solution. This is illustrated in the following industrial control application where the PSD301 eliminates seven chips and saves the designer from needing another board in the system.

The Design Application

The smart transmitter, shown in Figure 1, was developed by Bailey Controls, a manufacturer of process control instruments, to support a popular field bus protocol. One of its functions in this sensor application is to measure pressure, differential pressure, and flow rates through pipes in industrial environments such as chemical plants, oil refineries, or utility plants. A host system monitors the transmitter via a process control network.

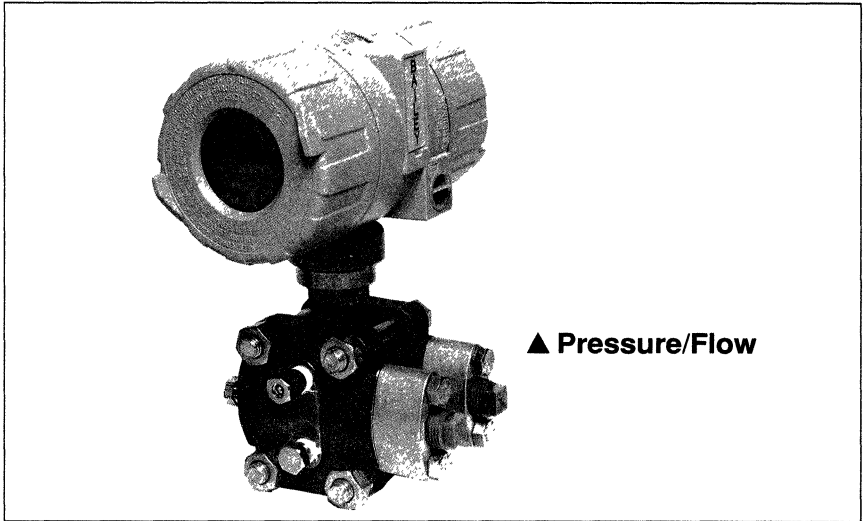
The completed transmitter design consists of three main boards. The first board includes the power supply and communications hardware to provide power to the rest of the system and feedback to the process control network. It consists of communications transformers and line drivers/receivers.

The second board is the digital microcontroller board and contains the 68HC11 microcontroller as well as the PSD301 programmable peripheral, a PLD, UART, and LCD display. Its function is to communicate and receive the inputs from the third board, process the data, and display the appropriate results to the LCD.

The third board or input board is mostly analog. It receives inputs from string gauge sensors which use a bridge circuit for measuring pressure using a diaphragm. The input board then converts the signals so the microcontroller can read them.



Figure 1.
"Smart"
Transmitter from
Bailey Controls



Design
Considerations

The smart transmitter system is rather small. Its case is only 2.5 inches in diameter and thus requires boards that fit this small form factor as shown in Figure 2. Not surprisingly, the major design consideration during development was board space. This was especially true for the microcontroller/digital board where real estate is at a very high premium.

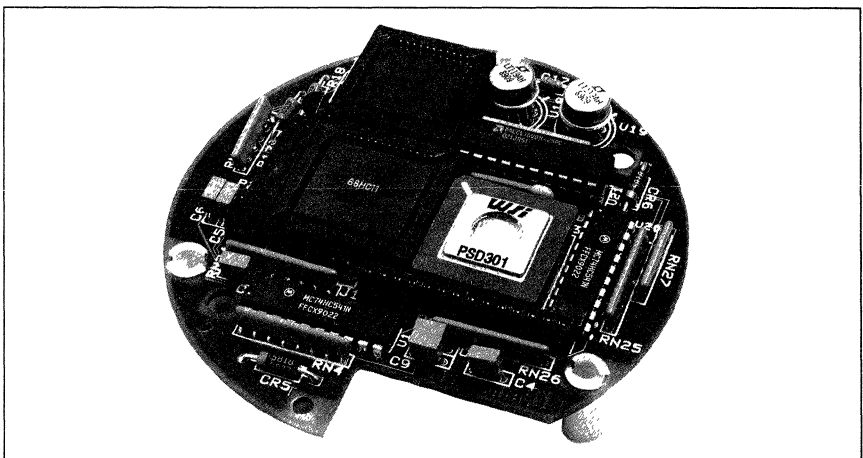
One of the problems was that there were already requirements for the 68HC11 microcontroller, a 256K EPROM, 16K SRAM, a PLD, TTL logic, a UART, and an LCD display on the digital board. This

meant extending the number of boards used beyond one unless a way could be found to integrate some of these elements.

Other important considerations, or goals actually, for the design were to reduce power consumption to less than 2.4W, improve reliability, lower design costs, and shorten the time-to-market.

To meet these objectives, Bailey Controls looked to WSI's user-configurable peripheral, the PSD301, for its integration capabilities, its flexibility, and its low power of less than 35 mA active and 90 μ A typical powerdown.

Figure 2.
The Bailey Smart
Transmitter Board
Using the WSI
PSD301.



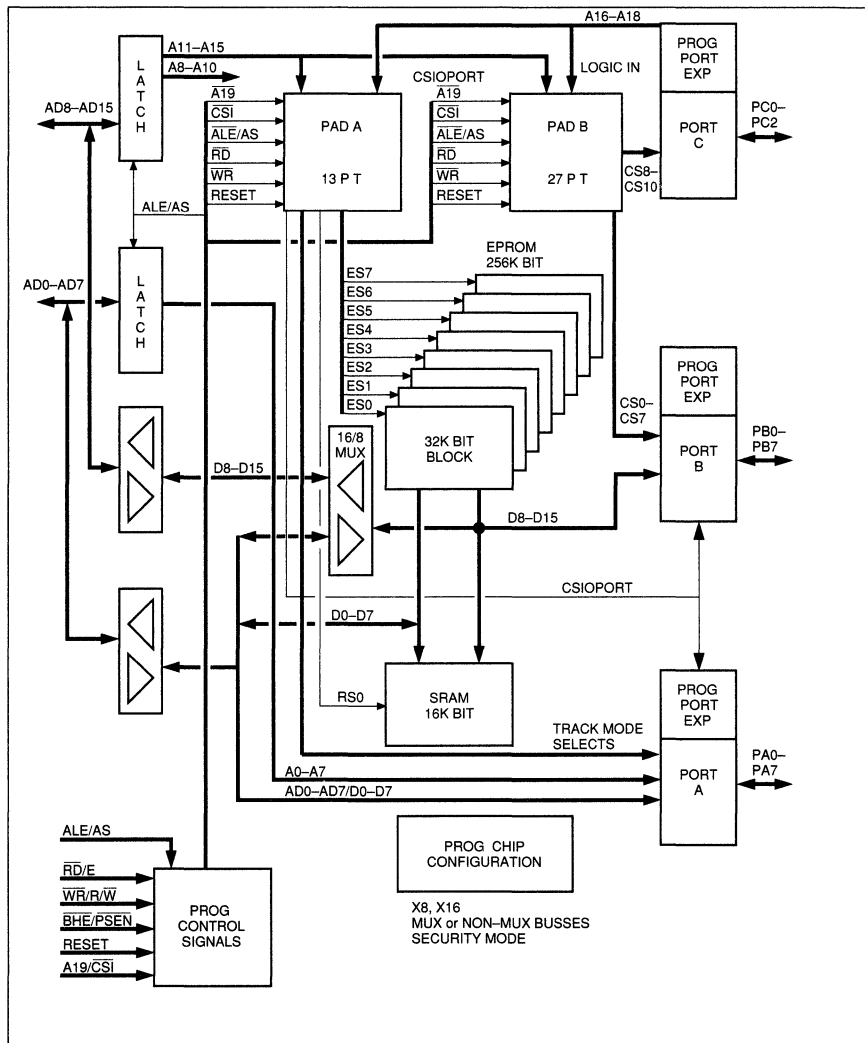
**PSD301
Architecture**

The PSD301 is a field programmable device that has the ability to interface to virtually any 8- or 16-bit microcontroller without the need for external glue logic. This is possible because the PSD301 combines the elements necessary for a complete microcontroller peripheral solution, such as user-configurable logic, I/O ports, EPROM and SRAM, all into one device. The functional block diagram of the PSD301 in Figure 3 shows its main sections: the internal latches and control signals, the programmable address decoder (PAD), the memory, and the I/O ports.

The control signals and internal latches in the PSD301 were designed so interfacing to any microcontroller would be easy and require no glue logic. For instance, the PSD301 can interface directly to all multiplexed (and non-multiplexed) 8- and 16-bit microcontroller address/data buses because it has two on-chip 8-bit address latches. This means no external latches are required to interface to multiplexed buses. It also has programmable polarity on the control inputs ALE/AS and RESET, so they can be configured to be active high or active low.

1

**Figure 3.
PSD301
Architecture**



**PSD301
Architecture
(Cont.)**

The other control signals, \overline{RD}/E , and $\overline{WR}/R/\overline{W}$, are also programmable as \overline{RD} and \overline{WR} or E and R/\overline{W} , enabling direct interface to all Motorola- and Intel-type controllers.

The programmable array decoder (PAD) is an EPROM-based reprogrammable logic "fuse" array with 11 dedicated inputs, up to 4 general-purpose inputs, and up to 24 outputs. The PAD is used to configure the 8 EPROM blocks on 2K word boundaries and the SRAM on a 1K word boundary anywhere within a 1 Meg address space. It is also used to generate a base address for mapping ports A and B, as well as to provide mapping for the track mode. The PAD, like a traditional PLD, can generate up to eight sum-of-product outputs to extend address decoding to external peripherals or to implement logic replacement on a board.

Memory in the PSD301 is provided by EPROM for program and table storage and SRAM for scratch pad storage and development and diagnostic testing. The EPROM density is 256K bits and the SRAM density is 16K bits. Both can be operated in either word-wide or byte-wide fashion, which translates to a 32K x 8 or 16K x 16 EPROM configuration and a 2K x 8 or 1K x 16 SRAM configuration. As described above, the EPROM is divided into 8 blocks (of 4K x 8 or 2K x 16), with each block typically on a 2K boundary locatable within a 1 Meg address space.

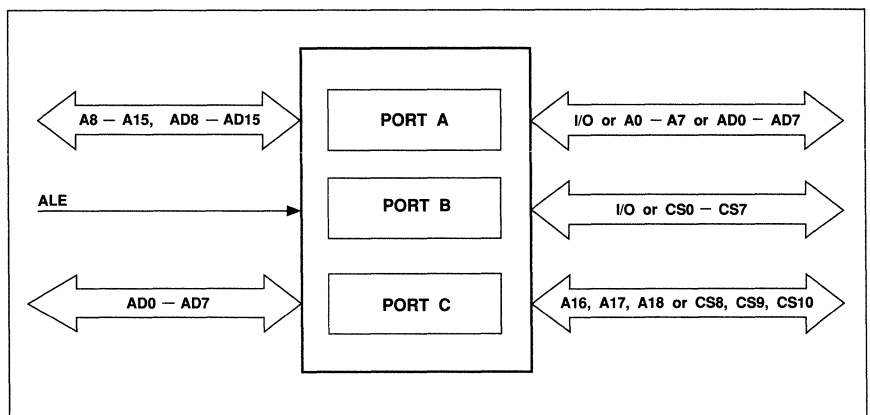
There are 3 ports on the PSD301 that are highly flexible and programmable: Ports A, B and C, illustrated in Figure 4. Port A is an

8-bit port that can be configured in a variety of ways. For example, if the PSD301 is in the multiplexed mode, port A can be configured pin-by-pin to be an I/O or a lower order latched address. Alternatively, port A can be configured in the track mode to transfer 8 bits of address and data inputs through port A. This enables the micro-controller to share external resources, such as additional SRAM, with other controllers. In either case, each port A output can be configured to be CMOS or open drain. If the PSD301 is in the non-multiplexed mode, port A becomes the lower order data for the chip.

Port B is another flexible 8-bit port. In the multiplexed mode or 8-bit non-multiplexed mode, each pin on port B can be customized to function as an I/O or a chip-select output. The chip-select signals are determined by the PAD programming and are used for general logic replacement or to extend the address decoding to external peripherals. Each pin in this mode can also be programmed to have a CMOS or an open drain output. In the 16-bit non-multiplexed mode, port B becomes the higher order data for the chip.

Port C is the third port which is available on the PSD301. It is a 3-bit port that can be programmed on a pin-by-pin basis to be chip-select outputs and/or general-purpose logic inputs or addresses to the PAD. Some uses for port C might be to extend the address range to 1 Meg, or to create finer address decoding resolution down to 256. Or, one might use port C to help create a simple state machine.

**Figure 4.
PSD301
Multiplexed
Address/Data
Configuration**



Simple Interfaces to the PSD301.

One of the overwhelming advantages of the PSD301 is its ability to interface to virtually any microcontroller without any glue logic, while providing additional I/O ports and memory. This is accomplished by configuring or programming the part to function in an operational mode geared for a specific application.

For instance, there are 45 configuration bits on the PSD301 that have to be programmed in addition to the EPROM prior to usage. These configuration bits are determined during development by the designer using the WSI MAPLE software package. After the configuration bits are determined, the EPROM code and configuration data can be merged during compilation and the part subsequently programmed.

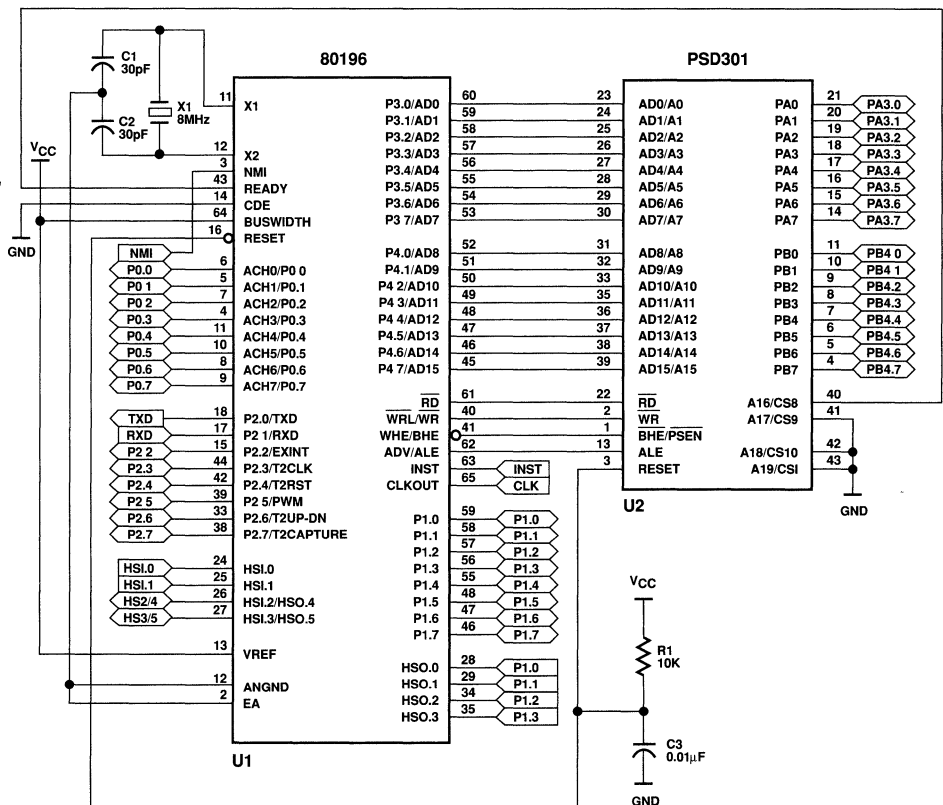
Interfacing the PSD301 to different microcontrollers is accommodated by the

configuration bits discussed above. To illustrate how this works, two examples are provided.

The first example is with the 80C196 microcontroller. This 16-bit microcontroller from Intel interfaces directly to the PSD301, providing it with additional off-chip program store EPROM and data store SRAM, as well as the flexibility that comes with three additional I/O ports. As illustrated in Figure 5, the 80C196's 16-bit multiplexed address/data bus and control signals (\overline{RD} , \overline{WR} , \overline{BHE} , \overline{ALE} , \overline{RESET}) connect directly to the PSD301. This is achieved with the PSD301 in the following configuration:

- 16-bit data bus
- Multiplexed address/data
- \overline{RD} and \overline{WR} mode set
- Active HIGH \overline{ALE}
- Active LOW \overline{RESET}
- A16–A18 configured as output
- Combined memory mode

Figure 5.
General Schematic Diagram of the 80C196 and PSD301.



Simple Interfaces to the PSD301 (Cont.)

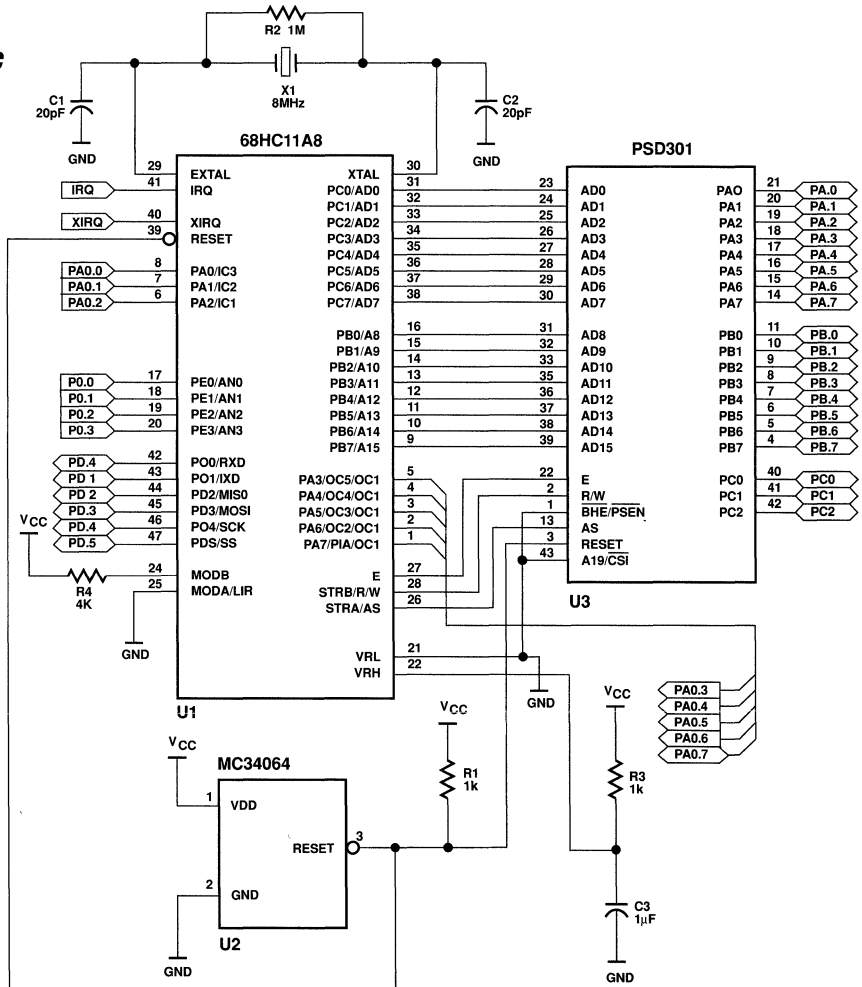
The other configuration options that are available, but not listed above, are application dependent and can be changed to meet the requirements of the design. For instance, on pin 43 (A19/ $\overline{\text{CSI}}$), the power-down option $\overline{\text{CSI}}$ could be selected if power consumption savings is important. If it isn't and another logic input to the PAD would be helpful, A19 could be selected. And, if open-drain drivers are important on one of the ports to drive a display, for example, they also could be selected instead of CMOS drivers.

All other microcontrollers have simple interfaces to the PSD301 as well. This includes all the variations of microcontrollers in the 8-bit 68HC11 family

from Motorola. For simplicity's sake, the PSD301 interface to 68HC11 versions with multiplexed address/data buses will be discussed, although the non-multiplexed versions will interface to the PSD301 in a similar manner, except in this case port A will become dedicated for 8-bit data.

Figure 6 illustrates the interconnections between the PSD301 and the 68HC11 microcontroller with multiplexed address/data buses. Again, all the address/data connections are direct, as well as the control signals (E, R/ $\overline{\text{W}}$, AS, and $\overline{\text{RESET}}$). Because $\overline{\text{BHE/PSEN}}$ is not used, this PSD301 input signal is tied HIGH.

Figure 6.
General Schematic Diagram of the 68HC11 and PSD301.



Simple Interfaces to the PSD301 (Cont.)

The PSD301 must be programmed using WSI's MAPLE software package in the following modes to achieve this configuration:

- 8-bit data bus
- Multiplexed address/data
- R/W and E mode set
- Active HIGH AS (ALE)
- Active LOW RESET
- Combined memory mode

Again, other parameters on the PSD301 can be set to fit additional design requirements. These include the security bit, the port I/Os, and the PAD inputs and outputs.

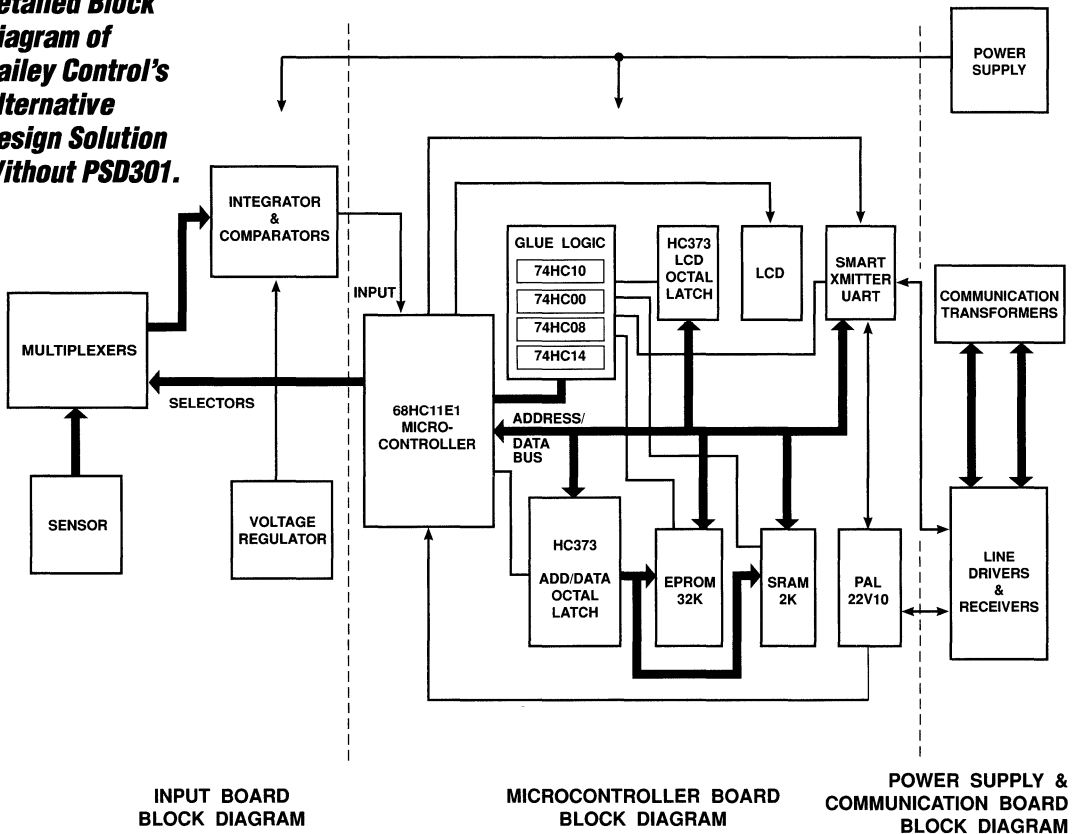
1

The "Smart" Transmitter Design.

The microcomputer-based smart transmitter design, by Bailey Controls, requires program store 256K bits EPROM for storing algorithms and data store 16K bits SRAM for storing A/D, communication and LCD routines. It also requires two octal latches, a PLD, and a variety of glue logic to interface to its

68HC11 microcontroller, UART, and LCD display. This is illustrated in Figure 7. Of course, with board space on the digital board being limited, another board would have been needed to accommodate these components, unless they in some way could be integrated.

Figure 7. Detailed Block Diagram of Bailey Control's Alternative Design Solution Without PSD301.



The "Smart" Transmitter Design (Cont.)

This is where the PSD301 provides exceptional value. As discussed, the PSD301 already integrates EPROM,¹ SRAM,² a PLD, and other glue logic all on one chip. It interfaces to the 68HC11 directly and actually integrates 8 chips from the alternative design into one, eliminating the need to add another board. The resultant architecture is illustrated in Figure 8.

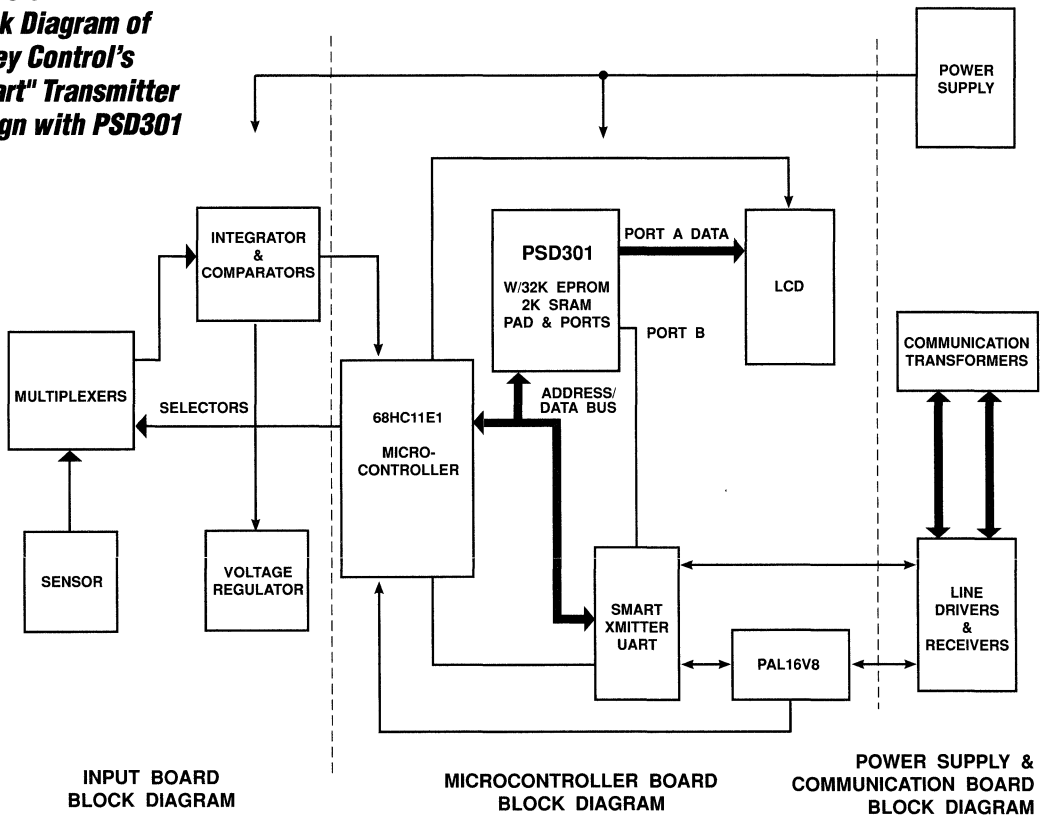
Note that in the alternative design shown in Figure 7, ports typically lost when connecting the microcontroller to external memory had to be recreated externally with latches and buffers when memory was connected to the microcontroller. With the PSD301, these ports are recreated internally, eliminating the latches and buffers.

For example, to interface the PSD301 to the 24-character LCD display, each pin of

port A is configured as an I/O and mapped to the byte-wide LCD data inputs. Then to write to or read from the LCD display, port A is accessed like a memory-mapped peripheral via an address offset from the base CSIOPORT defined in the PAD. Since port A is qualified by and handled through the PAD, there is no need for an external octal latch.

Other TTL logic is not required to interface to the 68HC11's control signals, memory, or peripherals either. It is all integrated in the PSD301. Thus, a smaller PLD than originally thought required in the design was used — a 16V8 instead of a 22V10 — because the PAD was able to reduce the amount of logic by creating chip selects for the UART and other logic functions.

Figure 8. Block Diagram of Bailey Control's "Smart" Transmitter Design with PSD301



PSD301 Bonuses

Besides considerably reducing board space in this smart transmitter design by reducing parts count, several other benefits of the PSD301 were also seen. These include reliability improvement, power consumption savings, inventory savings, faster time-to-market, and cost savings.

Reliability was improved because there are seven less chips required for implementation that could fail in the design. Also, by reducing chip count, 112 pins and about 100 traces were eliminated and the number of layers on the board were reduced from 8 to 4, making failures due to open or shorted pins and traces less likely to occur.

Power consumption was reduced because much faster discrete EPROM and SRAM devices with access times of ~75 ns would have been required in conjunction with glue logic for selecting different devices instead of using the PSD301, saving at least 20 mA Icc. (The access time for the PSD301 memories include decoding and input address latch delays). If the power-down feature on the PSD301 were also used, power savings could be increased further. For example, in a system which is accessing the PSD301 only a quarter of the time, the power consumption could be reduced by 75% to 8 mA typical.

As an added benefit, the PSD301 helped reduce inventory significantly by obsoleting multiple chips. And, if last minute changes in the design were required, the PSD301 would be able to accommodate them without additional hardware modifications. So, purchasing line item management is made simpler and easier.

With the reprogrammable PSD301, development time was kept to a minimum by easily accommodating design iterations in both hardware and software. Changes in I/O, address mapping, bus interface, and code were simple to make. Also, debugging was made easier with the PSD301's on-chip SRAM for downloading test programs. This all helped to shorten the design development cycle, reduce development costs, and speed up market introduction of the smart transmitter.

By using the PSD301, cost savings were realized by reducing system cost with fewer boards (or reduced board space), improving reliability, and reducing inventory levels. Savings were also attributable to lower manufacturing costs because there were fewer parts to program and place. And by getting to market faster, profits were improved significantly.

Summary

The PSD301 peripheral solved a fundamental problem often seen in that instead of getting "locked into" an inflexible multiple chip memory sub-system solution, the PSD301 was able to provide

much higher integration and flexibility all at the same time. Clearly, using the PSD301 was the better choice for the smart transmitter design.

Notes

1. If more EPROM was needed, the PSD302/312 w/512K bits EPROM and the PSD303/313 w/1024K bits EPROM are available in the same pinout and packages (please call your local WSI sales representative for availability). Or, multiple PSD301s can be cascaded together with the added benefit of increased functionality and I/O's.
2. If more SRAM is needed, it can be added externally without requiring any additional glue logic. See WSI Application Note 011. Note that many engineers have 8K x 8 SRAM in their systems now – not because they need it, but because 2K x 8 SRAMs are not as readily available.

Appendix 1.
PSD301
Configuration

Wsi PSD301 Configuration Save File for Smart Transmitter Design

ALIASES

CS0 = ASICCS

GLOBAL CONFIGURATION

Address/Data Mode: MX
Data Bus Size: 8
CSI/A19: CSI
Reset Polarity: LO
ALE Polarity: HI
WRD/RWE: RWE
A16-A19 Transparent or Latched by ALE: T
Using different READ strobes for SRAM and EPROM: N

PORT A CONFIGURATION (Address/IO)

Bit No.	Ai/IO.	CMOS/OD.
0	IO	CMOS
1	IO	CMOS
2	IO	CMOS
3	IO	CMOS
4	IO	CMOS
5	IO	CMOS
6	IO	CMOS
7	IO	CMOS

PORT B CONFIGURATION

Bit No.	CS/IO.	CMOS/OD.
0	CS0	CMOS
1	CS1	CMOS
2	CS2	CMOS
3	CS3	CMOS
4	CS4	CMOS
5	CS5	CMOS
6	CS6	CMOS
7	CS7	CMOS

CHIP SELECT EQUATIONS

/ASICCS = /A15 * A14 * /A13 * /A12 * E
/CS1 = /A15 * A14 * /A13 * A12 * E
/CS2 = /A15 * A14 * A13 * /A12 * E
/CS3 = /A15 * A14 * A13 * A12 * E
/CS4 = /A15 * /A14 * /A13 * /A12 * /A11 * E
+ /A15 * /A14 * /A13 * /A12 * /A11 * / R/W
/CS5 = /A15 * /A14 * /A13 * /A12 * A11 * E
+ /A15 * /A14 * /A13 * /A12 * A11 * / R/W
/CS6 = /A15 * /A14 * /A13 * A12 * /A11 * E
+ /A15 * /A14 * /A13 * A12 * /A11 * / R/W
/CS7 = /A15 * /A14 * /A13 * A12 * A11 * E
+ /A15 * /A14 * /A13 * A12 * A11 * / R/W



Appendix 1.
PSD301
Configuration.
(Cont.)

1

 PORT C CONFIGURATION

Bit No. CS/Ai.
 0 CS8
 1 CS9
 2 CS10

CHIP SELECT EQUATIONS

/CS8 = /A15 * /A14 * A13 * /A12 * /A11 * R/W
 /CS9 = /A15 * /A14 * A13 * /A12 * A11 * R/W
 /CS10 = /A15 * /A14 * A13 * A12 * /A11 * R/W

 ADDRESS MAP

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	EPROM	EPROM	File Name
	19	18	17	16	15	14	13	12	11	STRT	STOP	START	STOP	
ES0	N	N	N	N	1	0	0	0	N	8000	8FFF	8000	8fff	BCN2.0
ES1	N	N	N	N	1	0	0	1	N	9000	9FFF	9000	9fff	BCN2.0
ES2	N	N	N	N	1	0	1	0	N	A000	AFFF	a000	afff	BCN2.0
ES3	N	N	N	N	1	0	1	1	N	B000	BFFF	b000	bfff	BCN2.0
ES4	N	N	N	N	1	1	0	0	N	C000	CFFF	c000	cfff	BCN2.0
ES5	N	N	N	N	1	1	0	1	N	D000	DFFF	d000	dfff	BCN2.0
ES6	N	N	N	N	1	1	1	0	N	E000	FFFF	e000	efff	BCN2.0
ES7	N	N	N	N	1	1	1	1	N	F000	FFFF	f000	ffff	BCN2.0
RS0	N	N	N	N	0	1	1	0	0	6000	67FF			
CSP	N	N	N	N	0	0	1	1	0	3000	37FF			

***** END *****

CDATA = 0	CPAF1 [0] = 0
CADDRDAT = 1	CPAF1 [1] = 0
CRRWR = 1	CPAF1 [2] = 0
CA19/(/CSI) = 0	CPAF1 [3] = 0
CALE = 0	CPAF1 [4] = 0
CRESET = 0	CPAF1 [5] = 0
COMB/SEP = 0	CPAF1 [6] = 0
CADDHLT = 0	CPAF1 [7] = 0

CPAF2 = 0

CPACOD [0] = 0	CPBCOD [0] = 0
CPACOD [1] = 0	CPBCOD [1] = 0
CPACOD [2] = 0	CPBCOD [2] = 0
CPACOD [3] = 0	CPBCOD [3] = 0
CPACOD [4] = 0	CPBCOD [4] = 0
CPACOD [5] = 0	CPBCOD [5] = 0
CPACOD [6] = 0	CPBCOD [6] = 0
CPACOD [7] = 0	CPBCOD [7] = 0

CPBF [0] = 0	CPCF [0] = 1
CPBF [1] = 0	CPCF [1] = 1
CPBF [2] = 0	CPCF [2] = 1
CPBF [3] = 0	
CPBF [4] = 0	
CPBF [5] = 0	
CPBF [6] = 0	
CPBF [7] = 0	





Programmable Peripheral Application Note 014 Using the PSD3XX PAD for System Logic Replacement

By Jeff Miller

Introduction

In 1990, WSI introduced the Programmable System Device (PSD): the first device in the world integrating UVEPROM, SRAM and programmable logic on a single chip of silicon. The highly-successful PSD301 was the first device in the PSD family and is currently used in applications ranging from fluid analyzers to high performance computers. The PSD device, by combining most of the peripheral functionality required by a typical microcontroller unit into one package, has enabled designers to greatly reduce part count, power and board space which has translated into significant cost savings.

Even if the PSD3XX family were simply a collection of EPROM and SRAM with an

on-chip decoder, it would be capable of adding significant value to the system into which it were designed. However, the PSD3XX family is much more than just a combination of memory devices. The on-chip PLD may be used for many useful purposes in addition to providing the address decode capability. The purpose of this note is to demonstrate, in detail, the full capability of the PAD section of the PSD3XX family. A basic, though not extensive, knowledge of the PSD 3XX family and the Maple programming software is assumed by this note. Please consult Application Note 011 and/or the appropriate PSD3XX family data sheet for this general knowledge.

1

PAD Architecture

The Programmable Array Decoder (PAD) contained in the PSD3XX family is a standard PLD array designed to provide all of the internal memory and I/O device chip selects as well as an external logic replacement capability. It has 14 inputs, 24 outputs and 40 product terms with which to perform these functions. See Figure 1 for an illustration of the PAD.

The PAD's 14 inputs are as follows:

- A11 – A19
- ALE or AS
- \overline{RD} or E
- \overline{WR} or R/ \overline{W}

The A11 – A19 pins are labeled as address inputs, however, they do not have to be. A11 - A15 are generally sourced by the microcontroller or microprocessor that is connected to the PSD device. If the controller generates more than 16 bits of address, the A16 – A19 inputs may be used to connect the high order address bits for a full 1 MByte of address space. If the controller does not require this much address space, A16 – A19 may be used for other purposes, like general I/O or logic inputs.

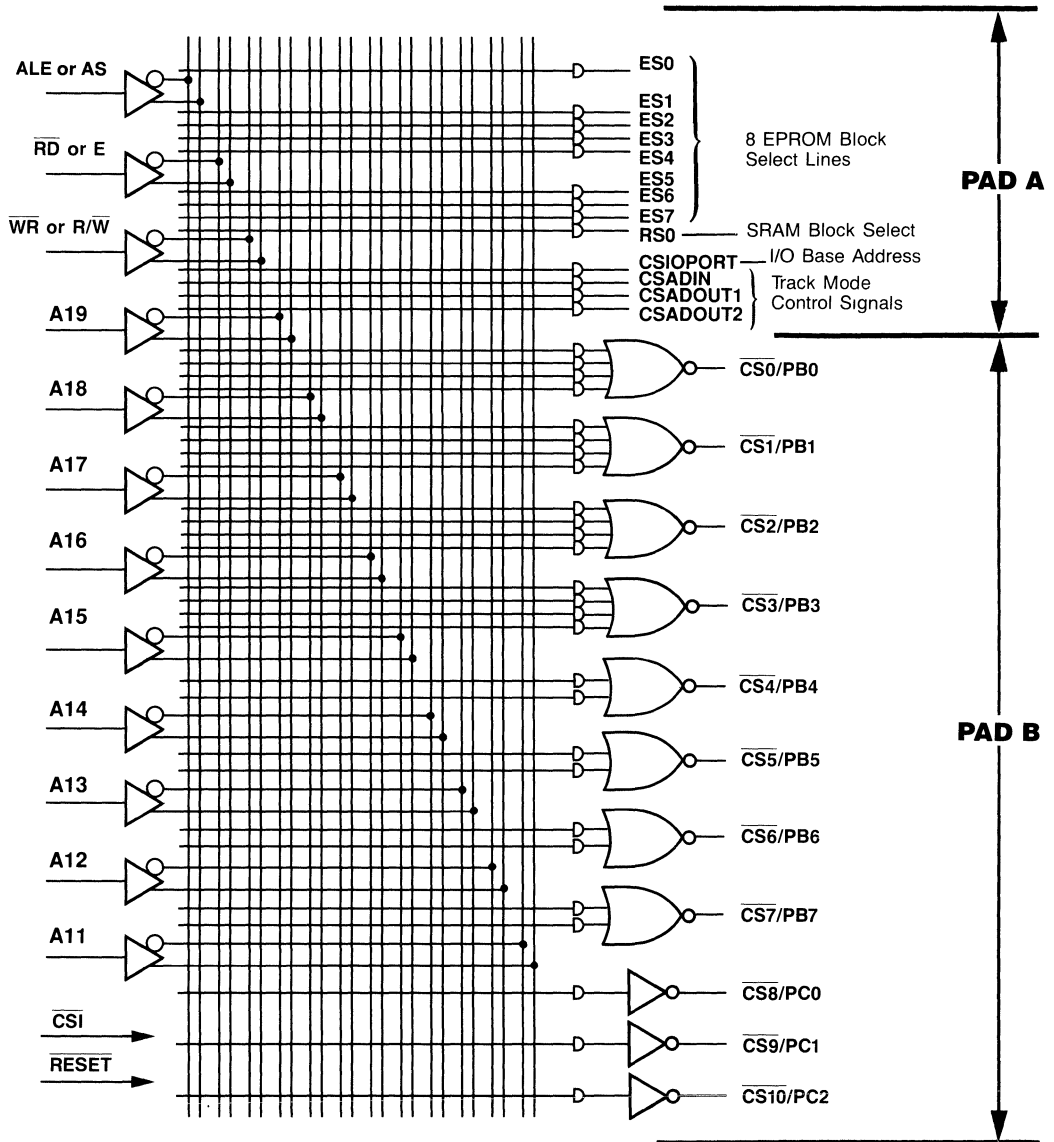
A19 is multiplexed with the $\overline{CS1}$ signal, which is used to place the PSD device in a

low power mode when the system requires it. When configured as $\overline{CS1}$, the A19 pin may not be used for any other purpose except the power down mode. In this mode, the $\overline{CS1}$ signal is used by the PAD only to disable it, causing it to expend less power. When configured as A19, this signal may be used as a general purpose input to the PAD from the external system. This capability will be described in more detail later in this note. A16 – A18, when not necessary for address expansion, may also be used as general purpose inputs to the PAD. Thus, a total of four of the 14 PAD inputs may be general purpose, allowing the replacement of external logic by the PSD device. These inputs may be combined with the other PAD inputs to form complex equations involving addresses, strobes and external signals.

When attempting to visualize the full capability of the PAD outputs, it is most clear when it is broken into two sections, labeled in Figure 1 as PAD A and PAD B. PAD A is responsible for providing all of the internal chip selects for the EPROM, SRAM and I/O ports and the track mode control signals, and PAD B is responsible for the external logic replacement function.



Figure 1.
PAD
Architecture



PAD A

Thirteen of the 24 PAD outputs and thirteen of the 40 product terms are dedicated to PAD A. PAD A should be considered the internal address decoder, used to select the various on-chip memories and I/O devices according to the memory map programmed by the user. Each output has a single product term, allowing a particular

resource to be allocated a single contiguous range of addresses which will be used to access it. All of the PAD inputs are available for generation of the PAD A outputs, allowing the designer to select internal resources using any combination of address, strobe and external signals.

**PAD A
(Cont.)**

The PAD A outputs are as follows:

- ES0 – ES7
- RS0
- CSIOPORT
- CSADIN
- CSADOUT1
- CSADOUT2

ES0 – ES7 are used to select the internal EPROM resources. Using the PSD301 as an example, there are eight select lines with which to access 32 KBytes of EPROM. Thus, each select line can enable a block of 4 KBytes of EPROM configured as 4K x 8 or 2K x 16. Each block must be contiguous, but the blocks may be placed anywhere within the address space of the microcontroller.

RS0 is used to select the SRAM resource. This single signal accesses a single 2 KByte block of SRAM which may be configured as 2K x 8 or 1K x 16. Again, this block must be contiguous but may be placed anywhere in the address map.

CSIOPORT is the signal which defines the base address of the on-chip I/O ports and control registers. The I/O ports and control registers occupy a 2K block of addresses which, like the memories, must be contiguous but may be located anywhere in the address space of the microcontroller. Once configured in the address map, CSIOPORT defines the base address of these ports and registers. An offset is added to the base address to individually access the registers. Table 1 below lists the offset values for these registers.

CSADIN, CSADOUT1 and CSADOUT2 are used to control the Track Mode operation. The Track Mode is an available option for Port A to allow it to “track” the Address/Data bus inputs to the PSD device from the microcontroller. This provides the capability to connect the PSD device, and therefore the microcontroller, to one or more shared resources. These resources may be memory or other devices which must be accessed by more than one microprocessor or microcontroller.

CSADIN is generated when the microcontroller is attempting to read data from Port A in the track mode. It is generated from one product term involving the address inputs and the \overline{RD} strobe (Intel mode) or R/W and E (Motorola mode). This allows the user to configure the address range in which the data is to be read from Port A. CSADOUT1 is generated when the microprocessor is accessing a “tracked” address. It is generated from a single product term involving the address inputs and ALE. When the address generated by the microcontroller is within the block specified by the user for track mode, and the ALE is active, CSADOUT1 becomes active, transferring the address and outputting it from Port A. CSADOUT2 is generated when the microcontroller is performing a write operation to a tracked address. It also has one product term involving the address inputs and \overline{WR} (Intel mode) or R/W and E (Motorola mode). When the microcontroller performs a write to the appropriate address, CSADOUT2 is generated, transferring the data and outputting it from Port A. For further details on the operation of the Track Mode, please consult Application Note 017.

**Table 1.
I/O Port
Offset
Addresses**

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

**Example:
Address
Mapping
With PAD A**

In this example, we will choose a sample address map which is similar to those used in typical microcontroller applications. This example assumes the use of a PSD301 device with 256 Kbits of EPROM and 16 Kbits of SRAM. Figure 2 below illustrates our sample address map.

In this example, we have located the boot code and interrupt service routines beginning at address 0000 in EPROM block 0. The SRAM is located in the 2K block beginning at address 0x1000 and can be used for the stack and/or other scratchpad data. The I/O ports occupy the 2K block beginning at address 0x1800. Addresses in this range will access ports A and B and their control registers. The area from 0x2000 to 0x8FFF is unused in this example, though it could be used for external resources as will be shown later. Finally, the main program resides in the 28K block of EPROM located from address 0x9000 to 0xFFFF and is selected by ES1 – ES7.

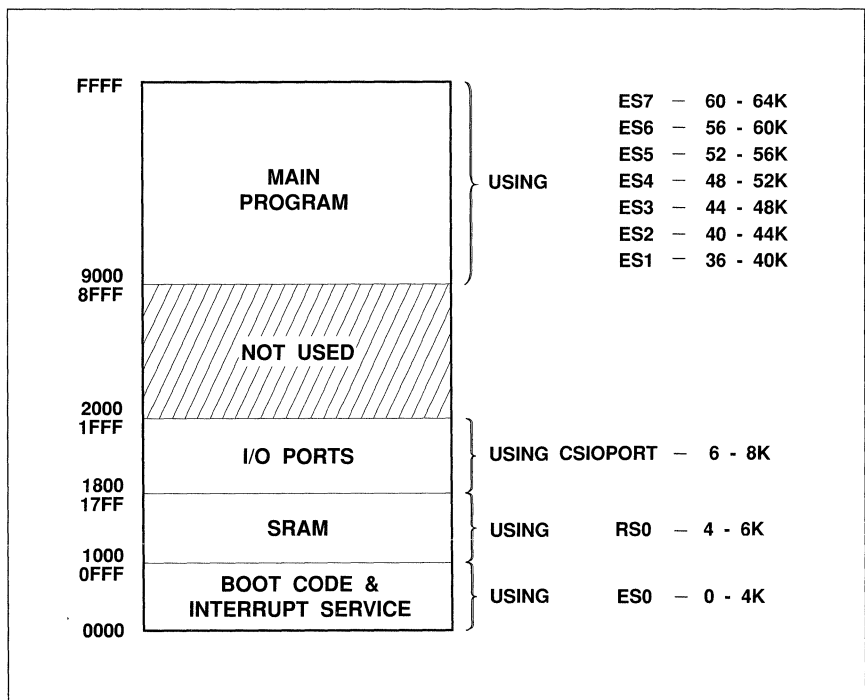
Configuring this memory map would normally require designing a decoder to generate the appropriate chip selects for each given address range. For example, assuming that a microcontroller with a 16-bit address bus is used, the chip select for EPROM bank 0 (ES0) would be generated with the following equation:

$$ES0 = /A12 \cdot /A13 \cdot /A14 \cdot /A15$$

Equations like this one would be formulated for each of the chip selects, and the entire function would probably be placed in some kind of programmable device. When the PSD device is used, PAD A replaces this programmable device. Programming PAD A to perform this function is a simple task using WSI's Maple software.

Entering the ADDRESS MAP menu in the Maple software running on a PC compatible computer, the user will see a screen similar to the one shown in Figure 3.

**Figure 2.
Example
Memory Map**



**Example:
Address
Mapping With
Pad A (Cont.)**

Upon displaying this screen, the Maple software is ready for the user to enter the memory map data. This is performed quite simply by moving the cursor to the appropriate point with the arrow keys, and then entering the appropriate data. The address mapping may be entered in either of two ways. First, the user may select each address bit individually for each chip select and enter a 0 or 1 as appropriate for the equation desired. In our example, for ES0 we would enter a 0 in the columns for A12, A13, A14 and A15. The other bits are don't cares. In the other method of programming the pad, the user simply moves the cursor to the SEGMENT START column and enters the desired starting address for the block. Again, using our sample memory map, the user would move to the SEGMENT START column for ES0 and enter 0000. Maple

then automatically programs the 0's and 1's into the address bits correctly to program a 4K block of EPROM beginning at address 0x0000. Note that all EPROM blocks must begin on 4K boundaries. Figure 3 shows the resulting address map table for our example.

The address inputs which were unused in this example (A16, A17, A18 and A19) could have been used as general purpose inputs to the PAD for specialized control of the on-chip memory and I/O resources. When this is done, the designer has complete flexibility as to the configuration of the PSD device resources and may easily absorb many system functions into the PSD device. More detail about the use of A16 – A19 will be provided later in this note.

1

PAD B

Eleven of the PAD outputs and 27 of the product terms are dedicated to PAD B. Where PAD A was used to control the on-chip PSD device resources, PAD B controls any off-chip resources required by the system. As with PAD A, all inputs to the PAD are available to PAD B, allowing the system designer to formulate outputs involving any combination of address, strobes and external signals. Unlike PAD A, several of the outputs of PAD B have up to four product terms each.

The outputs of PAD B are as follows:

- CS0 – 7 (Port B)
- CS8 – 10 (Port C)

The outputs from PAD B are brought to the outside world through Port B and Port C. These outputs are called chip selects, though they may be used for any function whatsoever. The port pins are configured as selected by the user when the device is programmed with the Maple output file. There are many configuration options for each port pin.

**Figure 3.
Maple Address
Map Entry**

ADDRESS MAP

	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMENT START	SEGMENT STOP	FILE START	FILE STOP	FILE NAME
ES0	X	X	X	X	0	0	0	0	N	0000	0FFF			
ES1	X	X	X	X	0	0	0	1	N	9000	9FFF			
ES2	X	X	X	X	1	0	1	0	N	A000	AFFF			
ES3	X	X	X	X	1	0	1	1	N	B000	BFFF			
ES4	X	X	X	X	1	1	0	0	N	C000	CFFF			
ES5	X	X	X	X	1	1	0	1	N	D000	DFFF			
ES6	X	X	X	X	1	1	1	0	N	E000	EFFF			
ES7	X	X	X	X	1	1	1	1	N	F000	FFFF			
RS0	X	X	X	X	0	0	0	1	0	1000	17FF			
CSP	X	X	X	X	0	0	0	1	1	1800	1FFF			

ALIASES:

Fill in A19 – A11 (Binary) or SEGMENT START (Hex): and FILE (START, STOP) and FILE NAME, Use SPACEBAR to erase any field value.

F1 – Return to Main Menu F2 – Temporary Exit to DOS F3 – Go to Help

Cursor – UP: ↑ Down: ↓ Left Col: ← Right Col: → Right – F4 Left – F5

**PAD B
(Cont.)**

If you require more information about port configuration, please consult application note 011. If the port outputs are configured as chip selects (outputs from the PAD), they may not be used for any other purpose. For example, the three Port C signals may be configured as chip selects (outputs) or addresses (inputs) but cannot be both. Fortunately, the flexibility of the PSD device and the Maple software allows the designer to configure each Port B and C pin individually, so that the number of outputs and inputs may be optimized for a particular design requirement. See Table 2 below for an example of this flexibility.

This sample port configuration demonstrates all of the possible uses of a particular port pin. Though only Ports B and C may be inputs or outputs to/from the PAD, Port A is included in the table for completeness. In this example, five of the port pins are configured as PAD outputs (CS) and two are configured as PAD inputs (A). The remaining port pins in this example are configured as either I/O or address outputs. Several of the CS outputs have been configured as open drain. This allows them to be connected together in a wired OR configuration to increase the number of product terms even further if desired.

**Table 2.
Sample Port
Configuration**

<i>Pin</i>	<i>Configuration</i>	<i>CMOS/OD</i>
PA0	Address Out	CMOS
PA1	Address Out	CMOS
PA2	Address Out	CMOS
PA3	Address Out	CMOS
PA4	I/O	CMOS
PA5	I/O	OD
PA6	I/O	OD
PA7	I/O	CMOS
PB0	$\overline{CS0}$	CMOS
PB1	$\overline{CS1}$	CMOS
PB2	$\overline{CS2}$	OD
PB3	$\overline{CS3}$	OD
PB4	I/O	CMOS
PB5	I/O	CMOS
PB6	I/O	CMOS
PB7	I/O	CMOS
PC0	A16	—
PC1	A17	—
PC2	$\overline{CS10}$	OD

**Example:
Generating a
Logic Equation
With PAD B**

Assume that it is necessary to generate the following equation given the port configuration in Table 2 above. This equation is a simple OR of three product terms.

$$CS0 = A15 \cdot A14 \cdot \overline{A13} \cdot \overline{A17} \cdot RD \\ + \overline{A15} \cdot A14 \cdot A12 \cdot WR + A16$$

Figure 4 illustrates the Maple programming sequence to generate this equation.

To program this equation, the PORT B menu is entered from the Maple software. CS0 is selected by moving the cursor to it using the arrow keys. With CS0 selected, the user then presses the F3 key to bring up the CHIP SELECT DEFINITION table for CS0. The table contains four rows for

data entry, each one corresponding to one of the available product terms for CS0. Implementing this equation required using three of the four available product terms. The fourth is left blank and will not be used to generate the output.

To enter the equation into the table, simply move the cursor around into the appropriate position and enter a 1 if the corresponding signal should be high for the equation to be true, 0 if it should be low, and X or SPACE if the signal is a don't care. The first term of the equation requires a low on A17, a high on A15, a high on A14, a low on A13 and a high on RD for the term to become active. Thus, 1's are placed in the A15, A14 and RD positions,

**Example:
Generating a
Logic Equation
With PAD B
(Cont.)**

and 0's are placed in the A17 and A13 positions. The remaining terms in the equation are entered in the same way. Note that A17 and A16 in this example (as well as A19 and A18) need not be address bits, but may instead be used to bring external signals into the PAD.

Four product terms are available on each of the CS0 – CS3 outputs, two terms are available on the CS4 – CS7 outputs and one term is available on CS8 – CS10. When planning the use of the PAD outputs, it is important to consider this so that the most efficient use of the product terms can be achieved.

**Figure 4.
Programming
PAD Outputs**

PORT B		CHIP SELECT DEFINITION CS0												
PIN	CS/I/O	CMOS/OD	A19	A18	A17	A16	A15	A14	A13	A12	A11	ALE	RD	WR
PB0	CS0	CMOS	X	X	0	X	1	1	0	X	X	X	1	X
PB1	CS1	CMOS	X	X	X	X	0	1	X	1	X	X	X	1
PB2	CS2	CMOS	X	X	X	1	X	X	X	X	X	X	X	X
PB3	CS3	CMOS												
PB4	CS4	CMOS												
PB5	CS5	CMOS												
PB6	CS6	CMOS												
PB7	CS7	CMOS												

ALIASES:

CS definition is the NOR of the product terms (rows). Enter 1 to select Active High signal, 0 to select Active Low signal, X to mean "don't care", SPACEBAR to erase. Enter values in columns relevant to your application; other blank columns will be treated as "don't cares".

F1 – Return to PORT B

Cursor – Up: ↑ Down: ↓ Left: ← Right: →

**Application
Examples**

The following section will illustrate the use of the PAD for system logic replacement in some common microcontroller applications.

Basic Chip Select Generation

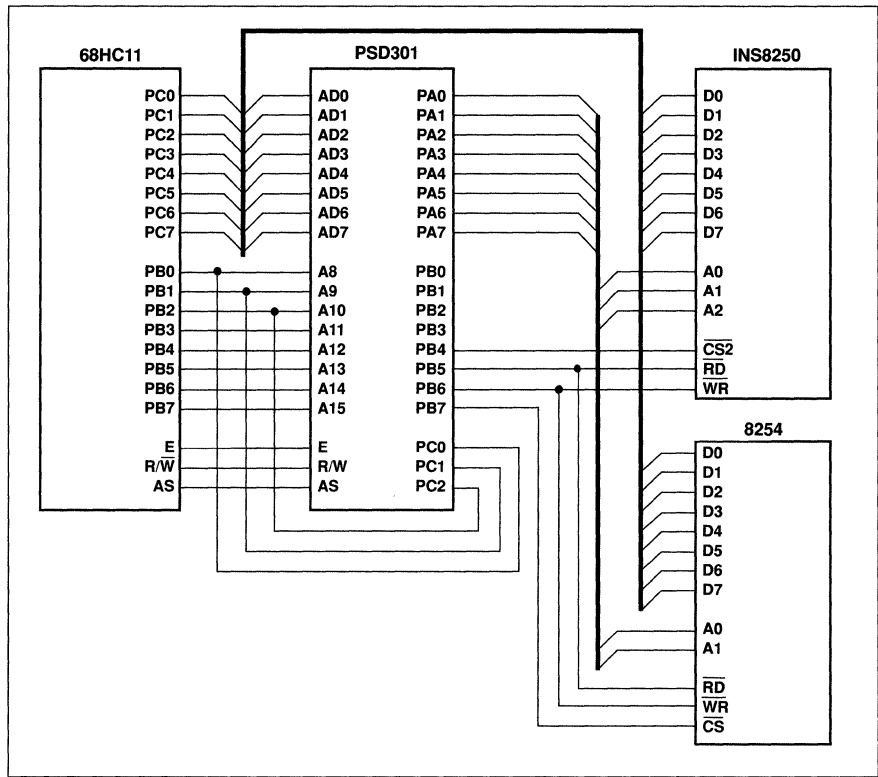
One of the simplest uses of PAD B is the generation of chip selects for off-chip resources such as I/O devices or memories. Figure 5 below depicts the connection between a 68HC11 microcontroller, the PSD301 and two common peripheral devices: the 8250 UART and the 8254 counter/timer.

The 68HC11 is an 8-bit microcontroller with a 16-bit address bus. The lower 8 bits of address are multiplexed with the data bus while the upper 8 bits are transmitted on their own bus. An address strobe (AS) is provided to latch the address off of the multiplexed bus. A R/W signal indicates whether the current bus transaction is a read or a write (R/W = 1 = read, R/W = 0 =

write). The E signal is the clock used to strobe the data in or out of the microcontroller. The PSD301 can be configured to exactly match this signal definition and then connected as shown in the diagram. Not all of the 68HC11 or PSD301 signals are shown, only those relevant to this example of PAD capability.

The 8250 is a UART device commonly used in microcontroller systems to provide a serial data communication port. It has a simple bus interface, yet does not directly connect with the 68HC11 bus architecture. It requires an 8-bit bus to transfer data to and from the microcontroller and a separate 3-bit address bus used to access its internal registers. It also requires a chip select and separate read and write strobes (RD and WR). The chip select is generated by decoding the address from the microcontroller. The RD and WR signals may be generated from the R/W and E signals

**Figure 5.
A Typical
Microcontroller
System**



**Application
Examples
(Cont.)**

according to the following equations:

$$/RD = /(R/\bar{W} \cdot E)$$

$$/WR = /(R/\bar{W} \cdot E)$$

These equations may be easily generated using PAD B and sent out through two of the chip select outputs. We have chosen CS5 and CS6, which come out on PB5 and PB6, for this example.

In order to provide the address lines to the 8250, we have configured Port A to output the latched address. This eliminates the need for any external latches to demultiplex the address/data bus from the microcontroller. Though all eight of the Port A pins have been configured as address outputs in this example, it is possible to configure only those address bits required for the application, A0 – A2 in this example, and configure the remaining Port A pins as general I/O.

The 8254 is a programmable interval timer

which, like the 8250, is a peripheral used in many microcontroller applications. Its bus connection is very similar to the 8250, allowing it to use the same read and write strobes (\bar{RD} and \bar{WR}) and address lines. It also requires a chip select which is decoded from the microcontroller address.

The chip selects for both of the peripheral devices may be easily decoded from the address inputs to PAD B. Normally, the addresses which are inputs to the PAD (A11 – A19) would give decoding resolution down to 2K. This means that each of the two peripheral devices that require chip selects would be allocated an address range of at least 2K. Since these devices do not require this much space and the 68HC11 has only a 16-bit address bus, it is possible to use the high order address inputs of the PSD device to improve the decoding resolution. To achieve this goal, we have configured Port C as address inputs A16 – A18, but have connected them to A8 – A10 from the microcontroller. This means that the PAD will now have

Application Examples (Cont.)

access to A8 – A15 for decoding, thus providing a resolution of 256 instead of 2K. This could actually be further reduced to a resolution of 128 if we were to configure the A19/CSI input to be A19, and then connect it to A7 from the microcontroller. In this example, we have not done this so that CSI is still available to place the PSD301 into low power mode if required.

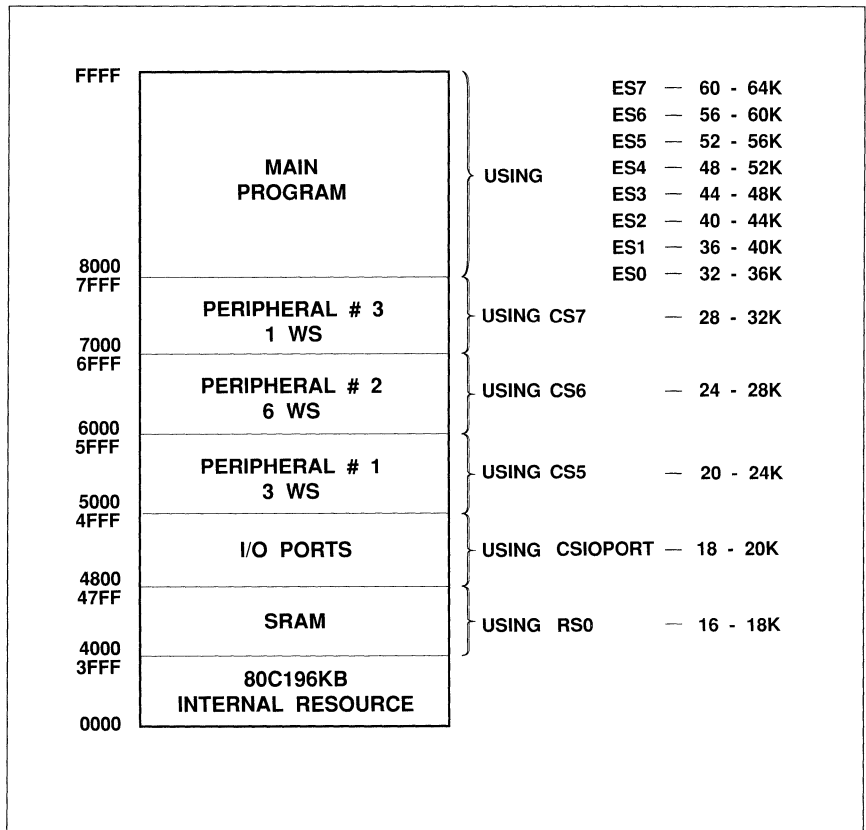
We now have to define the addresses of each of the peripherals so that the chip select equations may be defined. We will start from the memory map provided earlier in Figure 2. This map allocated all of the internal resources of the PSD device. The external peripherals may be easily added to the unused area between addresses 0x2000 and 0x8FFF. Figure 7 depicts the new map with the external devices added. Notice that the internal resources can keep

their original address mapping even though the additional address inputs (A8 – A10) have been added. This is because these inputs may be don't cares in the decoding for the internal resources even when they are being used for the external resources.

Now, to wrap up this simple design, we must enter the configuration and mapping information into Maple. The configuration of the PSD device must be consistent with the operation of the 68HC11 microcontroller. The address/data mode must be multiplexed, the data bus must be 8 bits wide, CSI/A19 may be configured either way, the reset polarity should be active low, the ALE polarity is active high, the read and write lines must be R/W and E, A19 – A16 should be latched so that these bits become available just like the rest of the address bus, and the read strobes for the

1

Figure 6. Memory Map With Peripherals



Application Examples (Cont.)

SRAM and EPROM will be the same. This configuration should be entered from the configuration menu of the Maple software.

The address map programming for this example will remain the same as the one used earlier in Figure 3. The only items remaining are the programming of the ports and the generation of the equations for the chip selects and read/write strobes. First we must configure Port A to provide the latched address to the peripherals. This is accomplished by entering the PORT A menu in the Maple software. Maple will then ask you if you would like Port A configured for address I/O or the Track Mode. For this example, we will use the address/I/O configuration. Next, Port A must be configured pin for pin as an address output. This is easily performed by using the cursor keys to select the appropriate pin and pressing the SPACE BAR to change the configuration. It is also possible to configure each pin as an open drain or CMOS output, but for address outputs, it is better to make them CMOS.

Now, PORT C must be configured to provide the three additional address inputs. This is performed by entering the PORT C menu in Maple and selecting the appropriate pin with the cursor. Each pin should be configured as an address bit (Ai). Maple will call the pins A16 – A18 even though we will be using them as A8 – A10.

Lastly, we must configure the Port B outputs to become the chip selects and read/write strobes. First, the PORT B menu must be entered. Now, we must configure each pin as an I/O or CS output. PB0 – PB3 may be configured as general purpose I/O pins. PB4 – PB7 must be configured as chip selects. Once configured as chip selects, the equations for each output may be entered by following the Maple instructions. The procedure is the same as the one used in the earlier chip select example. Our equations, including the ones developed earlier for the read and write strobes, are defined for each output as follows:

$$PB5 = /CS5 = /RD = /(R/W \cdot \bar{E})$$

$$PB6 = /CS6 = /WR = /(R/W \cdot \bar{E})$$

$$PB4 = /CS4 = /8250CS = /(A15 \cdot A14 \cdot A13 \cdot A12 \cdot A11 \cdot A18 \cdot A17 \cdot A16)$$

$$PB7 = /CS7 = /8254CS = /(A15 \cdot A14 \cdot A13 \cdot A12 \cdot A11 \cdot A18 \cdot A17 \cdot A16)$$

This completes the design integrating these four components with no additional logic whatsoever. There is also additional space in the PAD for more functions if necessary, so we have not yet reached the limit of the integration possibilities with the PSD301.

Wait State Generation

Often, when using some of the newer high-performance microcontrollers with slower external peripherals, it is not possible to complete a read or write cycle to the peripheral in the time allowed by the microcontroller's minimum bus cycle. In this case, one or more wait states must be added to slow the controller down to the speed of the peripheral. One way of doing this is to fix a number of wait states for all bus cycles to allow the slowest device enough time for its access. Some controllers even provide the capability to do this internally through the programming of a register. This works, of course, but can severely impact the performance of the system. There is no need to penalize the performance of the entire system, which can include zero wait state memory devices and other peripherals, simply because one or more of the external

devices requires some number of wait states. It is possible, with minimal logic, to create a completely programmable automatic wait state generator using the PSD301 which will allow the fast resources to operate at zero wait states and still provide from one to eight wait states for the slower resources.

For this example, we will use an Intel 80C196KB microcontroller running at 12 MHz. This controller has the capability to operate in a 16-bit data mode, providing the opportunity to further increase performance if the system can also operate in this mode. The PSD301 does have the capability of operating in the 16-bit mode, making it a good match for the 80C196. We will assume that the 80C196 must be interfaced to several slow 8-bit peripherals requiring from one to eight wait states. With

Wait State Generation (Cont.)

the PSD301, we can provide the correct number of wait states for each peripheral with the added capability of dynamically sizing the bus to the appropriate width for the current access.

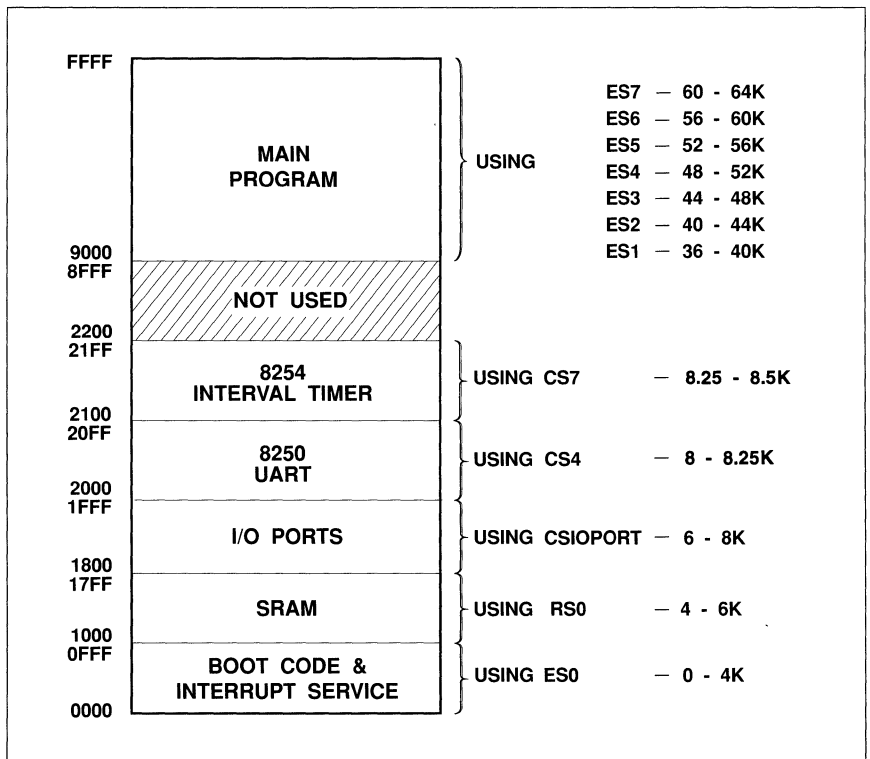
The memory map we will use for this design is depicted in Figure 6. The internal resources of some 80C196 derivatives occupy most of the address space from 0x0000 to 0x3FFF, though some have less resources. Therefore, we have constructed the memory map to place the PSD device resources above address 0x4000. The PSD301 SRAM and I/O devices occupy from address 0x4000 to 0x4FFF. This leaves the area from 0x5000 to 0x7FFF for external peripherals while leaving 0x8000 to 0xFFFF for the EPROM banks. We assume that we must connect three external peripherals to the PSD device using this address space, one requiring one wait state, one requiring three and one requiring six. This memory map is entered into the part similarly to the previous examples.

To achieve the variable number of wait states, the ideal solution is to decode the address to determine the number of wait states required for a particular address range, and then to use a counter to count the appropriate number. By using the PAD to initialize an external counter, a variable wait state counter can be created in this manner. This wait state generator requires only one external device, a 74FCT191 counter. The circuit used to implement this function is illustrated in Figure 8. The 80C196KB is directly connected to the PSD device which in turn provides the three chip select signals for the external peripherals (PER1CS, PER2CS and PER3CS) as well as the wait state generator function and the dynamic bus sizing. Ports B and C are fully utilized to provide the logic inputs and outputs required to implement these functions, while Port A is still available for general I/O or address use.

This circuit uses PAD B to decode the addresses driven by the microcontroller

1

**Figure 7.
Memory Map**



Wait State Generation (Cont.)

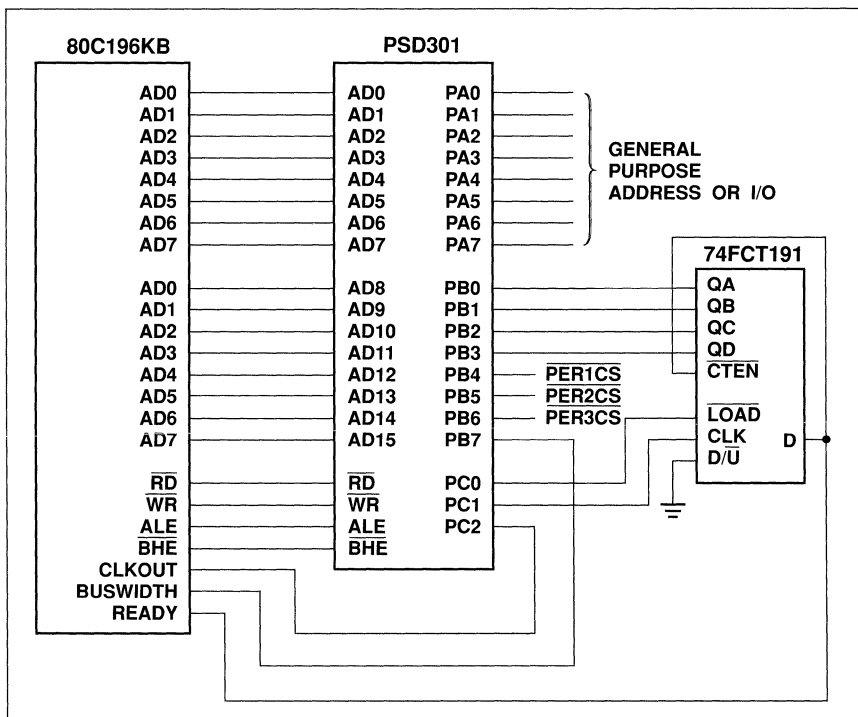
and provide four outputs, based on these addresses, which are used to initialize the 74FCT191 counter with its initial value. The counter is initialized using ALE to latch these four PAD outputs. The load signal for the counter is active low, however, while ALE is active high, so ALE is inverted using PAD B and sent out through Port C. Though the 80C196KB can be configured to provide an active-low address strobe, \overline{ADV} , the timing of the signal is inappropriate for use as the \overline{LOAD} input to the counter. Once the counter is initialized, it counts up from the initial value until the most significant bit increments from 0 to 1. The output of the most significant counter bit is routed to the READY input of the microcontroller. Thus, the controller will be held in wait states until the most significant counter bit is incremented. This output is also routed to the \overline{CTEN} signal of the counter so that counting will cease once the READY signal has been issued to the controller. The clock for the counter is an inverted version of the CLKOUT signal from the controller. This clock must be inverted since the 80C196KB uses the falling edge of the clock to sample the

READY input. PAD B again provides the inversion function by routing CLKOUT into one of the Port C pins, inverting it and routing it back out through another Port C pin.

The counter provides from zero to eight wait states depending on the initialized value. For zero wait states, the most significant counter bit is initialized to a "1", which provides the READY signal to the controller immediately and disables the counter from incrementing. If one wait state is desired, the counter is loaded with the value 7 (0111 binary) so that after it increments once, the most significant bit switches to a "1" and provides the READY to the controller. When two wait states are required, a 6 (0110 binary) is loaded into the counter, and so on for the rest of the wait state values.

To properly size the bus to the appropriate width, PAD B is again used to decode the addresses of the 8-bit devices. When the address of an 8-bit device is encountered, the BUSWIDTH signal is driven to eight

Figure 8. Wait State Generation Circuit



Wait State Generation (Cont.)

bits. For all other addresses, the width is set for 16 bits. The BUSWIDTH signal is output from one of the Port B pins.

The PSD device must now be configured to provide the functions required by the example circuit. The configuration of the PSD must first be programmed to function with the 80C196KB. This is easily performed by the Maple software as in the previous example. The address/data mode should be multiplexed, the data bus width should be 16 bits, CSI/A19 may be configured as required for the application, the reset polarity should be active low, the ALE polarity should be active high, separate \overline{RD} and \overline{WR} strobes should be used and A19 – A16 should be transparent, not latched, since they are used as logic inputs to the PAD.

Next, we must program the functionality of Port C. For this example, PC0 and PC1 are used as outputs from the PAD to provide the \overline{LOAD} and CLK signals for the '191 counter. This is performed by entering the PORT C menu in Maple and configuring PC0 and PC1 as CS8 and CS9, respectively. PC2 is used to input the CLKOUT signal from the microcontroller to the PAD so that it may be inverted. Therefore, it must be configured as address input A18. Now, the equations used to generate the PC0 and PC1 outputs must be entered into the PAD. PC0 is the \overline{LOAD} signal which is just the ALE input inverted. PC1 is an inverted version of A18, which contains the

CLKOUT signal. These equations are listed below:

$$PC0 = \overline{LOAD} = \overline{ALE}$$

$$PC1 = \overline{CLKOUT} = \overline{A18}$$

The equations are programmed by entering the CHIP SELECT DEFINITION menu for each of the two chip selects, as in the previous example, and entering the appropriate 1's, 0's and DON'T CARES. In the case of PC0, there are don't cares in all of the PAD inputs except ALE, where there is a 0. Similarly, for PC1, the A18 input is a 0 while the rest of the PAD inputs are don't cares.

Port A is usually configured next, and in this example it is free to be configured in any mode necessary for the application. It may become either I/O or address outputs, or may be set in the Track Mode as described earlier.

We are now ready to configure Port B. This example requires that all of the Port B pins be used as chip selects (logic outputs) from PAD B. PB0 – PB3 are used to initialize the counter with the correct number of wait states for each device. These outputs are defined according to the address ranges for each of the peripherals and the number of wait states required for each. Table 3 summarizes the outputs required for each peripheral so that we may define the correct equations for the outputs.

Table 3.
Wait State
Summary

Peripheral No.	Address Range	No. Wait States	PB0–PB3
1	0x5000-5FFF	3	1010
2	0x6000-6FFF	6	0100
3	0x7000-7FFF	1	1110

Wait State Generation (Cont.)

This table can be easily used to form the necessary equations for PB0 – PB3. PB3 can be considered the enable for the wait state generator which is active low only in the address ranges of the three peripherals. It must remain high for all other address ranges. The other three outputs simply encode the proper number of wait states. The resulting equations are listed below:

$$PB0 = /QA = /(A15 \cdot A14 \cdot A13 \cdot /A12)$$

$$PB1 = /QB = /(A15 \cdot A14 \cdot /A13 \cdot A12)$$

$$PB2 = /QC = /(A15 \cdot A14 \cdot A13 \cdot /A12)$$

$$PB3 = /QD = /(A15 \cdot A14 \cdot /A13 \cdot A12 + /A15 \cdot A14 \cdot A13 \cdot /A12 + /A15 \cdot A14 \cdot A13 \cdot A12)$$

PB4 – PB6 are used as chip selects for each of the three peripherals and are simply decoded from the address inputs by PAD B corresponding to the address ranges listed in Table 2. These equations are listed below:

$$PB4 = /PER1CS = /(A15 \cdot A14 \cdot /A13 \cdot A12)$$

$$PB5 = /PER2CS = /(A15 \cdot A14 \cdot A13 \cdot /A12)$$

$$PB6 = /PER3CS = /(A15 \cdot A14 \cdot A13 \cdot A12)$$

Finally, PB7 is used to perform the bus sizing function. It should be sized to eight bits whenever any of the external peripherals is accessed. It should be sized to 16 bits for all other accesses. The 80C196KB requires a high on the BUSWIDTH input for 16-bit operation and a low for 8-bit operation. This is accomplished by the equation below:

$$PB7 = BUSWIDTH = /(A15 \cdot A14 \cdot A13 + /A15 \cdot A14 \cdot /A13 \cdot A12)$$

This completes the equations for Port B. These equations are entered in the Maple software by selecting the Port B chip select definition screens as described in the previous example and entering 1's and 0's in the appropriate locations. Remember that don't cares (X's or blanks) must be entered in all inputs which are not used by a particular equation.

Finally, we must enter the memory map into Maple Address Map screen. This is performed as in the previous example by entering 1's, 0's or don't cares in the appropriate places.

Conclusion

The PSD device may be used in a variety of applications requiring the simplicity, space savings and performance possible by the integration of memory and programmable elements. But a significant portion of the value of the PSD device, is its ability to absorb much of the logic functionality which normally surrounds a

microcontroller application. The programmability of the device allows the designer to make changes to both the software and the design itself as required. This is not possible with masked ROM or ASIC-based designs. The PSD device can truly turn a microcontroller into a complete two-chip solution.



Programmable Peripheral Application Note 015 Using Memory Paging with the PSD3XX

By Jeff Miller

Introduction

The PSD3XX is a compact, high performance microcontroller peripheral used to extend the capabilities of a microcontroller in a space-limited embedded control system. It provides the programmable logic, memory and I/O requirements needed by most microcontroller designs in a single small package.

The PSD301, introduced in November 1990, was the first of a six-member family of devices providing varying amounts of on-chip resources. The PSD301 contains 32K Bytes of EPROM for program storage and 2K Bytes of SRAM scratchpad memory. As the family expanded, the EPROM memory size grew to 128K Bytes in some versions. This large memory may be needed in many applications requiring large feature sets. In many cases the

microcontroller is capable of addressing only 64K Bytes of memory with its limited 16-bit address bus. In these applications, the designer is often faced with the difficult choice of eliminating features, using a more expensive microcontroller with a wider address bus, or adding external paging logic requiring several extra components.

With this in mind, designers at WSI have included a simple but very effective paging system in the PSD3XX models containing more than 32K Bytes of EPROM. This enables cost effective microcontrollers like the 80C31, 80196, Z80, 68HC11 and others to take full advantage of additional memory without any additional hardware or design effort.

What is Paging?

The primary purpose of a page register is to extend the width of the address bus by a number of bits to increase the size of the address space. These bits are added to the address bus as outputs of a register which is loaded from the data bus of the MCU. Each additional bit doubles the effective address space. Though the page register address bits increase address space, they are not the same as the true address bits which are generated by the microcontroller since they do not appear with the same timing or sequence of the address. They must be controlled carefully to avoid unexpected behavior. They can also be a problem for compiler-generated code since the compiler does not inherently know how to use a page register. Because of this, the designer must take care in designing software which uses the PSD3XX page register.

The purpose of this note is to explain the usage of the page register and some of the techniques which may be used when designing software which uses the page register. A typical page register design is shown in Figure 1. In the figure, a typical 8-bit microcontroller with a multiplexed address/data bus is shown

connected with the logic required to implement a 4-bit page register. The least significant address bits are demultiplexed from the data bus by the '573 transparent latch, which is clocked by the ALE signal. The most significant 8-bits of address are driven directly by the microcontroller. When combined with the least significant address bits from the address latch, the address bus is 16-bits wide. This provides the capability to directly access 64K Bytes of address space, which may be any combination of program and data storage. To implement more address space, two '74 devices (a dual D-type flip flop) have been used to create a page register. The inputs of the '74 are four bits of the address/data bus. These bits are stored into the '74 when a write to a specific address, as decoded by the '138, is performed by the microcontroller. The outputs of the '74 form an additional 4 address bits, thus extending the address bus to 20-bits or 1 MByte of address space. The '74 page register can be considered to hold a page number. Each page number provides a complete duplication of the microcontroller's memory space. To get to another 64K Byte page of address space,



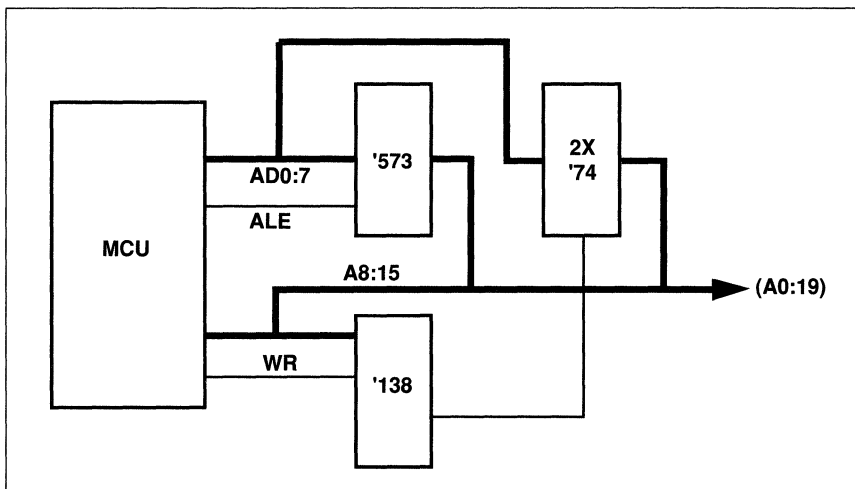
What is Paging (Cont.)

the controller simply has to change the page number by writing a different value to the page register.

The circuit below has one major complication. If the microcontroller is currently in a particular memory page, page X, and it changes the page number to Y using a store instruction which it fetched from page X, as soon as the store is

complete the next instruction fetched will come from page Y. This means that page Y must pick up the programming sequence exactly as it was left off from page X. This is a complication that must be handled in software and can make programming very difficult. Additionally, interrupts can be a significant problem since they must force the program to an interrupt vector which may exist on a different page.

Figure 1.
Discrete Page Register



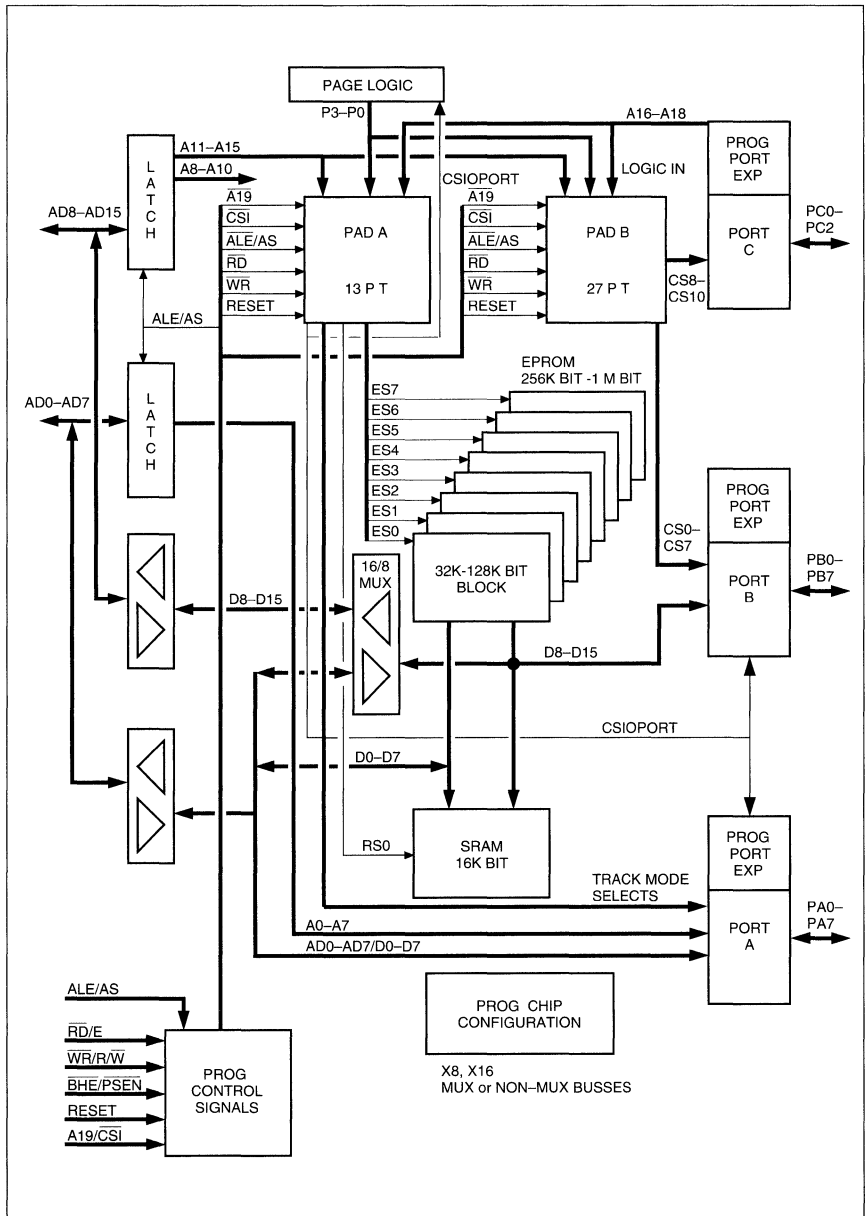
The PSD3XX Implementation

Figure 2 illustrates the block diagram of the PSD3XX with the internal page register. It is similar to the discrete circuit above, but with some important differences. The page register provides 4-bits of additional addressing capability, but does not provide them directly to the memory devices themselves. Instead, the page register output bits are taken into the Programmable Array Decoder of the PSD3XX. This enables the user to program them as necessary for the system design.

The PAD provides a flexibility that most page register implementations are not capable of providing. If you are unfamiliar with the capabilities of the PSD3XX PAD, please consult Application Note 014, *Using the PSD3XX PAD for System Logic Replacement*. Figure 3 illustrates the PAD logic in a PSD3XX with a page register. The PAD generates the outputs which are used to select the PSD3XX's eight EPROM blocks, the SRAM block, the I/O ports, the shared resource interface, the page register itself and all external functions which use

the chip selects provided by Port B and Port C of the PSD3XX. Thus, the page register bits may be combined with the address bits and control signals in any combination to generate the select signals for all of the above resources. In addition, any or all of the page register bits may be don't cares in any or all of the PAD chip select equations, enabling the user to select which resources may be selected from which page, or to select some resources from any page. This extremely useful feature enables the programmer to avoid the problem of software continuity between pages described above by making at least one of the EPROM blocks appear in all pages and then using that block to contain code for interrupt servicing and page switching. This is performed simply by making the page register bits 'don't cares' in the chip select equation for that block. All of this is fully programmable with the PSD3XX, enabling the designer to choose the paging scheme that is best for the application.

Figure 2.
PSD3XX
Architecture



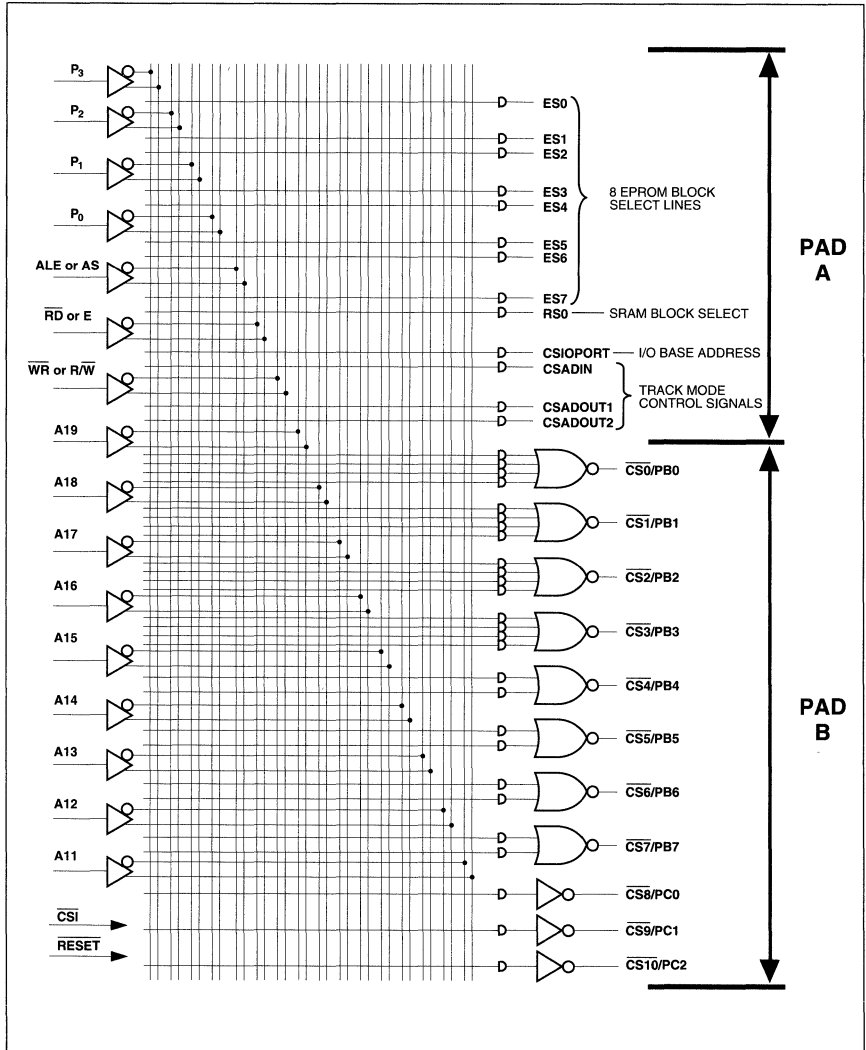
1

**The PSD3XX
Implementation
(Cont.)**

The Microcontroller can write or read the page register to place a new page number in it or read the current page number. To perform this, the microcontroller must simply access the address programmed in the PAD for the page register. This address

is based on the CSIOPORT select signal programming. If address 8000 hex is programmed for CSIOPORT, the corresponding page register address is 8018 hex and read and write data will be to and from the page register.

**Figure 3.
PSD3XX PAD
Diagram**



A Simple Paging Example

To illustrate the operation of the PSD3XX page register, assume that a designer requires a full 128K Bytes of program storage space, 32K Bytes of buffer SRAM and three peripheral devices which also must be memory mapped. We can also assume that the required program is easily broken into four modules which are somewhat independent, but do need the capability to call one another and must be able to pass global data among one another. Further, assume that the external peripheral devices may be selected from three of the four modules, but must not be accessed from the fourth for security reasons. Lastly, assume that the designer is constrained by cost and compatibility considerations to use an 8-bit microcontroller with a 16-bit address bus (in this example, an 8031).

These requirements may be easily implemented using the PSD313 device. The PSD313 is an 8-bit device with 128K Bytes of EPROM for program storage. It also contains the PAD and page register logic described above. The memory map required for this application is shown in Figure 4.

The memory map shown utilizes the page register to provide a unique address for all of the PSD313's 128K Bytes of EPROM in addition to the SRAM and peripherals. This memory map consists of four pages of 64K Bytes each. The map is further divided into program and data space by the PSEN and RD signals which are available in the 8031 microcontroller. This enables the PSD313 to overlap the addresses of the EPROM, I/O and SRAM. The pages are numbered 0 – 3, and are written into the page register by the microcontroller. The page register is part of the I/O addressing and resides in the RD = 0 map.

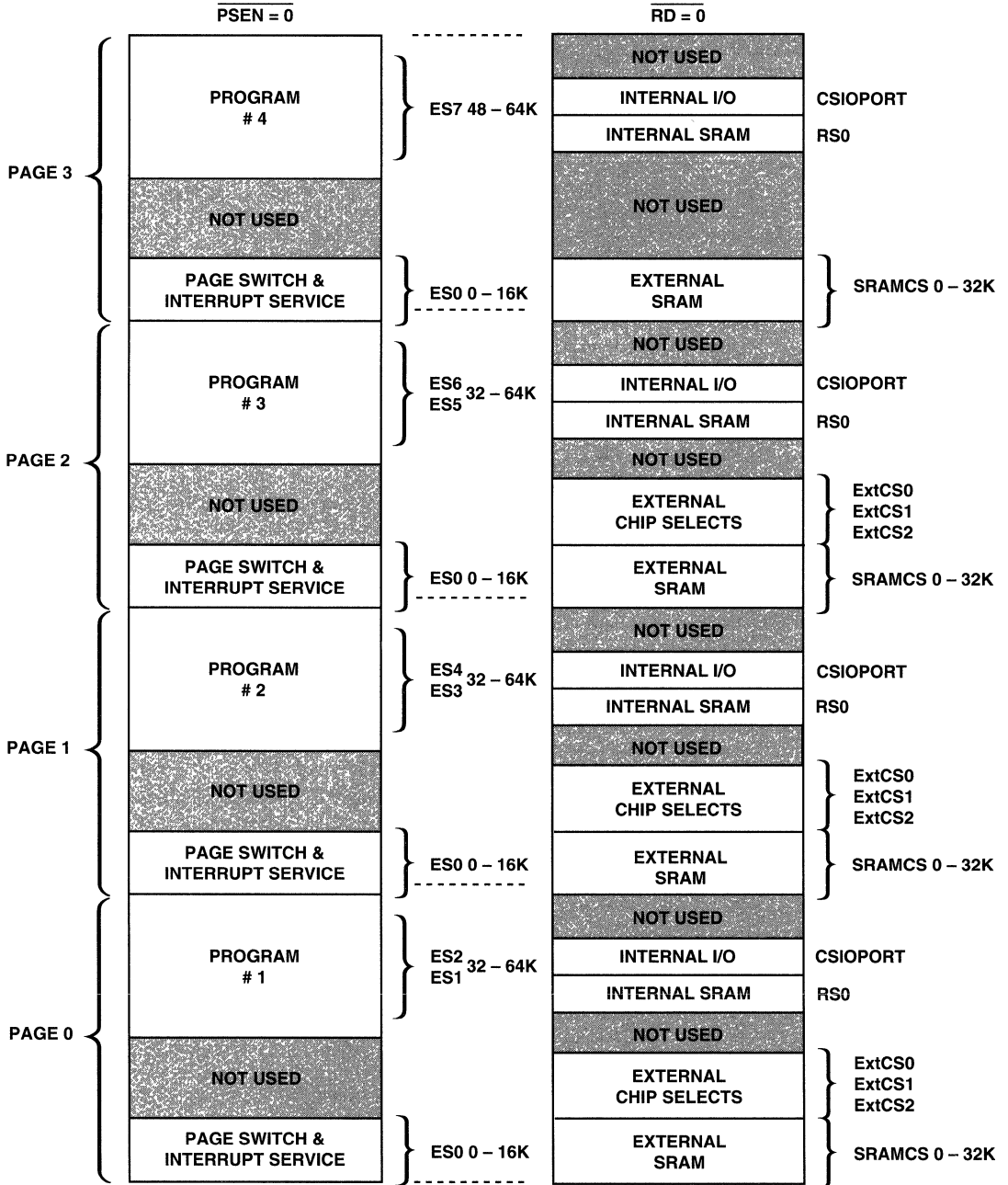
The software must be broken into four segments, one residing in each page, in order to function efficiently with this memory scheme. The software which enables the machine to boot, service interrupts and switch memory pages is located in a block of EPROM which is mapped into all memory pages. This enables simple page switching and interrupt servicing regardless of the page that the microcontroller is currently operating in. Locating an EPROM block in

multiple pages is very simple using the PAD 'don't care' feature. In this example, EPROM block 0 has been chosen to hold the page-independent software. The PAD output which controls block 0 is ES0. Therefore, in the definition of the ES0 signal, all four of the page register bits (P0 – P3) are programmed as 'don't cares'. ES0 is further defined to be from address 0000 to 3FFF. Thus, whenever the microcontroller places an address on the bus which is in this range with PSEN low, the data will be read from EPROM block 0, regardless of the contents of the page register.

The remaining EPROM blocks are evenly distributed into the four pages. This segmentation has been used in this example, but there is no requirement that the pages contain equal memory sizes. Each can have a different amount of resources contained within it. We have placed EPROM blocks 1 and 2 in page 0. This is done by requiring P0 – P3 to be 0's to generate the ES1 and ES2 selects. Similarly, ES3 and ES4 in page 1, ES5 and ES6 in page 2 and ES7 in page 3 require the P0 – P3 signals to be in the correct states to generate the ES signals.

The SRAM and I/O devices most likely must be accessible from all pages, like the page switching software and interrupt service routines. In this way, each of the program segments may store and load data from the SRAM which may be used to pass global parameters among the programs. All programs may also communicate with the external I/O devices, which is most likely required. It is very important that the internal PSD3XX I/O registers, which include the I/O port control and data registers as well as the page register itself, be mapped into all pages. Otherwise, after the page has been switched, there will be no way of switching back to the original page since the page register would not be accessible. To make the page register accessible from all pages, the designer must simply make the page register bits (P0 – P3) 'don't cares' in the equation for the CSIOPORT signal. This can also be done for any of the external chip select equations which are generated by the PAD and brought to the outside world through Port B or Port C.

**Figure 4.
Memory
Map**



A Simple Paging Example (Cont.)

If it is desirable for some pages not to have access to some resources, this may be done also. The designer must simply use the page register bits in the equation which selects the resource which is to be protected. This can provide a program security or error handling feature while protecting certain I/O or memory devices from accidental corruption.

Figure 5 contains the output of WSi's Maple software for the above example. The part chosen to implement the sample design was the 8-bit only PSD313, chosen because it contains the required 128K Bytes of EPROM but is less expensive than the PSD303. The PSD303, which also contains 128K Bytes of EPROM, can be configured in a 16-bit data bus mode which would be suitable for use with 16-bit microcontrollers like the 80196.

The PSD313 was programmed and configured to implement the memory map shown in Figure 4. Not all of the capability of the PSD313 has been utilized in this example but is available to satisfy other system design requirements if necessary. The PSD313 has been configured to function with the 8031 microcontroller and its associated control signals. This can be seen in the Configuration portion of the output file in Figure 5. We have also configured Port B 0-3 to provide the required chip select functions for the external I/O and SRAM devices. These chip selects have been given the aliases ExtCS1, ExtCS2 and ExtCS3 for the I/O devices and SRAMCS for the SRAM. The equations entered for the chip selects correspond to the addresses for which they should be active. ExtCS1 will become active when address 8000 - 87FF hex is accessed. ExtCS2 and ExtCS3 will become active for addresses 8800 - 8FFF hex and 9000 - 97FF hex respectively. The SRAM chip select will become active for address 0 - 7FFF hex. All of these chip selects will function independently from the page register contents since the page register outputs (P0 - P3) do not appear in the equations. This means that all of these external devices will be selectable from any page.

The address map lists the start and stop addresses and the page numbers for each of the blocks of memory and I/O inside the PSD313. The first EPROM block is selected by ES0, which has been mapped from address 0000 to 3FFF hex. This block has been designated to contain the page switching software and the boot and interrupt service routines. Since all pages need the capability to switch from one to another, and since an interrupt may be received at any time while the software is executing in any page, EPROM block 0 has been made accessible from all pages by making the page register bits 'don't cares' (x's) in the address map for ES0.

ES1 and ES2 map EPROM blocks 1 and 2 into address 8000 - FFFF hex in page 0. Thus, whenever a program address within this range is accessed by the microcontroller while the page register contains a 0, ES1 or ES2 will activate EPROM block 1 or 2. While the microcontroller is executing code from one of these blocks in page 1, it may still access internal or external SRAM, or internal or external I/O without changing pages. ES3 - ES7 are mapped to pages 1 - 3 in a similar manner.

In addition to the external SRAM, the PSD313's internal SRAM has been mapped into all address pages where it may be used to supplement the microcontroller's register file and internal SRAM. This SRAM may be used for global variable storage, stack space or many other purposes. The PSD313's I/O ports have been mapped at address C800 - CFFF hex in this example. This places the page register address at C81A hex (see the PSD3XX data sheet for I/O addressing in the PSD3XX). As discussed earlier, the page register has been mapped into the same address from all memory pages, so that it may be accessed from all program subroutines in the system.

**Figure 5.
MAPLE
Software
Example**

```

PSD PART USED: PSD313
*****PROJECT INFORMATION*****
Project Name   :   = Page Register App Note
Your Name     :   = Jeff Miller
Date         :   = 1/15/92
Host Processor:   = 8031
*****

*****ALIASES*****
/CS4         = ExtCS1
/CS5         = ExtCS2
/CS6         = ExtCS3
/CS7         = SRAMCS
*****

*****GLOBAL CONFIGURATION*****
Address/Data Mode:   MX
Data Bus Size   :   8
Reset Polarity  :   HI
Security       :   OFF
ALE Polarity    :   HI
A15-A0 ALE dependent (Y) or Transparent (N):   N
Using Different READ strobes for Data and Program: Y
*****

*****READ WRITE CONTROL*****
/RD and /WR

*****

*****PORT A CONFIGURATION *****
ADDRESS/IO

*****

*****PORT A (ADDRESS/IO)*****
PIN   Ai/IO   CMOS/OD
PA0   A0     CMOS
PA1   A1     CMOS
PA2   A2     CMOS
PA3   A3     CMOS
PA4   A4     CMOS
PA5   A5     CMOS
PA6   A6     CMOS
PA7   A7     CMOS

*****

*****PORT B CONFIGURATION*****
Pin   CS/IO   CMOS/OD
PB0   CS0     CMOS
PB1   CS1     CMOS
PB2   CS2     CMOS
PB3   CS3     CMOS
PB4   CS4     CMOS
PB5   CS5     CMOS
PB6   CS6     CMOS
PB7   CS7     CMOS
*****

```

**Figure 5.
MAPLE
Software
Example
(Cont.)**

*****PORT B CHIP SELECT EQUATIONS*****

```
ExtCS1 = / (A15 * /A14 * /A13 * /A12 * /A11 * /P0
          + A15 * /A14 * /A13 * /A12 * /A11 * /P1)

ExtCS2 = / (A15 * /A14 * /A13 * /A12 * A11 * /P0
          + A15 * /A14 * /A13 * /A12 * A11 * /P1)

ExtCS3 = / (A15 * /A14 * /A13 * A12 * /A11 * /P0
          + A15 * /A14 * /A13 * A12 * /A11 * /P1)

SRAMCS = / (/A15)
```

*****PORT C CONFIGURATION*****

Pin	CS/Ai	LOGIC/ADDR
PC0	A16	ADDR
PC1	A17	ADDR
PC2	A18	ADDR
A19	CSI	

*****PORT C CHIP SELECT EQUATIONS*****

*****ADDRESS MAP*****

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	FILE	FILE	File Name	Page	Reg	Q.F
	19	18	17	16	15	14	13	12	11	STRT	STOP	STRT	STOP		3210		ALE
ES0	N	0	0	0	0	0	N	N	N	0	3fff	0	3fff	PROG0.HEX	XXXX		X
ES1	N	0	0	0	1	0	N	N	N	8000	bfff	0	3fff	PROG1.HEX	0000		X
ES2	N	0	0	0	1	1	N	N	N	c000	ffff	4000	7fff	PROG1.HEX	0000		X
ES3	N	0	0	0	1	0	N	N	N	8000	bfff	0	3fff	PROG2.HEX	0001		X
ES4	N	0	0	0	1	1	N	N	N	c000	ffff	4000	7fff	PROG2.HEX	0001		X
ES5	N	0	0	0	1	0	N	N	N	8000	bfff	0	3fff	PROG3.HEX	0010		X
ES6	N	0	0	0	1	1	N	N	N	c000	ffff	4000	7fff	PROG3.HEX	0010		X
ES7	N	0	0	0	1	1	N	N	N	c000	ffff	0	3fff	PROG4.HEX	0011		X
RS0	N	0	0	0	1	1	0	0	0	c000	c7ff	N/A	N/A	N/A	XXXX		X
CSP	N	0	0	0	1	1	0	0	1	c800	cfff	N/A	N/A	N/A	XXXX		X

*****END*****

*****ADDRESSES OF I/O PORTS*****

```
Pin Register of Port A : C802 Page (Binary): XXXX
Direction Register of Port A : C804
Data Register of Port A : C806
Pin Register of Port B : C803
Direction Register of Port B : C805
Data Register of Port B : C807
Page Register : C818
```



Software Considerations

The software example shown in Figure 5, has been divided into four sections to facilitate placing it into the four pages. These four program blocks have been called PROG1.HEX, PROG2.HEX, PROG3.HEX and PROG4.HEX. In order to create these files to be loaded into the PSD3XX, the software designer must plan for this event when the software is written. It is most easily accomplished by breaking the tasks into logical groups that do not need to access one another frequently. Most software can be split in this manner. Then, the designer can create the page switching algorithm which is used to jump between the tasks which are on different pages.

There are many ways to implement this capability, but we will provide as an example one method which can be used. This method of memory paging involves the use of a table of addresses and page numbers of all program tasks which may be called from page to page. This table can be made global when the code is compiled so that it may be used in all four of the programs used in this example. This table would reside in EPROM block 0 along with the interrupt service routines and page switching algorithms so that it may be accessed from all memory pages. Thus, when PROG1 is executing and must run a task or subroutine which is in PROG2, the software should jump to the page switching algorithm while passing the table lookup address of the task that it wishes to run. In this way, only the pointer into the table must be known by all programs instead of the address and page number of each routine. This simplifies the process of modifying the software by permitting the programmer to keep all of the pointers into the table constant, even if the actual subroutine addresses change. In this table, the page switching routine will find the page that it must switch to as well as the address to jump to after the page has been switched. The return address and page number may simply be pushed onto the stack, which is stored in the SRAM. Since the SRAM is also page independent, all programs may share the same stack.

To build the table, the labels of all subroutines which may be shared among pages must be accumulated from all of the programs. These labels must be placed in the table along with the corresponding page numbers. This table must then be placed in the global EPROM block. The labels must be made global so that each program may have access to them. Then pointers into the table must be assigned, one for each global subroutine. These must also be made global so that they may be used by each program. The pointers must remain constant, even when the software is modified. This way, software modifications may change the values of the labels, but not the pointers.

This provides a very clean paging solution which may be implemented using high level language compilers. The only penalty when using this method is the overhead experienced when switching from page to page. This overhead may be minimized by careful software design which minimizes the number of program calls and jumps between routines on different pages. Care must also be taken when nesting jumps from page to page if it is important to keep track of return addresses. Interrupts, since they are accessible from all pages, are very simple to handle. The page need not even be switched to service an interrupt unless the service routine needs to access a task which is not located in the global EPROM block. Even then, the only consideration is that before returning from the interrupt, the page number must be restored to its value prior to the interrupt. This paging scheme is illustrated in Figure 6.

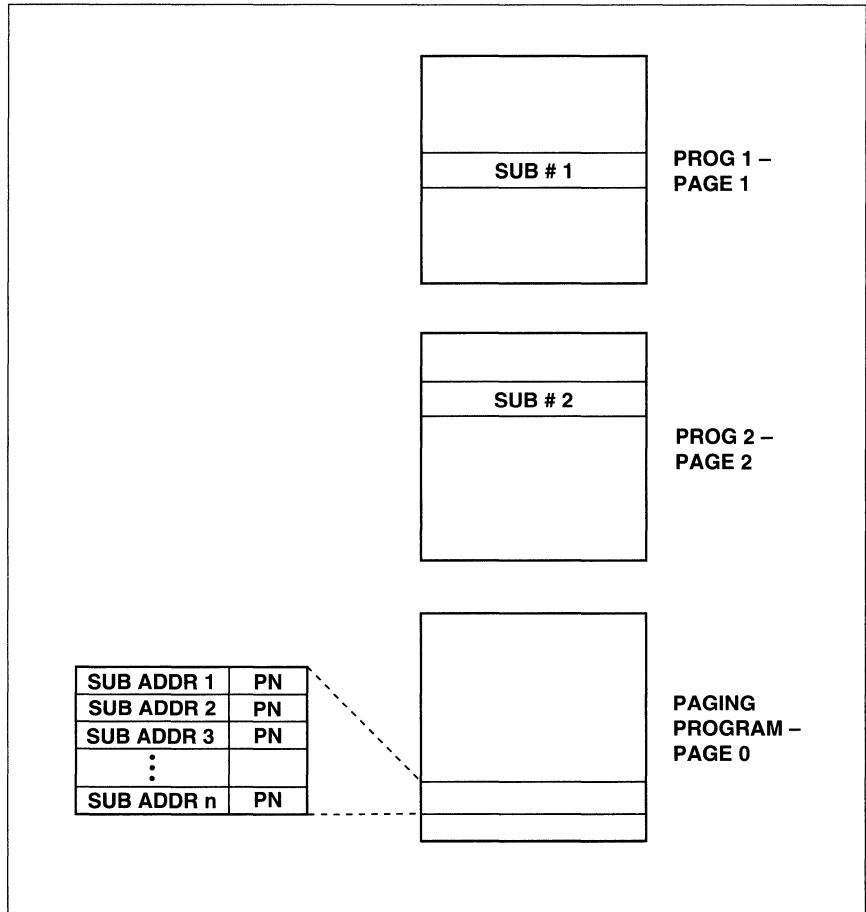
Compiler Issues

The paging algorithm shown below is relatively easy to implement and somewhat automatic. However, it is not a totally transparent solution for the software programmer. There is such a solution available from at least one compiler manufacturer. Archimedes makes compilers for several microcontrollers including the 8031 family and 68HC11 family. These compilers are available with built in memory paging which use some of the microcontroller's port bits as additional

address bits. These compilers generate bank switching code automatically which can be easily modified to utilize the page register inside the PSD3XX.

When this is done, the use of the page register becomes transparent to the user. The attached code excerpt shows the calling structure resulting from the use of the Archimedes compiler with the modifications to utilize the PSD3XX page register.

Figure 6. Software Paging Flow



**Archimedes
Code**

```

836
837                               /* Init DCC & SCR registers */
838   init_dsl_hdsl_dcc_scr();
\   0268 7400                     MOV     A,#$BYTE3 init_dsl_hdsl_dcc_scr
\   026A 900000                    MOV     DPTR,#$REFFN init_dsl_hdsl_dcc_scr
\   026D 120000                    LCALL  ?X_CALL_L18
839
840                               /* Init Master/Slave Polynomial */
841   init_ms_poly();
\   0270 7400                     MOV     A,#$BYTE3 init_ms_poly
\   0272 900000                    MOV     DPTR,#$REFFN init_ms_poly
\   0275 120000                    LCALL  ?X_CALL_L18
842
843
844
845           }

```

- Notes: 1. \$Byte 3 is a directive that addresses the "page" of the specified function.
2. \$REFFN Addresses the 16-bit offset of the function.

```

MODULE    ?BANK_SWITCHER_L18
TITLE     '8051 - C - BANK-SWITCHER'
RSEG     RCODE

;-----;
;
;                               - L18.S03 -
;
;   Function(s):   Banked switched CALL and RET
;
;   Must be tailored for actual bank-switching hardware.
;   In the sample system the P1 port was used.
;
;   Version: 4.00 [IANR]
;-----;

;-----;
;
;   Call a non-local function
;
;-----;
;
;   Inputs:
;   Stack: 16-bit return address
;   DPTR: 16-bit function-address
;   A: 8-bit page address
;-----;

```

The above Archimedes code is courtesy of Jeff Fayne, Tellabs, Inc.



Archimedes Code (Cont.)

```

PUBLIC  ?X_CALL_L18
?X_CALL_L18
=====
      Save old bank
=====

      PUSH  P1

=====
      Bank-switch
=====

      MOV   P1,A

=====
      Go to function
=====

      CLR  A
      JMP  @A+DPTR

;-----;
;           ;
;   Leave current function           ;
;-----;
;           ;
;   Input:                             ;
;   Stack: 24-bit return address      ;
;-----;
PUBLIC  ?X_RET_L18
?X_RET_L18:
=====
      Bank-switch
=====

      POP  P1

=====
      Return
=====

      RET

      END

```

1

Conclusion

The PSD3XX page register system can greatly assist designers of systems requiring large memory spaces with 16-bit address buses. The PSD3XX offers capability not found in most discrete page register implementations. The capability to define global resources as well as page-specific resources enables the designer to implement the paging technique most suitable for the application. The page register is included in the PSD302, PSD312, PSD303 and PSD313 devices, all of which are pin compatible

with one another. This provides the capability of expanding the memory size as required even after a system has been designed. The designer can simply drop the new, and larger, PSD3XX into the same footprint as the old, and update the software to add more memory pages. This capability can be important for product feature additions after a design is complete. Since the system is fully programmable, it may be updated and changed anytime.



Programmable Peripheral Application Note 016 Power Considerations In The PSD3XX/3XXL

By Jeff Miller

Introduction

The PSD3XX is a configurable microcontroller peripheral integrating programmable logic, EPROM and SRAM technologies into a single piece of silicon. It has been used extensively in microcontroller applications around the world by virtue of its high level of integration, configurability and ease of use. This integration makes possible the design of very compact microcontroller systems, enabling the user to squeeze a great deal of functionality into a very small space. Thus, the PSD3XX has found its way into many small hand-held and/or battery operated applications such as cellular phones, medical instrumentation and laptop or notebook computers which usually require, in addition to small space, a very low power consumption.

The PSD3XX family is based on a patented high-performance CMOS technology and,

like other CMOS devices, requires very low power consumption even when no particular effort is made to minimize the PSD3XX power. But, when some special care is taken during the programming and configuration of the device, power can be reduced even further, making the PSD3XX even more valuable in these power-sensitive applications. This application note will describe the methods which can be used to reduce the PSD3XX power consumption in both active and stand-by modes. It makes sense to use some of these techniques even when low power is not a primary design requirement since they are easy to implement and require no additional expense. We believe that proper implementation of the material in this note will make the PSD3XX an invaluable member of any low-power microcontroller system.

Power Use In The PSD3XX

The PSD3XX contains several modules internally, each of which can be considered a power consumer when in operation. These modules include the PAD, (Programmable Address Decoder) EPROM and SRAM blocks. The key to reducing the power used by the PSD3XX is to reduce the power used by each of these modules individually.

Under normal operation, several of the functional modules may be operating, while others may be standing by. A module in stand by uses much less power than one that is active. For example, whenever the SRAM is not being actively used, it is disabled and therefore consumes less power. This is also true of the PAD. A PAD term which is active expends more power than one which is inactive. This would also be true of the EPROM. However, in some PSD3XX models, the EPROM is always active, in which case it will always draw power. This is done in order to provide the best access time possible for the EPROM. The Low Power family of PSD3XXs does not keep the EPROM enabled at all times,

and thus the designer can save power by minimizing the time during which the EPROM is accessed. Use of this feature does impact the speed of the PSD3XX EPROM, which results in the loss of the 120 ns speed grade. There are other methods of reducing EPROM power even when the EPROM is enabled. These will be discussed in detail later in this note. When the time that each PSD3XX function is kept in standby mode is maximized, the power expense is minimized.

There is a way to place the entire PSD3XX into the standby mode at once, thereby reducing power usage to the bare minimum. This can be done through the use of the CSI (Chip Select Input) pin. When the PSD3XX is deselected by the CSI pin, the entire part enters the standby mode using only about 50 μ A of current. While in this mode, the PSD3XX is incapable of performing any functions, including PAD logic equations, but this is an excellent method of reducing system power in designs which have low active duty cycles.

CMOS Power Characteristics

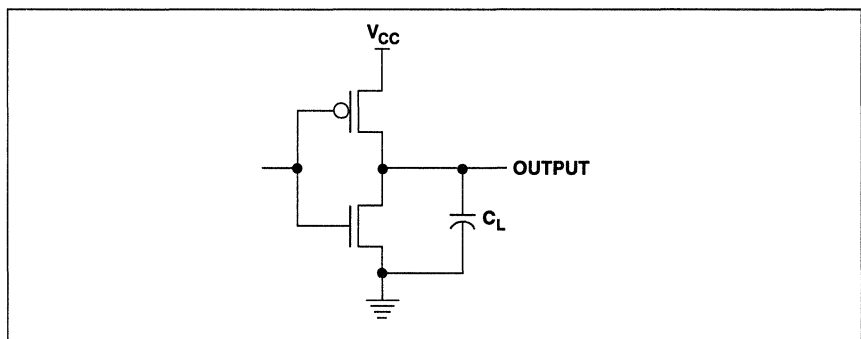
As a CMOS part, the PSD3XX behaves in the same way as other CMOS devices in terms of power dissipation. The PSD3XX consumes the most power when the temperature is low, the voltage is high and the frequency is high. Low temperature in CMOS devices, unlike in bipolar devices, causes the transistors to speed up, thus consuming more power. Therefore, if the system will never operate in low temperature environments, power dissipation will be lower. Another result of this characteristic is that CMOS parts do not generally experience thermal runaway. As temperature increases, the power expended by the CMOS device decreases, thus the part tends to effectively cool itself off.

Another characteristic of CMOS devices is the effect of voltage variations. CMOS behaves similarly to TTL devices with respect to voltage. When input voltage

rises, the current drawn by a CMOS device also rises. As input voltage falls, input current also falls. Thus, the CMOS device will draw the least current at its lowest allowable supply voltage. This voltage is 4.5V in the PSD3XX. Taking the voltage below this level will generally slow the device down to below its specified speed as well as jeopardize its data retention capability. Between 4.5 and 5.5V, the PSD3XX varies by about 0.85mA per 0.1V variation. Thus, the PSD3XX will draw approximately 0.85 mA less current at 4.9V than at 5.0V V_{CC} .

Lastly, frequency of operation plays an important role in the power dissipation of a CMOS device. A CMOS gate expends the greatest power while it is switching between the logic 0 and logic 1 states, or vice versa. This can be easily understood when looking at the circuit diagram for a typical CMOS output shown in Figure 1.

Figure 1.
Typical CMOS
Output Circuit



The circuit above represents a typical CMOS inverter output. Normally, either the top transistor is off (output = logic 0) or the bottom transistor is off (output = logic 1). MOS transistors have very low leakage currents which means that under these normal conditions, very little current will be passing from V_{CC} to ground. However, when the input to the inverter is switching, both transistors will not switch from their present conditions to their new conditions at precisely the same instant. Therefore, both transistors will be on for a very brief instant during the transition. During this time there is a low impedance path from V_{CC} to ground and some current is drawn by the circuit. In addition, the output will have some load capacitance (C_L) which must be charged during switching,

even if the load itself draws little or no static current. Thus, during the switching process the power expended by a CMOS device is at its highest.

The switching current drawn by the device is dependent on the number of times the outputs are forced to switch logic states in a unit of time. Therefore, the frequency of operation of the part directly influences its dynamic power consumption. The lower the operational frequency, the lower the dynamic power expended by the device. In the PSD3XX, frequency of operation is determined by the rate at which the addresses are changing, usually indicated by the frequency of the ALE or AS signal. Generally, the PSD3XX draws about 3 mA of additional current for each 1 MHz added to the frequency of operation.

Power Management Techniques In The PSD3XX

The above mentioned features and characteristics can be used to the designer's advantage when designing compact microcontroller systems which have a tight power budget. In the sections that follow, several methods for reducing the PSD3XX power will be presented.

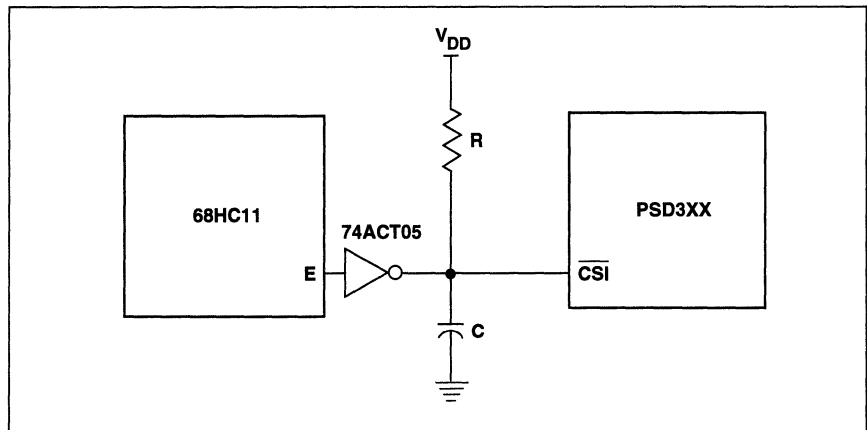
Power Down Mode

Many system designs do not require the microcontroller, and therefore the PSD3XX, to operate continuously. Systems, like cellular telephones and notebook computers, spend a large amount of time inactive – waiting for something to happen like a press of a button or keyboard. During this time, many designers place the microcontroller into a low power idle or sleep mode. In the sleep mode, the controller expends significantly lower power. The microcontroller is usually awakened by some event – a key on a keypad being pressed, for instance, which may result in an interrupt. There is no need for the PSD3XX to be active during the time that the microcontroller is not active.

Therefore, the PSD3XX should be placed in the power down mode (CSI inactive) to reduce the PSD3XX current down to its standby value.

The PSD3XX must also be awakened when the microcontroller is awakened so that it may provide an instruction to the controller when it requires one. If the microcontroller itself has a chip select output, like the Motorola 683XX series controllers, it may be used to awaken the PSD3XX as necessary. However, if it does not, there will be a problem. If the microcontroller itself is used to power down the PSD3XX, through an I/O port pin for example, there will be no way to power up the PSD3XX again since the PSD3XX itself contains the instruction that the microcontroller must use to activate the CSI signal to awaken the PSD3XX. The way to correct this situation is to design a circuit which detects when the microcontroller is coming out of its power down mode before it must fetch the first instruction. Such a circuit is depicted in Figure 2.

Figure 2.
Simple Power Down Circuit



In this circuit diagram, a Motorola 68HC11 microcontroller is connected to a PSD3XX in a low power system. The circuit functions quite simply. The E signal from the HC11 is normally a free running clock at 1/4 the frequency of the input clock. When the HC11 is placed into the sleep mode by the software (by executing the STOP instruction), the E signal stops oscillating and remains low until an interrupt or internal timer event occurs. After the

interrupt has been received by the controller, the E signal resumes toggling, but there will be a minimum of two E clock cycles prior to the first AS. This characteristic can be used to place the PSD3XX into its low power standby mode whenever the STOP has been executed in the HC11 and to awaken it before it must supply an instruction to the HC11.

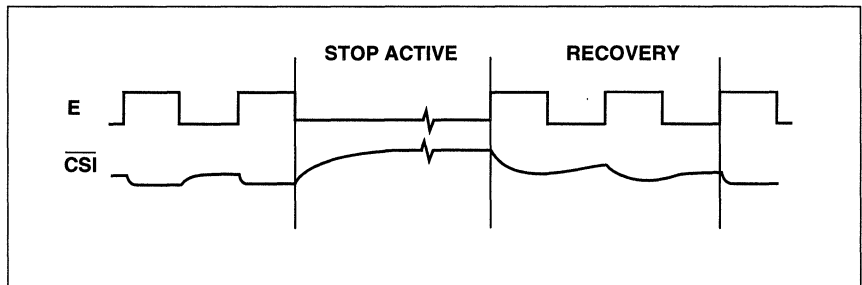
Power Management Techniques In The PSD3XX (Cont.)

The ACT05 device shown in the diagram is simply an open collector inverter. When the E signal is oscillating, the output of the inverter will be toggling between ground and high impedance. When the output is at ground, the capacitor will rapidly discharge from its present state into the ACT05. When the output is high impedance, the capacitor will slowly charge up to V_{CC} through the resistor. Thus, under normal operation the \overline{CS} input of the PSD3XX will be at or near 0 V, provided the RC time constant is large enough to prevent the capacitor from charging up beyond a logic zero level of 0.6 V.

When the HC11 enters the sleep mode the E signal remains low. This enables the

capacitor to slowly charge up to a logic one level which then places the PSD3XX into the standby mode in which it will consume only about 50 μ A of current. After the controller exits the sleep mode, the E signal will resume oscillating which rapidly discharges the capacitor. This, in turn, activates the \overline{CS} input to the PSD3XX, bringing it out of the power down mode. Since the E signal will oscillate for at least two full cycles before the first AS strobe begins a new bus cycle, the PSD3XX will have ample time to recover from the power down mode before having to supply an instruction to the HC11 for processing. In operation, the circuit results in a timing diagram similar to the one in Figure 3.

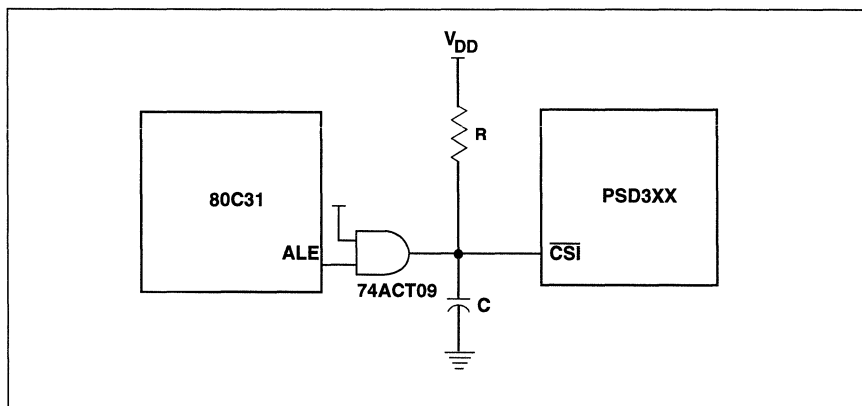
Figure 3. 68HC11 Stop Timing



A similar circuit can be used for Intel 8031 type controllers. Controllers conforming to the Intel 8031 family generally have two low power modes: IDLE and POWER DOWN. The IDLE mode causes the controller to cease instruction execution, but its internal clocks continue to run. This saves significant power while leaving the

internal timers and other functions operational. When in the IDLE mode, both the ALE signal and the \overline{PSEN} signal are held high. A circuit similar to the one illustrated for the 68HC11 may be used to detect the end of oscillation on the ALE signal. This circuit is shown in Figure 4.

Figure 4. 8031 Idle Circuit



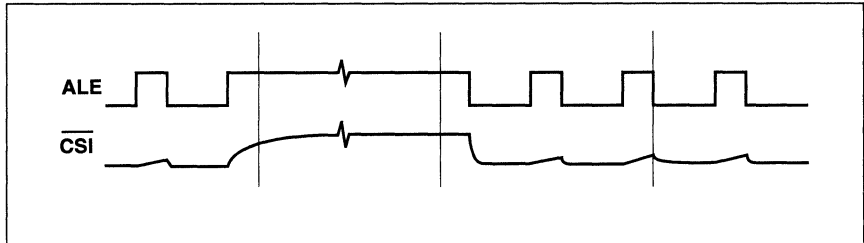
Power Management Techniques In The PSD3XX (Cont.)

The circuit operates on the same principle as the one used earlier for the Motorola processor. The ALE signal normally oscillates high for 2 clocks out of every 6 or 12 clocks, depending on whether instruction or data accesses are being performed. The software places the 8031 into the Idle mode by setting bit 0 in the PCON register. Once set, the ALE and PSEN signals remain high until an interrupt or hardware reset occur. During this time, the $\overline{\text{CSI}}$ signal will float high with the RC circuit, as in the earlier example. The

ACT09 is simply an AND gate with an open collector output. It performs the same function as the inverter in the previous example without inverting the signal. When an interrupt or reset is received, the ALE signal begins to toggle again, but at least two “dummy” unused ALE cycles will occur before the first meaningful instruction is fetched, giving the PSD3XX time to recover from the power down mode. The timing for the above circuit is shown in Figure 5.

1

Figure 5. 8031 Idle Timing

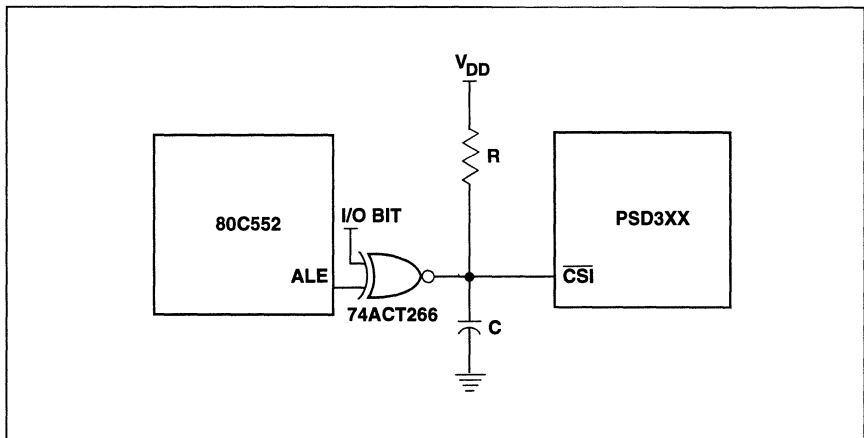


If the system requires truly the lowest power available, the 8031 POWER DOWN mode may be used. This disables all internal operations of the 8031 as well as the external ones. Thus, any on-chip peripherals like timers and serial communication links will be disabled. This places the controller into its lowest power mode possible. Software may place the 8031 into the POWER DOWN mode by setting bit 1 in the PCON register. When execution of the instruction is complete, the ALE signal will be driven low and will remain in this

state until a hardware reset is received. Thus, a circuit similar to the one above may be used to detect the static condition of the ALE signal, but an inverting gate must be used instead of the ACT09 (such as the ACT05 used in the Motorola example earlier).

If both the POWER DOWN and IDLE modes must be used, the gate may be replaced with an ACT266 exclusive NOR with an open collector output. This circuit is shown in Figure 6.

Figure 6. 8031 Power Down or Idle Circuit



Power Management Techniques In The PSD3XX (Cont.)

The I/O bit can be provided by either the PSD3XX or the controller itself. If the controller is used to provide the I/O bit, it must hold the correct value on the output even when in the idle or sleep mode, as the PSD3XX does. When the I/O bit is low, the POWER DOWN mode is enabled (a low on ALE and a LOW on the I/O bit will result in a high on \overline{CS}). When the I/O bit is high, the IDLE mode is enabled (a high on ALE and a high on the I/O bit will result in a high on \overline{CS}).

For all of the above circuits to operate correctly, the value of the RC network must be carefully calculated to insure proper operation in the normal mode. This means that under normal operation, \overline{CS} must never climb above 0.4 V, which will guarantee that it is always recognized by the PSD3XX as a low.

For example, the 68HC11 circuit shown in Figure 2 used the E signal from the controller to disable the PSD3XX. The E signal oscillates at 1/4 the frequency of the HC11's input clock. If an 8 MHz HC11 is used, the E signal will oscillate at 2 MHz. This results in an E signal clock period of 500 ns. During this 500 ns the E signal will be low for 250 ns. Thus, the RC network must be chosen to prevent the \overline{CS} signal from climbing above 0.4 V for at least 250 ns. The equation below governs the voltage across the capacitor (V_C), and thus the voltage present on the \overline{CS} pin:

$$V_C = V_{CC}(1 - e^{-t/RC})$$

where V_C is the voltage across the capacitor (which is the same as the \overline{CS} pin), V_{CC} is the supply voltage, and t is the time in seconds after the output of the open collector gate switches from a low to an open circuit. Solving for RC we get:

$$RC = -t/\ln(1 - V_C/V_{CC})$$

In order to determine the minimum values for R and C, we must solve this equation for the point of time which is of interest. We must have V_C no greater than 0.4V at time $t = 250$ ns. Thus, with $V_{CC} = 5$ V, the equation may be rewritten as follows:

$$RC = -250 \times 10^{-9} / \ln(1 - 0.4/5.0) = 3.0 \times 10^{-6}$$

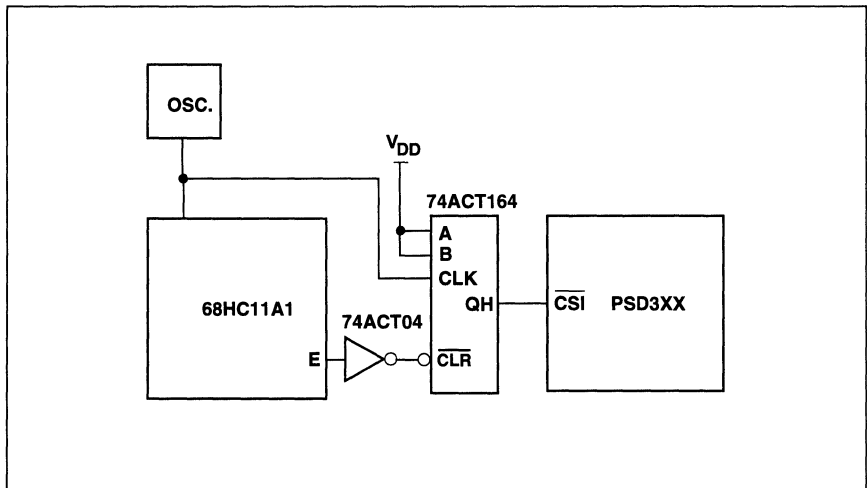
An acceptable RC network for this case might be a resistor of 100K Ω and a capacitor of 30pF. These values will provide no margin for the circuit so some additional resistance or capacitance may be desired. Of course, larger values may be used without harming the circuit, they will just cause the low power mode to be entered more slowly. The case of leaving the low power mode is less critical, since the capacitor will discharge more quickly through the gate than it will charge up through the resistor. In the interest of minimizing power use by the circuit itself, it is best to use a larger resistor value and a smaller capacitor value, since this will cause less current to be sunk by the gate which drives the circuit.

Using this equation, it is possible to determine the RC value required for any controller and/or frequency. It is only necessary to determine the length of time that the RC will be required to hold the \overline{CS} signal below 0.4 V and plug that value into the above equation.

If a more deterministic method is desired for placing the PSD3XX in the power down mode, a fully digital circuit may be implemented which uses very few additional components. This circuit is shown in Figure 7 for the 68HC11 controller.

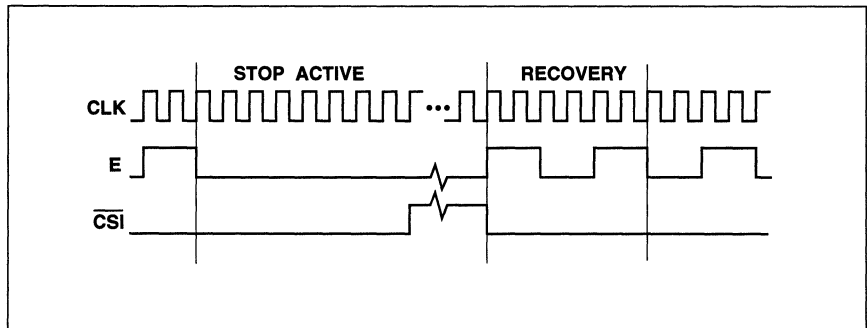
This circuit performs the same function as the RC circuit described earlier, but does it digitally. The 74ACT164 is a shift register which is used in this example to detect when eight HC11 input clocks occur while the E signal remains low. In normal operation, no more than two clocks should occur without E transitioning from low to high, thus providing a clear to the ACT164. If the HC11 is stopped, the E signal will remain high until an interrupt is received, but the input clock continues to run freely. Thus, the shift register will shift in "one's" until the E signal goes high again. When the ACT164 has shifted eight times, the \overline{CS} signal will go high, placing the PSD3XX into the power down mode. The timing diagram corresponding to this circuit is shown in Figure 8.

**Figure 7.
Digital Sleep
Circuit For
68HC11**



1

**Figure 8.
68HC11 Stop
Mode Timing**



**Power
Management
Techniques
In The PSD3XX
(Cont.)**

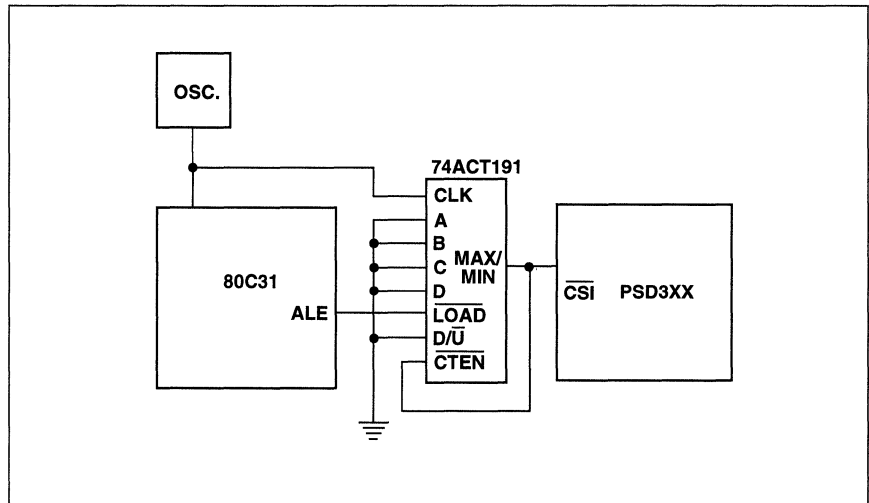
A similar circuit may be used for the 8031 family of controllers, and is depicted in Figure 9.

This circuit, like the others, detects when ALE stops toggling. Since up to 10 clocks may normally occur without an ALE pulse, a counter which can count to at least 11 is required in order to function properly. Thus, an 8-bit shift register like the one used with the HC11 will not work. In this case, a 74ACT191 is used to count 16 clocks prior to raising its MAX/MIN output high. A low on the ALE signal will load zero's into the counter and clear the MAX/MIN output. The MAX/MIN output is also used as the

counter enable to prevent the counter from counting further after attaining the count of 16. The circuit shown will function with the IDLE mode of the 8031. If the POWER DOWN mode is used, an inverter must be inserted in the ALE signal path.

Other controllers, not listed here, may also have power down modes which may function with these circuits. Any controller which has some sort of external indication when the power down mode has been entered may usually be used to place the PSD3XX in its low power mode also.

**Figure 9.
Digital Sleep
Circuit for
8031 Family**



**Power
Management
Techniques
In The PSD3XX
(Cont.)**

PAD Programming Techniques

The preceding section has described methods of using the power down capability of the PSD3XX with several microcontrollers. There are also techniques which may be utilized during programming of the device to further reduce power. These techniques can significantly reduce the power expended by the PSD3XX when it is in full operation.

The programmable logic section of the PSD3XX, called the PAD, provides much of its great flexibility and configurability. It is used to control the internal resources of the PSD3XX and can also be used to control external resources as well. The power use of the PAD varies greatly depending on how its product terms are programmed and used.

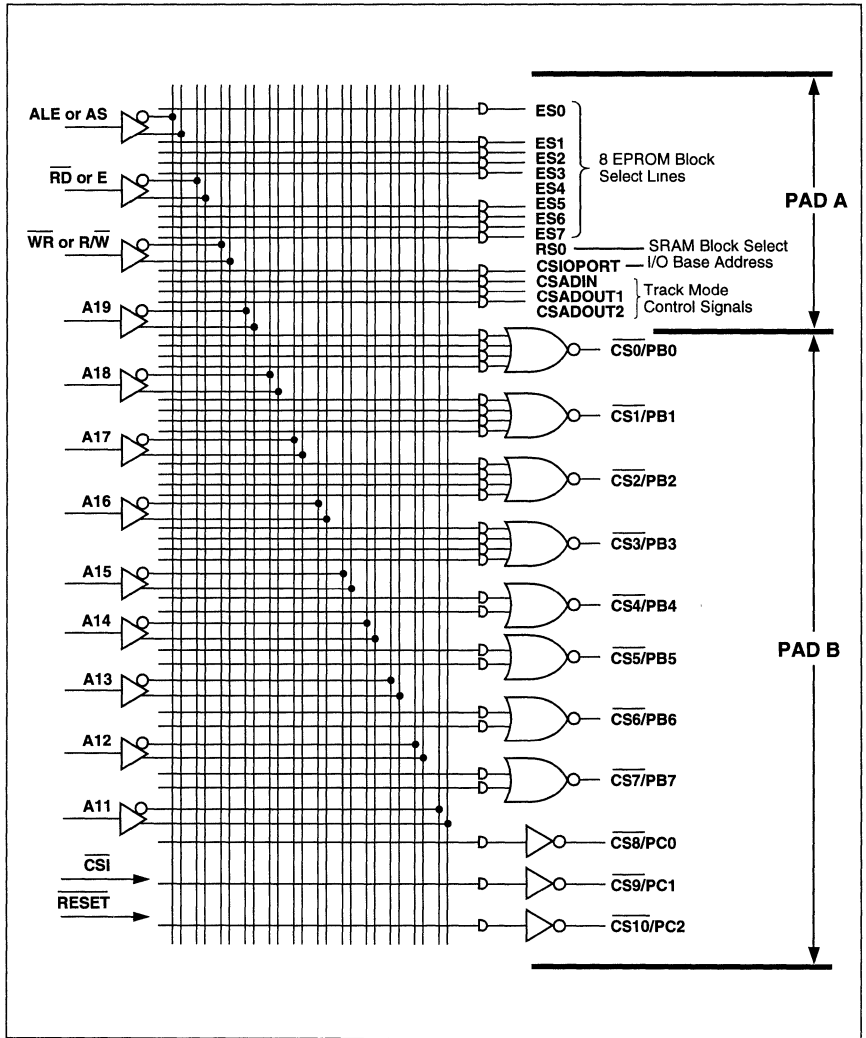
The PAD is illustrated in Figure 10. It is divided into two sections, called PAD A and PAD B. PAD A is responsible for generating the control and selection for the internal resources of the PSD3XX and utilizes 13 product terms to perform these functions. PAD B provides any external chip selection and logic replacement that is necessary for the system and has 27 product terms for this purpose. A single product term is functionally illustrated in Figure 11.

Each of the PAD inputs and its complement is available to each of the 40 product terms of the PAD. Each of these inputs is connected to an n-channel transistor which is used to connect the entire line to ground when the input is in the appropriate state. A high on the input to the gate causes the transistor to turn on. When the device is programmed, each of these transistors may be left in place or may be functionally removed (programmed out) from the circuit. If all of the transistors are programmed out, the line is left connected only to the pull-up resistor which makes it always high. Thus, the output of the inverter is always low. If an equation such as:

$$/CSx = In\#1 \cdot /In\#2$$

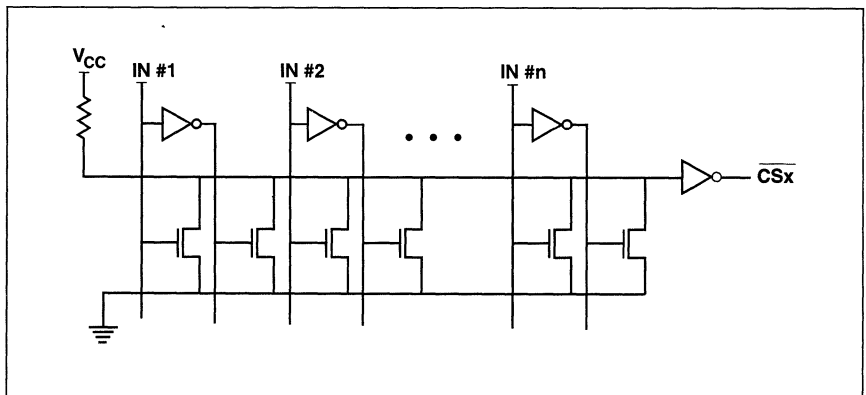
is programmed into the PAD, the output \overline{CSx} must be high except when In#1 is high and In#2 is low. Thus, all of the transistors are programmed out except the ones connected to In#1 and In#2. This means that unless In#1 is high and In#2 is low, there will always be at least one of the two remaining transistors turned on, which in turn results in the \overline{CSx} output being high. When the appropriate input condition is met, the remaining two transistors will turn off, which allows the output to become low.

**Figure 10.
PAD
Illustration**



1

**Figure 11.
Product
Term
Functionality**



**Power
Management
Techniques
In The PSD3XX
(Cont.)**

As can be seen in the figure, the product term expends very little power when all of the transistors are either programmed out or turned off. The only power used in this case is the result of the leakage current through the various off transistors, which is very low in CMOS technology. When one or more of the transistors is turned on, there will be current drawn through the pull-up resistor to ground. Therefore, the power used by a product term varies greatly according to the way it is programmed.

Experimental data has shown that a product term with all of the transistors programmed out draws approximately 380µA less current at room temperature and 5.0 V V_{CC} than a product term which has some active transistors. WSI's MAPLE software packages take advantage of this fact to reduce power as much as possible.

When the user intends to use some or all of the Port B pins as I/O signals, then they are not connected to the PAD in any way. Thus, the MAPLE software is free to program the unused PAD B product terms in any way. In MAPLE versions 4.03B and subsequent, the software automatically programs out all transistors in each unused product term, which can eliminate up to 24 product terms for Port B. This results in a power reduction of up to 9.1 mA.

If one or more of the Port C pins is programmed as an address or logic input, MAPLE is free again to program out all of the transistors in each unused PAD B product term dedicated to Port C. This can eliminate up to 3 additional product terms resulting in a power reduction of over 1 mA.

Finally, there are three product terms from PAD A which are dedicated to controlling the Port A Track Mode operation. If the Track Mode is not used in the application, these product terms may also be eliminated by MAPLE for a power reduction of over 1 mA.

The remaining ten product terms are the 8 EPROM select lines, the SRAM select line and the I/O port select line. These terms may not be eliminated by MAPLE without disrupting the operation of the device. But in a system which uses Port A and Port B as I/O or address outputs, and Port C as address or logic inputs, the total system power saving is 10.2 mA typical.

The same methods may also be used in non-multiplexed microcontroller applications. In this case, Port A and Port B may be used as microcontroller data input pins, depending on whether the controller is 8- or 16-bit. As in the earlier cases, if the ports are used as data input pins, they are not connected to the PAD which allows MAPLE to program out the appropriate product terms.

Again, MAPLE 4.03B or a subsequent revision must be used to obtain this capability. If your software is an older revision, contact your local WSI regional sales office for a free update.

EPROM Programming Techniques

Like the PAD, the EPROM in the PSD3XX uses varying amounts of power depending on how it is programmed. When programmed to a one, an EPROM bit draws more current than when programmed to a zero. Thus, for minimum power usage it is best to have the majority of the EPROM programmed to zeros.

Unfortunately, the contents of the EPROM are fixed by the program and data requirements of the system and thus cannot be easily optimized for power. However, the user can program all unused sections of the EPROM to zeros. This will not substantially cut the power used by the PSD3XX under normal operation when EPROM accesses are being performed, but it will reduce the power consumption during periods when there is not a valid address on the bus because these invalid addresses will often point to unused EPROM locations. When an EPROM location is currently addressed, it is expending power even if the \overline{RD} or \overline{PSEN} signals are not actually enabling an output. Therefore, it is best that unused EPROM locations be filled with zeros so that power is minimized during these periods of invalid addresses. It should be noted that all power figures used in this application note as well as those specified in the PSD3XX data sheet are based on an average of 50% "ones" and 50% "zeros" contained in the EPROM. An EPROM location programmed to "ones" will draw approximately 1.5 mA of additional current over an EPROM location programmed to "zeros".

Power Management Techniques In The PSD3XX (Cont.)

An even better way to help minimize power usage is to control the addresses which appear on the bus when there is no valid address being driven by the microcontroller. The least power expense will be when this unused address points to an area which has no PSD3XX resource mapped into it. This will result in no internal resource block receiving a chip select and thus the least amount of current will be drawn. The next best approach is to have the unused address point to an EPROM area containing zeros. The next lowest power would be to have the unused address point to an EPROM area containing something other than zeros. Finally, the highest power will occur when the unused address points to an SRAM location.

Since there is not much that can be done about the address that is appearing at the output of the microcontroller, the best that can be done is to know what address the controller will have active on its bus at various non-operational times and insure, if possible, that the PSD3XX's address map maps that address into a desired range of memory (preferably no memory at all). This will truly minimize the power expended by the PSD3XX during these times.

1

CMiser-Bit

The CMiser-Bit provides a programmable option for power-sensitive applications that require further reduction in power consumption. The CMiser-Bit (CMiser = 1) in the Maple portion of the PSD3XX system development software can be used to reduce power consumption. The CMiser-Bit turns off the EPROM blocks in the PSD3XX whenever the EPROM is not accessed, thereby reducing the active current consumed by the PSD3XX.

In the default mode, or if the PSD3XX is configured without programming the CMiser-Bit (CMiser = 0), the device operates at specified speed and power rating as specified in the A.C. and D.C. Characteristics.

However, if the CMiser-Bit is programmed (CMiser = 1), the device consumes even lower current, and is reflected in the data sheet. This mode has an added propagation delay in T5, T6, and T7 parameters in the A.C. Characteristics, and should be added to compute worst-case timing requirements in the application.

**Summing
It All Up**

After taking all of these factors into account, what kind of power use can you expect from the PSD3XX in your own system? As a guideline, we will calculate the typical power required of a PSD3XX installed in a hypothetical system. The requirements of this system are listed in Table 1.

Using this information, we can calculate the approximate typical power requirements of the PSD3XX. Before we can begin, we must know what the base power of the PSD3XX is under the voltage and temperature conditions specified. The base power of the PSD3XX is the power used by the PSD3XX when only the product terms which control the EPROM, SRAM and I/O ports are not programmed out (10 active product terms). The base power also assumes that no internal resources (EPROM, SRAM and I/O ports) are being currently accessed. The current drawn by the PSD3XX under these conditions has been determined experimentally to be 16 mA. To this current, we must add additional current for the other active product terms, SRAM access and EPROM access.

The system is requiring only four of the 11 available chip select outputs. Therefore, most of the PAD B product terms may be programmed out. To determine how many product terms we will be using, we must look at the equations for the four chip selects. Assume that the following equations are to be used:

$$/CS\#1 = /(A15 \cdot A14 \cdot RD + A13 \cdot A12 \cdot WR)$$

$$/CS\#2 = /(A18 + /A17)$$

$$/CS\#3 = /(A16 \cdot A18 + A17 \cdot ALE)$$

$$/CS\#4 = A17$$

In order to configure the system for the lowest power usage, we must be sure that we place these chip selects on the output pins which will require the minimum number of product terms to remain active. Since the maximum number of product terms required to generate the above equations is only two, there is no need to place these chip selects on Port B pin 0,1,2 or 3 since these pins each have four product terms. The lower power configuration would place these chip selects on Port B pin 4,5,6 and 7, where only two product terms will be drawing power for each chip select. One of the above chip selects, #4, actually requires only one product term, meaning that it could be placed on one of the Port C pins which have only one product term.

**Table 1.
Hypothetical
System
Requirements**

Characteristic	Specification
PSD3XX Operational Frequency	2 MHz
Port A	Address Output
Port B	4 Chip Select, 4 I/O
Port C	Logic inputs
CSI	Configured for Auto. Power Down
V _{CC}	5.0 V
Temperature	25°C
Standby duty cycle	60%
EPROM duty cycle	30%
SRAM duty cycle	10%

**Summing
It All Up
(Cont.)**

However, all of Port C is used in this case as logic inputs (A16, A17 and A18) and therefore cannot be used as chip selects. Since the rest of the Port pins are not used as PAD outputs, the MAPLE software will automatically program them out.

If we do configure the chip selects to output on PB[0:3], we must add 8 product terms to the 10 used in calculating the base power number. Using the current per product term of 380µA provided earlier, eight additional product terms result in an additional 3.0 mA of current.

Experimental data has shown that accessing the SRAM results in an additional current expense of 31 mA above the base current. Also, accessing the EPROM draws an additional 0.5 mA over the base current. The standby current has been measured at 50 µA. Finally, we must consider the additional current used by the frequency of operation. This is 3 mA per 1 MHz for a total of 6 mA, since the PSD3XX will be operating at 2 MHz. This provides us with all of the data that we need to calculate the total power usage of the PSD3XX in this system.

Table 2 can be used to calculate the EPROM access current, the SRAM access current and the standby current.

Now we must account for the duty cycle of the system to determine the total average power for the PSD3XX. In order to apply the duty cycle, we simply multiply each power component by its duty cycle and add them all together. The equation to perform this is given below:

$$\text{Total Current} = 0.6(i_{\text{SBY}}) + 0.3(i_{\text{EPROM}}) + 0.1(i_{\text{SRAM}})$$

where i_{SBY} is the standby current, i_{EPROM} is the active EPROM current and i_{SRAM} is the active SRAM current. Plugging in the numbers we developed earlier, the equation becomes:

$$\text{Total Current} = 0.6 (50 \mu\text{A}) + 0.3 (9 \text{ mA}) + 0.1(47 \text{ mA}) = \underline{7.4 \text{ mA}}$$

The average current drawn by the PSD3XX under the specified conditions of configuration, frequency and environment is therefore 7.4 mA. The peak typical current used by the PSD3XX is 54 mA while the SRAM is being accessed. The minimum current is 50 µA, drawn by the PSD3XX while it is in the Power Down mode. This compares very favorably with the typical current usage of a fully discrete solution.

1

**Table 2.
Summary of
PSD3XX Current
Usage In
Hypothetical
System**

PSD3XX Block	Current Used
Base Configuration (CMiser = ON)	9 mA
PAD (as configured)	3.0 mA
EPROM	0.5 mA
SRAM	31 mA
Frequency Component	6 mA
Standby Current	50 µA

Now, summarizing further, the total EPROM access current is:

$$\begin{aligned} &\text{Base Current} + \text{PAD Current} + \text{EPROM} \\ &\text{Current} + \text{Frequency Component} \\ &= 9 \text{ mA} + 3.0 \text{ mA} + 0.5 \text{ mA} + 6 \text{ mA} \\ &= \underline{18.5 \text{ mA}} \end{aligned}$$

The total SRAM access current is:

$$\begin{aligned} &\text{Base Current} + \text{PAD Current} + \text{SRAM} \\ &\text{Current} + \text{Frequency Component} \\ &= 9 \text{ mA} + 3.0 \text{ mA} + 31 \text{ mA} + 6 \text{ mA} \\ &= \underline{47.0 \text{ mA}} \end{aligned}$$



Typical vs. Maximum Current

The typical and maximum current numbers are both specified by most integrated circuit manufacturers. Many designers are unsure of what these parameters are and how they relate to the power which will actually be dissipated by the system. This is compounded by the configurability of the PSD3XX.

The maximum power numbers published in most product specifications are usually chosen as the number which will never be exceeded by the device under any circumstances, including variations in processing, V_{CC} and temperature. To truly be a maximum number, all three of these parameters must be at their worst cases simultaneously, which is quite unlikely. Therefore, power use will more likely follow the typical values when the system is actually running.

In the PSD3XX data sheet published by WSI, two current values are published for typical conditions and another two are published for worst case conditions. These two sets of numbers are used to specify current use in two different PSD3XX configurations. The lower numbers represent the current drawn by the PSD3XX while configured with 10 active product terms. To arrive at the maximum value for this configuration, we assume that the programming of the device has not changed, but we take the temperature, voltage and processing to their worst case

conditions. These numbers are generated again for the configuration of the PSD3XX which has all 40 product terms active. To determine the typical current drawn by the PSD3XX in your system, it is best to use the techniques presented in this application note. All of the typical current values used in this note are the result of careful experimentation, and should parallel very closely the values measured in your own system. To extrapolate the worst case current for your configuration from your calculated typical value, you must add about 50% to account for voltage, temperature and process variation.

When calculating the worst case current for your entire system it is usually best to use the typical current numbers for all of the components installed and then apply some margin to allow for worst case conditions. This is much more accurate than using the worst case parameters for each component since it is *extremely* unlikely that *all* of the components used are simultaneously at their worst case process parameters, though they may all be at worst case voltage and temperature. Usually 20% margin above the typical numbers will sufficiently cover the worst case for the entire system.

Table 3 summarizes the typical current numbers for the PSD3XX which can be used when calculating the current used in your own system.

Table 3. Summary of PSD3XX Typical Current Usage

Base Current (10 product terms, SRAM and EPROM Unselected and CMiser = ON)	9 mA
Additional Current per Product Term	0.38 mA
Additional Current for SRAM Access	31 mA
Additional Current for EPROM Access	0.5 mA
Additional Current for Frequency Effects	3 mA/MHz
Additional Current for Voltage > 5V	0.85 mA/0.1V
Standby Current	50 μ A

Table 4.
PSD3XXL
Power
Consumption

Using the same example described in Table 1, a PSD3XXL operating at 3 volts and 1 MHz will exhibit the following values:

PSD3XXL Block	Current Used
Base Configuration (CMiser ON)	2 mA
PAD	$0.19 \times 10 = 1.9$ mA
EPROM	0.25 mA
SRAM	13 mA
Frequency Component	2 mA
Standby Current	1 μ A

EPROM Access Current = $2 + 1.9 + 0.25 + 2 = 6.15$ mA

SRAM Access Current = $2 + 1.9 + 13 + 2 = 18.9$ mA

Total Current = $0.6 \times 1 \mu\text{A} + 0.3 \times 6.15 \text{ mA} + 0.1 \times 18.9 \text{ mA} = 3.74$ mA

Conclusion

The PSD3XX and PSD3XXL are very important devices in the design of compact, low-power systems. It provides a cost effective minimum part count solution for a typical microcontroller system. It also provides a very low power solution for those designs which are handheld and/or battery operated. As the PSD3XX family grows and evolves, more innovations will

be presented in terms of integration and power usage. The new low power PSD3XX family will be introduced soon, providing the designer with an even lower power solution. Until then, use of the techniques described in this note will provide a minimum power solution for your microcontroller system.



Programmable Peripheral Application Note 017 Track Mode Implementation of PSD3XX

By Ravi Kumar

Introduction

Resource sharing is becoming an inevitable issue in ever growing and complex microcontroller applications. Increased throughput and efficiency requirements of complex data acquisition systems such as automotive electronics, high speed/high density disk drive electronics, cellular phones etc., are driving microcontrollers to share resources. This results in increased discrete component count, complexity in design and timing issues and thereby

lengthens the time it takes to get to market. Using a Dual-port RAM is an expensive solution. A lower cost solution would be to use a PSD3XX and SRAM in "TRACK MODE". The PSD3XX also replaces the glue logic, EPROM, SRAM and I/O port requirements of the microcontroller based system. The following application note explains the implementation of the PSD3XX's "TRACK MODE" using Intel's 80C31 microcontrollers.

1

Bus Sharing

In a master/slave configuration of microcontrollers the typical bus sharing sequence is:

- Slave processor sends a request to the master processor to tri-state its bus in order to access the resource they share.
- Master processor acknowledges the slave processor's request and tri-states its own bus connected to the shared resource.
- Then the slave processor proceeds to write/read to/from the shared resource.
- After the completion of writing/reading data to/from the shared resource, the slave processor relieves the control of the shared resource.

- Then the master processor resumes the possession of the bus connected to the shared resource and continues its interaction with it until it receives a request from the slave processor to release the bus.

Although the master/slave processors tri-state their bus connected to the shared resource, by using PSD3XX devices we can avoid the situation where the processors wait. Therefore the processors continue to attend to other chores when the shared resource is not available. This application note explains how this can be achieved.

PSD3XX Architecture Related to TRACK Mode of Operation

Tracking of Address/Data inputs to a PSD3XX is possible only through port A and only for microcontrollers with a multiplexed bus. The default configuration of port A is I/O. Alternately, each bit of port A can be configured as a lower order latched address bus bit. Another mode of port A 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 bidirectional buffer. The direction is controlled by using the input signals ALE, RD/E/DS, WE/V_{PP} or R/W, and the internal PAD outputs CSADOUT1, CSADOUT2 and CSADIN (see Figure 1).

CSADOUT1 is generated when the microprocessor is accessing a "tracked" address. It is generated from a single product term involving the address inputs and ALE. When an address generated by the microcontroller is within the block



**PSD
Architecture
Related to
Track Mode
of Operation
(Cont.)**

specified by the user for track mode and the ALE is active, CSADOUT1 becomes active, transferring the address and outputting it from port A. Carefully check the generation of CSADOUT1 and ensure that it is stable during the ALE pulse; i.e. signals in the product term involved in generating CSADOUT1 should not be those addresses which change frequently; instead they should be stable I/O signals.

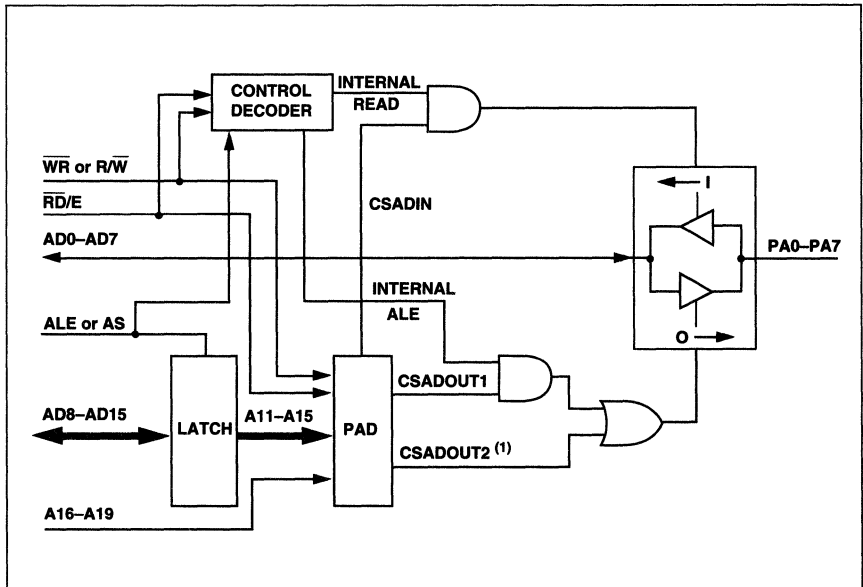
CSADOUT2 is generated when the microcontroller is performing a write operation to a tracked address. It also has one product term involving the address inputs and WR (Intel mode) or R/W and E or DS (Motorola mode). When the microcontroller performs a write operation to the appropriate address, CSADOUT2 is

generated, transferring the data and outputting it from port A.

CSADIN is generated when the microcontroller is attempting to read data from Port A in the track mode. It is generated from one product term involving the address inputs and the RD strobe (Intel mode) or R/W and or DS (Motorola mode). This enables the user to configure the address range in which the data is to be read from Port A.

In this operational mode, port A is tri-stated when none of the above conditions exist.

**Figure 1.
Port A
Track Mode**



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

Track Mode of PSD3XX Using Intel's 80C31s in Master/Slave Configurations

In this configuration two PSD3XX's are used with two Intel's 80C31s. In figure 2, a common SRAM CD6116 and a '373 Latch are the shared resources of the Master/Slave configuration of these two microcontrollers.

Referring to the flowchart in figure 3, under normal operation, the master processor and the master PSD3XX are in control of the shared LATCH and SRAM, i.e. pins P1.0 and P1.1 of the master/slave processors are all HIGH. The slave processor, whenever it needs to access the shared LATCH and SRAM, requests the control of the address/data bus from the master processor by generating a HOLD signal from the slave 80C31's P1.0 as output LOW. The master 80C31 takes this HOLD signal on to its input pin P1.0 and INTO pin as an interrupt, processes it and acknowledges by returning HLDACK on P1.1 (LOW), i.e. hold acknowledge signal to slave processor. Meanwhile the master PSD3XX senses that both P1.0 and P1.1 pins of the master processor are LOW and tri-states the ports A,B and C of the PSD3XX, thereby disabling its control over the bus connected to the shared LATCH and SRAM. In order for a successful interaction between master and slave processors, care should be taken in the master processor to properly mask the interrupt generated on INTO.

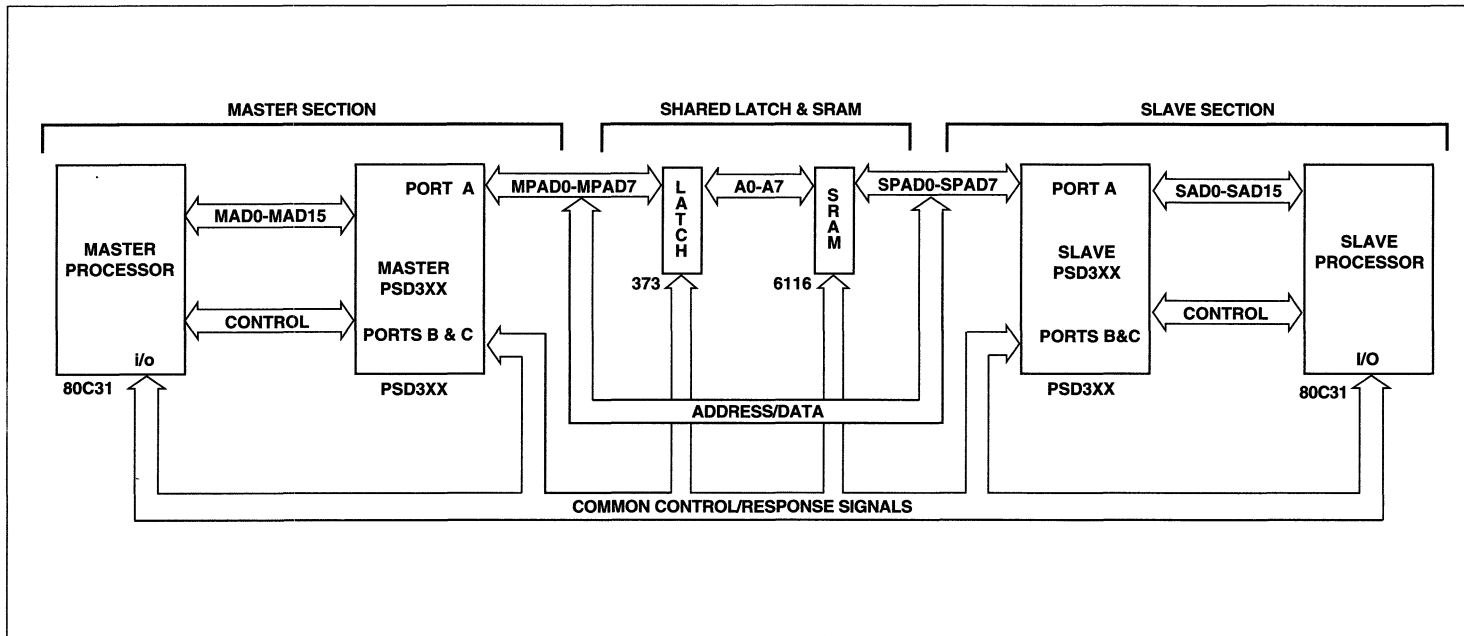
The HLDACK signal coming from the master is fed into P1.1 of the slave 80C31 processor, which processes HLDACK and takes control of the bus connected to the shared LATCH and the SRAM. The slave PSD3XX senses that both P1.0 and P1.1 of the slave processor are LOW and starts tracking the address and data flowing into and out of the CD6116 SRAM. When the slave processor is successfully accessing the shared SRAM, the master processor attends to other chores while polling its input P1.0 (turned HIGH if the slave is done). The PAD in the PSD3XX with the master processor tri-states its PORT A and port B during slave processor ('373 Latch – SRAM) activity and vice-versa. The PADs in the master/slave processors generate the necessary ALE, \overline{RD} and \overline{WR} signals necessary to write into or read out of the shared SRAM.

The common SRAM (CD6116) is accessed by both microcontrollers in byte mode. All the control lines on port B of both PSDs are configured as open drain drivers. Both microcontrollers can access the latch and the SRAM without conflict because the PAD equations controlling the port B on both PSD3XXs are based on HOLD and HLDACK signals generated by the master/slave processors. The HOLD signal is connected to port C pin 0 or A16 and the HLDACK signal is connected to port C pin 1 or A17 of both master and slave PSD3XXs and also to INTO (master), P1.0 and P1.1 of both master/slave processors respectively. Refer to figure 4 for specific details.

As long as no HOLD request comes from the slave processor, HLDACK generated by the master processor remains HIGH which tri-states the slave processor's PSD3XX's port B and enables the master processor's PSD3XX's port B control to the '373 latch and the SRAM (CD6116). Please refer to the schematic in figure 4, Master PSD equations in figure 5 and Slave PSD equations in figure 6 for specific details.

The CD6116 is a 2K x 8 SRAM and the PSD3XX's port A address lines (A0–A7) are connected to the SRAM's address lines. This implies that only 256 bytes of the shared SRAM are accessible and these bytes roll over for every 256 bytes in the higher address ranges.

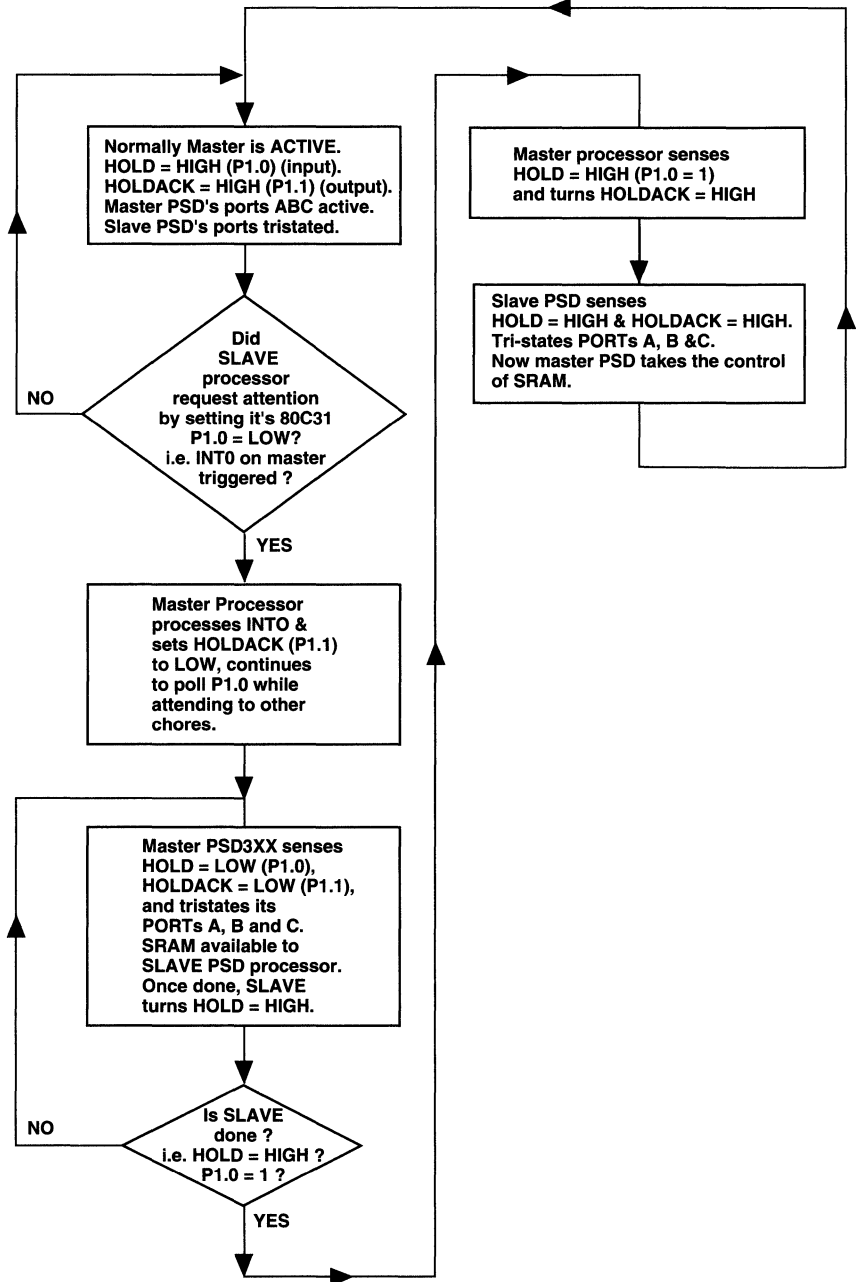
Figure 2.
Block Diagram
of PSD3XX
Track Mode
Implementation



- Legend:**
- MAD0 - MAD15 = Master Processor's Address/Data Bus.
 - MPAD0 - MPAD7 = Master PSD3XX's Address/Data from Port A.
 - SAD0 - SAD15 = Slave Processor's Address/Data Bus.
 - SPAD0 - SPAD7 = Slave PSD3XX's Address/Data from Port A

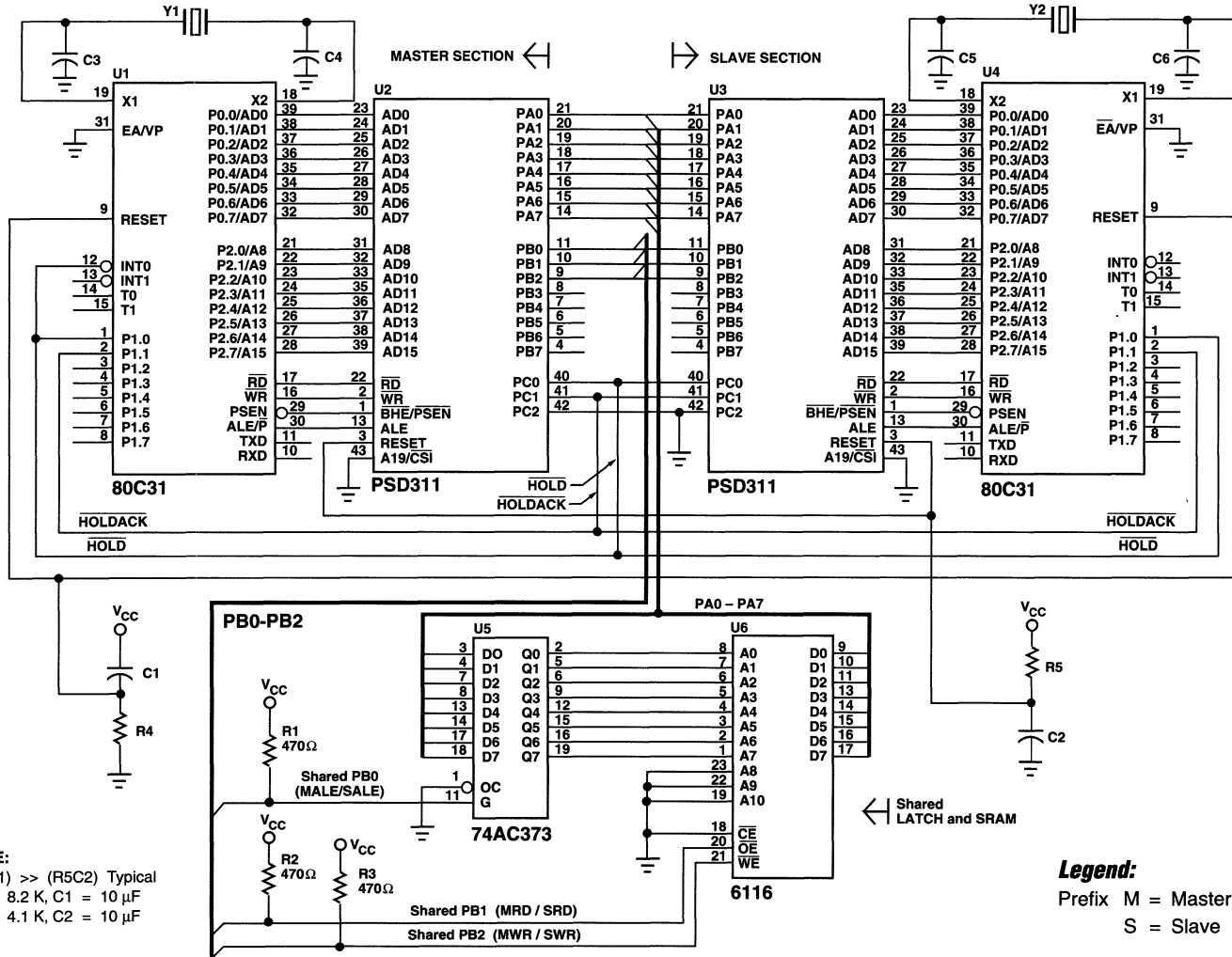


Figure 3.
Flow Chart of
Track Mode
Implementation



1

Figure 4. Track Mode Schematic



NOTE:
(R4C1) >> (R5C2) Typical
R4 = 8.2 K, C1 = 10 μF
R5 = 4.1 K, C2 = 10 μF

Legend:
Prefix M = Master
S = Slave

Figure 5. Master PSD Equations

ALIASES

```
/CS8/A16 = HOLD
/CS9/A17 = HOLDACK
/CS0 = MALE
/CS1 = MRD
/CS2 = MWR
```

GLOBAL CONFIGURATION

```
Address/Data Mode:           MX
Data Bus Size:               8
CSI/A19:                     CSI
Reset Polarity:              LO
ALE Polarity:                 HI
WRD/RWE:                     WRD
A16-A19 Transparent or Latched by ALE: T
Using different READ strobes for SRAM and EPROM: Y
Separate SRAM and EPROM Address spaces: N
```

PORT A Address/Data Direction Control

$CSADIN = HOLDACK * HOLD * A15 * /A14 * A13 * /A12$

$CSADOUT1 = HOLDACK * HOLD$

$CSADOUT2 = HOLDACK * HOLD * A15 * /A14 * A13 * /A12 * RD * /WR$

PORT B CONFIGURATION

Bit No.	CS/IO.	CMOS/OD.
0	CS0	OD
1	CS1	OD
2	CS2	OD
3	CS3	OD
4	CS4	OD
5	CS5	OD
6	CS6	OD
7	CS7	OD

CHIP SELECT EQUATIONS

$MALE = /(HOLDACK * HOLD * / ALE)$

$MRD = /(HOLDACK * HOLD * A15 * /A14 * A13 * /A12 * / RD)$

$MWR = /(HOLDACK * HOLD * A15 * /A14 * A13 * /A12 * / WR)$

**Figure 5.
Master PSD
Equations
(Cont.)**

PORT C CONFIGURATION

Bit No.	CS/Ai.
0	A16
1	A17
2	A18

ADDRESS MAP

Name	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	EPROM	EPROM	File
	19	18	17	16	15	14	13	12	11	STRT	STOP	START	STOP	
ES0	N									N				
ES1	N									N				
ES2	N									N				
ES3	N									N				
ES4	N									N				
ES5	N									N				
ES6	N									N				
ES7	N									N				
RS0	N													
CSP	N													

Figure 6. Slave PSD Equations

ALIASES

```

/CS8/A16 = HOLD
/CS9/A17 = HOLDACK
/CS0 = SALE
/CS1 = SRD
/CS2 = SWR

```

GLOBAL CONFIGURATION

```

Address/Data Mode:           MX
Data Bus Size:              8
CSI/A19:                   CSI
Reset Polarity:             LO
ALE Polarity:               HI
WRD/RWE:                   WRD
A16-A19 Transparent or Latched by ALE: L
Using different READ strobes for SRAM and EPROM: Y
Separate SRAM and EPROM Address spaces: N

```

PORT A Address/Data Direction Control

```

CSADIN = /HOLDACK * /HOLD * A15 * /A14 * A13 * A12
CSADOUT1 = /HOLDACK * /HOLD
CSADOUT2 = /HOLDACK * /HOLD * A15 * /A14 * A13 * A12 * RD * /WR

```

PORT B CONFIGURATION

Bit No.	CS/IO.	CMOS/OD.
0	CS0	OD
1	CS1	OD
2	CS2	OD
3	CS3	OD
4	CS4	OD
5	CS5	OD
6	CS6	OD
7	CS7	OD

CHIP SELECT EQUATIONS

```

SALE = /( /HOLDACK * /HOLD * /ALE )
SRD = /( /HOLDACK * /HOLD * A15 * /A14 * A13 * /A12 * /RD )
SWR = /( /HOLDACK * /HOLD * A15 * /A14 * A13 * /A12 * /WR )

```

**Figure 6.
Slave PSD
Equations
(Cont.)**

PORT C CONFIGURATION

Bit No.	CS/Ai.
0	A16
1	A17
2	A18

ADDRESS MAP

Name	A	A	A	A	A	A	A	A	A	SEGMENT	SEGMENT	EPROM	EPROM	File
	19	18	17	16	15	14	13	12	11	STRT	STOP	START	STOP	
ES0	N									N				
ES1	N									N				
ES2	N									N				
ES3	N									N				
ES4	N									N				
ES5	N									N				
ES6	N									N				
ES7	N									N				
RS0	N													
CSP	N													

Conclusion

This application note clearly shows how PSD3XX's track mode could be best utilized in resource sharing configurations with microcontrollers. Using PSD3XXs in this kind of a design provides the following significant advantages to the designer:

- Real estate savings on-board (reduced chip count).
- Cost savings

- Shorter time to market
- Power savings
- No additional glue logic.

It also offers flexibility in redesigning efforts by simply changing the configuration of the PSD3XXs.



Programmable Peripheral Application Note 018

Security of Design in the PSD3XX

By Oudi Moran

Introduction

The PSD3XX is a family of field programmable and UV erasable microcontroller peripherals that have the ability to interface to virtually any microcontroller without the need for external glue logic.

Any PSD3XX family member is a complete microcontroller peripheral solution with Memory (EPROM, SRAM), Logic, I/O Ports and a Security bit on chip.

In today's competitive business environment, where the cost of the product and its quick introduction to market are the most important factors for success, some companies tend to copy a competitor's design. By doing so, they can save development time which can reduce their engineering cost and eventually reduce the product's price and its introduction time to the market.

This is true mainly for the consumer and commodity product markets where microcontrollers are widely used. The PSD3XX, as the primary microcontroller peripheral, contains all the important code and architectural data that a potential competitor may want to copy.

Since the PSD3XX is a field programmable device, its contents may be read by an I.C. programmer, decompiled and copied by a competitor.

Obviously, it is an undesirable situation for the EPROM, PAD and configuration data of the PSD3XX to fall into the hands of a competitor. To prevent this, the PSD3XX device implements a security "fuse" or programmable bit feature to protect its contents from unauthorized access and use by a competitor.

Uploading the programmed data from EPROM, PAD, ACR and NVM port configuration sections of a secured PSD3XX device is disabled by the security bit (if turned ON). The RAM of the programmer (after trying to upload a secured PSD3XX device) will contain invalid random data.

A secured PSD3XX device will function properly in the system – the microcontroller will be able to access the EPROM, SRAM, PAD and the I/O ports but any attempt to read or verify the contents of a secured PSD3XX by external hardware will fail.

Use of the Security Bit

PSD3XX devices contain non-volatile configuration bits to enable the user to set and configure the device to the proper operational mode. The configuration bits will configure the device to interface successfully with the microcontroller and also configure the PSD3XX I/O Ports. The configuration bits are programmed during the programming phase and cannot be accessed in operational mode.

During programming the configuration bits are programmed as two separate sections:

- 1) The ACR section of the PSD3XX device contains global configuration bits for proper microcontroller interface. The security bit resides as an individual configuration bit in the ACR section of the device.

- 2) The NVM section of the PSD3XX device contains port configuration bits for proper set up of Ports A, B and C.

PSD3XX devices use the security bit to prevent unauthorized access to the configuration data inside. Since the security bit is part of the ACR global configuration bits section, it can be programmed in the same manner as all other configuration bits.

All ACR and NVM configuration bits of the PSD3XX are non-volatile, so their contents will not be erased or corrupted during the power down mode of the device (when the PSD3XX is deselected with $CS/A19 = \text{High}$) or during power down when V_{CC} is removed.



Use of the Security Bit (Cont.)

The security configuration bit is user programmable and UV erasable as well, so a secured part can be erased completely and be reprogrammed (only if the device is in a windowed package).

Setting the security bit will lock all the contents of the PAD, ACR global configuration bits, and NVM port configuration bits. By setting the security bit the device cannot be entered into Initialization and Override mode (resets the device and enters it to a known default configuration before activating the individual read mode for each section). Any attempt afterwards to enter the device to DIRECT mode for uploading or programming will fail. Setting the security bit prevents a programmer from directly accessing the various sections of the device.

Even though the EPROM, SRAM and I/O port contents are not directly disabled by

setting the security bit, it is impossible to read them by using external equipment (except by the microcontroller in the system where the PSD3XX designed in). This is because the external equipment will lack information about the address mapping of the eight EPROM blocks, SRAM and I/O ports in the memory map of the microcontroller and the unknown status of the global and I/O port configuration bits.

Even if an unauthorized user figures out the configuration of the part by knowing what microcontroller is interfaced (ALE polarity, what type of read and write signals, etc.) and gets data out of the PSD3XX (after applying address and control signals to the device), the user will have no idea where it came from: EPROM, SRAM, I/O Port Register, Page Register, etc. This effectively renders the data useless.

Setting the Security Bit

The security configuration bit is called CSECURITY.

If CSECURITY = 0, it means security is off (security bit is not set and its value will be '1' in the object file).

If CSECURITY = 1, it means security is on (security bit is set and its value will be '0' in the object file).

Setting the security bit and activating the security mode can be done in two different ways:

- 1) By turning security ON in the configuration menu of Maple development software.
- 2) By setting the security in the programming software (done after the device is fully programmed and verified).

Using Maple development software to turn security ON gives the security bit the value '0', and will integrate it in one of the ACR

addresses of the object file created after compilation. (See Security Bit File Location section of this document).

If Setting of the security bit is done in the programming software (Third party programming software or WSI Mappro programming software), the user should program and verify the device using a Maple generated object file (with security option OFF) and then set the security ON by using a separate programming software command.

Some third party programmer manufacturer's software will load the Maple generated object file but mask the security bit before programming the device. In that case the user will have to set the security bit (if necessary) by using a separate command in the programming software menu.

Security Bit File Location

The object file created by compilation with Maple software is an Intel Intelec format, compatible file.

The programming algorithm defines the address scrambling that translates the file addresses to device addresses (the address that the device “sees” on its address pins during programming). By looking at a screen dump or a hard copy of the object file the user can determine the status of the security bit.

The security bit of the PSD301/311 resides in data bit #1 of file address 81D3h. This address contains three configuration bits that reside in data bits 0 – 2, so this address in the file can have any value between 0 and 7.

If this address has a value X1X (where X can be either 0 or 1), the security bit is off ('1' value means an unprogrammed bit) and CSECURITY = 0 (displayed by Mappro WSI programmer interface software as SECA = 0).

If this address has a value X0X, the security bit is on and CSECURITY = 1 (displayed

by Mappro WSI programmer interface software as SECA = 1).

The security bit of PSD302/312 resides in data bit #1 of file address 10253h. This address contains three configuration bits that reside in data bits 0 – 3 (bit 3 is reserved for future usage). This address can have any value between 0 and F. If this address has a value XX1X (where X can be either 0 or 1), the security bit is OFF ('1' value means an unprogrammed bit) and CSECURITY = 0 (displayed by Mappro WSI programmer interface software as SECA = 0). If this address has a value XX0X, the security bit is ON and CSECURITY = 1 (displayed by Mappro WSI programmer interface software as SECA = 1).

If users do not want to look for the security bit status in the object file, they can call MAPPRO programming software from the main menu of MAPLE, Load the RAM with the object file and Display the ACR configuration bits status on the screen.

The value of SECA will indicate the status of the security bit (SECA = 0 means security is OFF, SECA = 1 means security is ON).

Summary

The PSD3XX family of programmable microcontroller peripheral devices provides security of design not readily available in conventional PLDs and EPROMs.

Though not entirely fool-proof, the security bit feature helps make it more cost effective for competitors to design their own hardware instead of trying to copy systems that already exist.



Programmable Peripheral Application Note 019

The PSD311 Simplifies an Eight Wire Cable Tester Design and Increases Flexibility

in the Process — By Timothy E. Dunavin, Antec — Anixter Mfg.
and Karen S. Spesard, WSI

1

Abstract

With the ever increasing complexity of wiring networks and cables to match a wide variety of computer and telecommunication systems, a means of testing them becomes a necessity. The wire tester design described below is a simple yet effective

design which uses the Motorola 68HC11 and WSI PSD311 pair to create a system that insures 8-wire cables are wired properly, and at the same time offers a substantial increase in design flexibility over alternative hardware solutions.

Introduction

More and more microcontroller and microprocessor designers are trying to design integrated core-based systems with the intention of being able to easily configure their systems to fit a wide variety of product applications. The problem is that when these applications require new or changing features such as expanding I/Os or address maps, they may find their designs are not flexible enough to accommodate the new requirements, forcing a lengthy and expensive redesign anyway.

A solution to this problem is to design in user-configurable programmable peripheral products which are flexible enough to accommodate future design revisions without the need for board relayout. The PSD3XX family from WSI, Inc., fits this profile exactly in that the products can be tailored to a specific application and then

can be re-configured for other applications using the same core design. Also, the PSD3XX product family can enhance microcontroller-based systems in other ways. For instance, it can improve system integration resulting in lower system costs, and it can significantly shorten time to market resulting in increased revenues and profits.

In the cable tester system in which the PSD311 was used with the 68HC11, the PSD311 integrates address decoding, latches, 32K x 8 EPROM, and 2K x 8 SRAM all into a one-chip user-configurable microcontroller peripheral. It also replaces the two ports lost by the 68HC11 to extend program and data memory outside the MCU with two additional configurable 8-bit I/O ports, and adds a third 3-bit port, while easily enabling still further port expansion.

The Cable Tester System Design

The cable tester described below operates by sending a known bit pattern through the cable under test and checking the bit pattern at the other end. The hardware configuration utilized to achieve this function is shown in Figure 1.

Note that there are very few components overall in the design. The core contains just the 68HC11 microcontroller from Motorola, the PSD311 Programmable Peripheral with Memory from WSI and a few other key components including a keypad, LCD display, and an optional RS232 communications device.

Also note that the interconnections between the 68HC11 and PSD311 are direct and require no "glue logic". That means that no external latches are needed to demultiplex the multiplexed address and data bus from the 68HC11. And, no other external logic is needed to generate the address mapping for the on-board EPROM and SRAM and to select external peripherals, or create the control signal interface. The PSD311 already incorporates these features internally, thereby simplifying the design considerably. In fact, the PSD311's architecture, as shown in Figure 2, specifically includes 32K x 8 mappable EPROM for program

Figure 1. PSD311/68HC11 Implementation in the Cable Tester Design

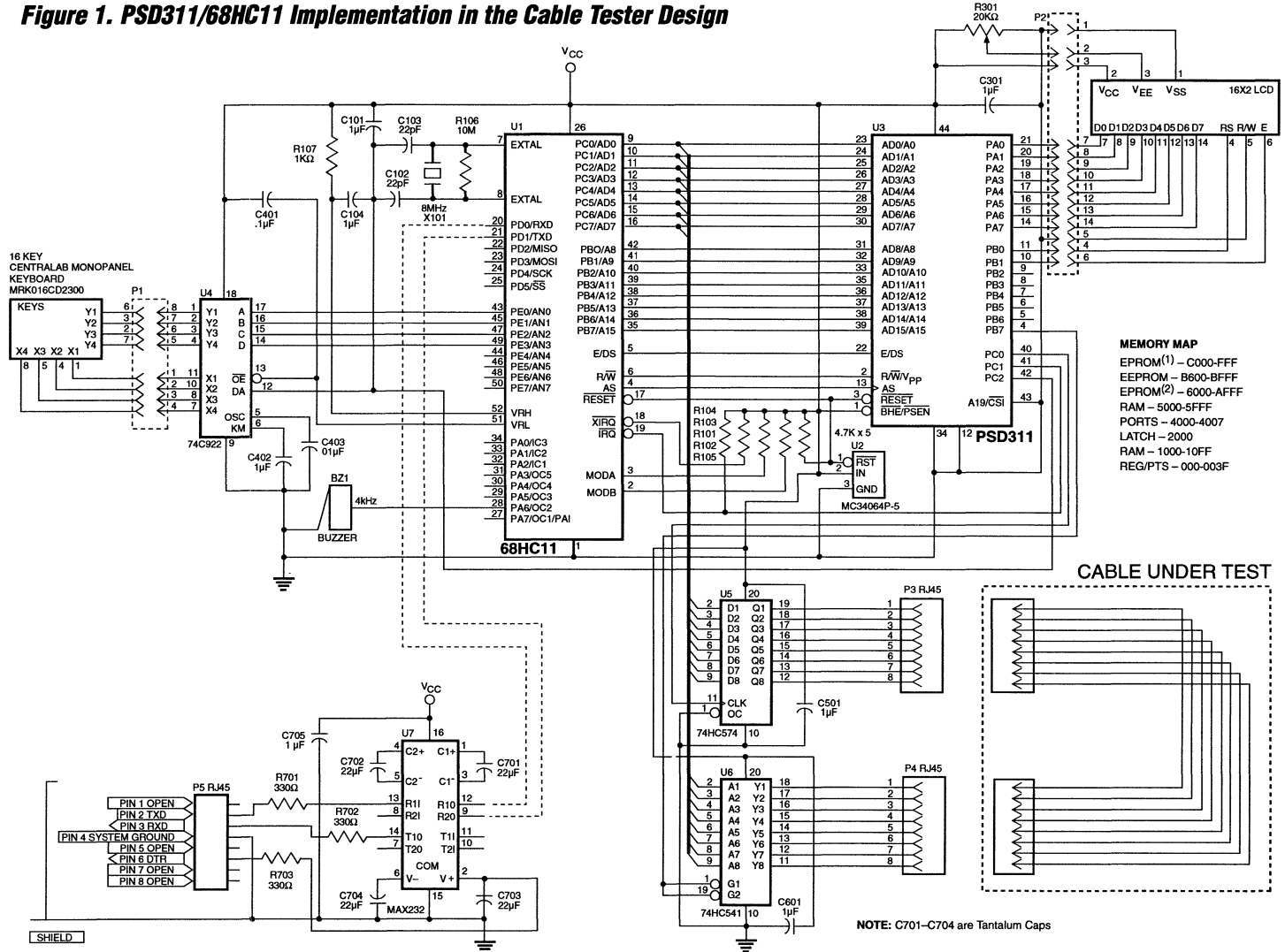
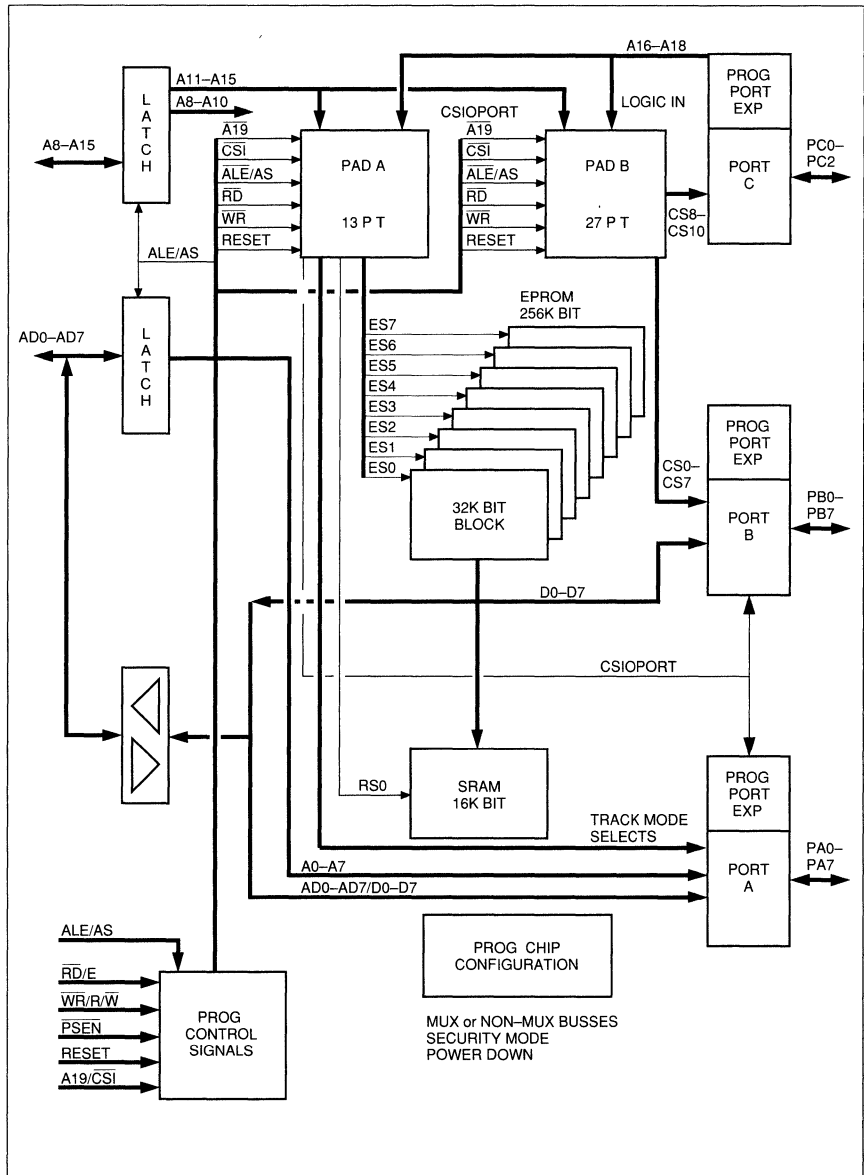


Figure 2.
PSD311
Architecture



1

The Cable Tester System Design (Cont.)

storage, 2K x 8 mappable SRAM for data storage (or 16K x 16 EPROM and 1K x 16 SRAM, if using the similar PSD301 configured to interface to x16 micros) three highly configurable I/O ports, a programmable address decoder, and chip select logic.

In this design, the reconstructed port space of the PSD311 is used to add a keypad

and an LCD display to the system, as well as additional output control and input lines with an 8-bit latch and an 8-bit buffer/line driver. Besides these components, the completed cable tester design also includes an undervoltage sensing circuit for generating a reset signal and an encoder for interfacing to the keypad.

Interfacing To The PSD311

Not only does the PSD311 interface to the 68HC11 simply and directly because of its internal latches and programmable control signals – as it does with any 8-bit microcontroller – it also facilitates easy interfacing to other components. (The PSD301 interfaces to any 8- or 16-bit microcontroller.) This is possible because of its three I/O ports and the Programmable Address Decoder (PAD) which offer unsurpassed flexibility. The PAD block diagram is shown in Figure 3.

For instance, the no “glue-logic” interface of the keypad in the system is accomplished by using a 74C922 encoder in conjunction with the PAD section of the PSD311. The PAD is useful because the Data Available (DA) line of the 74C922 is a logic “1” when a key is pressed, and the signal must be inverted before it reaches the /IRQ input of the 68HC11. Connecting the encoder’s DA line to the PSD311’s PC2 pin and configuring it to be a general-purpose logic input enables the signal to be inverted inside the PAD. The inverted signal is then “outputted” on PC1 which is configured as a chip select and routed to /IRQ. (See Port C Configuration and Chip Select Equation in Appendix A.) This simple internal manipulation inside the PSD311 helps reduce the number of components in the system. By connecting the 74C922 outputs directly to PE0-PE3 on the 68HC11, reading of the data is straightforward.

The display used in the system is a 16 character by 2 line dot matrix LCD module. The interface to the LCD display is handled by mapping the data bus directly to Port A of the PSD311, which is configured pin-by-pin to be general-purpose I/O. The control logic for the LCD is handled through two pins on Port B: PB0 and PB1, which are also configured to be general-purpose I/O. (See Ports A and B Configuration in Appendix A.) With the display used as a “WOM” (Write Only Memory), its R/W

line is tied to ground to free an I/O pin of the PSD311 for other purposes. To free up Port A completely on the PSD311, an alternative approach would have been to connect the LCD directly to the 68HC11.

To expand the I/O capabilities of the system further, two port pins from the PSD311 are used with a 74HC574 and a 74HC541 to create 8 additional inputs and 8 additional latched outputs, both at the same address. (This is shown in Figure 1.) The PSD311’s chip select outputs from ports B and C are derived from the addresses, DS strobe, and R/W signal available as inputs into the PAD. These chip selects will enable data to be latched to the outputs or enable input data onto the extended address/data bus from the outside world, imitating the capability of a PIA.

The chip select equation for the output latch, 74HC574, is decoded from the upper address byte, the DS/E signal, and the active low R/W signal as follows:

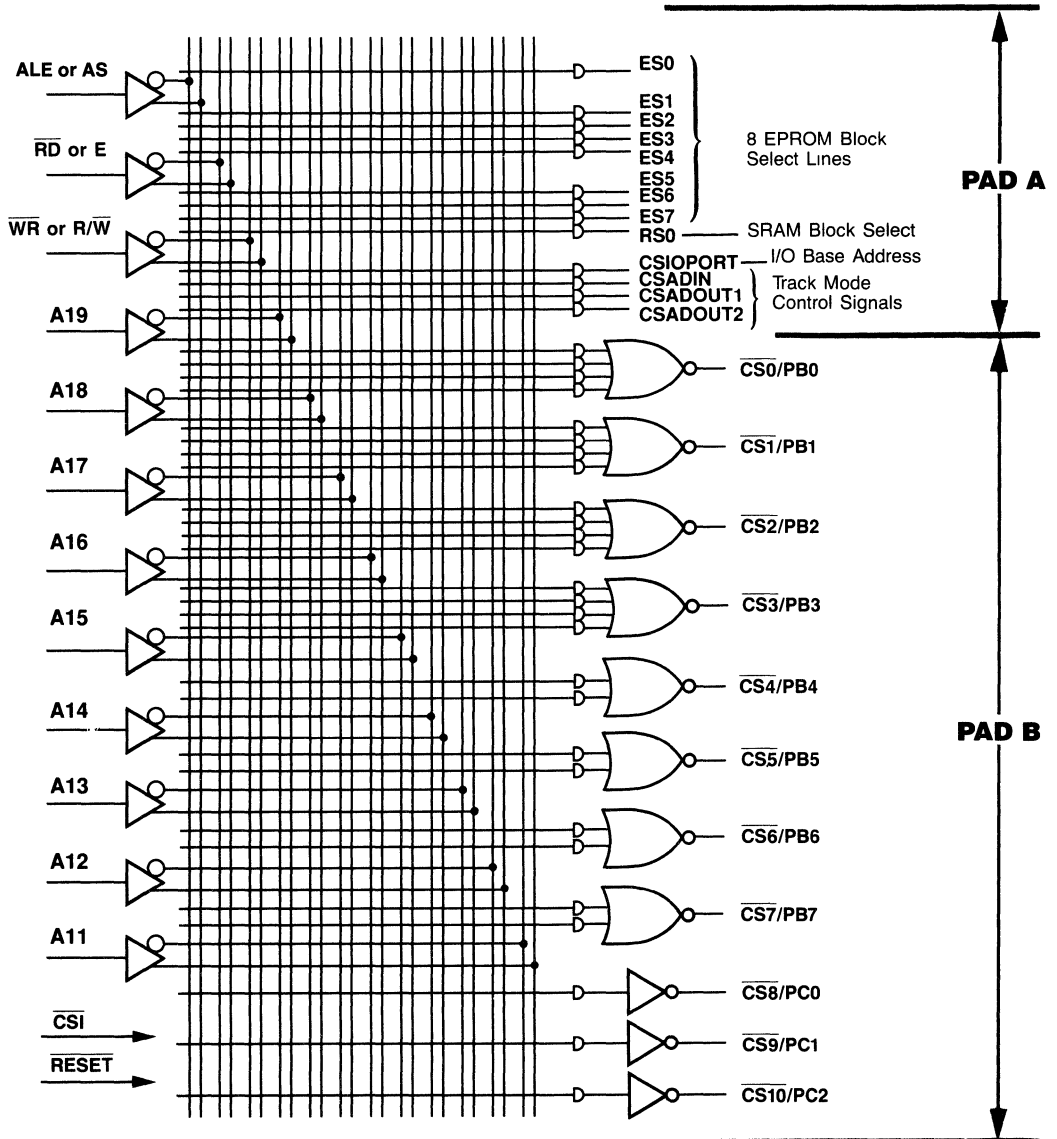
$$/CS8 = /A15 \cdot /A14 \cdot A13 \cdot /A12 \cdot DS \cdot /R/W.$$

The resulting latched address is \$2000H with DS = 1 and R/W = 0. The chip select equation for the input driver, 74HC541, is the same, because the address is the same (\$2000H), except that R/W is active high. So, this equation becomes:

$$/CS9 = /A15 \cdot /A14 \cdot A13 \cdot /A12 \cdot DS \cdot R/W.$$

The PSD311 simplifies the interface to the program and data memory, external peripherals, and I/O ports in the system by integrating the address decoder internally. This is illustrated with the direct interconnection between the microcontroller and other peripherals and the PSD311, without the need for a PLD or other logic.

Figure 3.
Programmable
Address
Decoder
Block Diagram



**Interfacing
To The PSD311
(Cont.)**

The PAD enables the 8 blocks of 4K bytes EPROM (256K bits) to be located anywhere within the available address space – in this case, the address space of the 68HC11 is 64K bytes. So, the EPROM memory is split into two segments of 16K bytes EPROM each, separated by the 512 bytes of the internal E2PROM on the 68HC11. This means that the first 4 EPROM blocks are mapped contiguously, as well as the last 4 EPROM blocks. Here, the program memory (6000H-9FFFH: EPROM2, and C000H-FFFFH: EPROM1) is allocated to the upper portion of address space.

The data or SRAM memory, on the other hand, is allocated to the lower portion of address space and is partitioned into

two segments: one segment containing the SRAM internal to the 68HC11 (256 bytes) and the other containing the SRAM internal to the PSD311 (2K bytes). The SRAM in the PSD311 is mapped via the address decoder to location 5000H-5FFFH, respectively.

Data direction and data registers of the PSD311's two ports are paired and accessed via an offset from a configurable I/O port mapped base address, such as 4000H in this cable tester design. This enables 16-bit data instructions to access the two I/O ports together, which in turn reduces both the Load and Store times during program execution.

**Benefits
of the PSD311
Usage in
System**

Board layout of the cable tester design was greatly simplified with the PSD311. In fact, when pin 1 of the PSD311 is oriented 180 degrees from pin 1 of the 68HC11 in the PLCC package, port B of the 68HC11 is directly across from the AD8-AD15 pins of the PSD311. This positioning enables close layout of the two parts, greatly reducing costs due to less board space.

Additional space is saved by using the latch and buffer for general-purpose I/O instead of the larger and more expensive PIA. And other I/O port lines are not sacrificed by using the multiplexed address/data bus instead of the Serial Peripheral Interface of the 68HC11.

In fact, board space is estimated to have been reduced by more than 50% over the alternative cumbersome design because of the PSD311 positioning on the PC board, its port expansion capabilities, and of course, the number of parts it replaces: including a 256K EPROM, a 16K SRAM, a latch, a decoder, and other miscellaneous CMOS logic.

A benefit of parts reduction is lower CMOS power consumption that results from an integrated single-chip CMOS peripheral/memory solution. By analyzing the power that would have been consumed with the alternative design and comparing that against the PSD311 solution, it was found that power was reduced by at least 30%.

This translates into requiring a smaller power supply and a further reduction in cost.

The flexibility of the PSD311 in the cable tester design is also an advantage when design changes need to be made quickly. Since the I/O ports, PAD, control signals, and EPROM are all programmable, the part just needs to be reprogrammed when the configuration or program memory for the entire system needs modifying.

For instance, the current system has ten I/O, eleven input, and eleven output lines remaining. This can change if other variables need to be stored or other peripherals need to be accessed. To avoid relaying out another board to accommodate these changes, the PSD311 may be able to be reconfigured to easily handle them. Also, if more features and/or capabilities in EPROM are required, the PSD312 and PSD313 with 512Kbits (64K x 8) and 1Mbits (128K x 8) EPROM, respectively, are available in the same package and pinout.

The PSD311 also provides additional SRAM beyond the limited amount that may be on the microcontroller being used. This provides obvious benefits including more scratchpad RAM for such uses as storing cable "signatures" and system tests that can be downloaded for diagnostic purposes.

Benefits of the PSD311 Usage in System (Cont.)

But other benefits not readily seen are also important. For product designs that have a short life cycle and are “pushed” to go to market quickly, the additional SRAM gives the designer the option of writing the code in a high-level language such as “C”, without the worry of running out of variable

storage space. The capability of writing software in “C” could speed up the software development cycle, thereby reducing time-to-market!

Configuring and Programming the PSD311

All of the control logic, address mapping, and port configurations for the PSD311 are handled during device configuration as part of WSI’s easy-to-use, menu-driven PSD MAPLE software program, which is included in the PSD-SILVER or PSD-GOLD software development package. See Appendix A for the PSD311 configuration used in this application.

After the configuration for the PSD311 has been determined and “Save”d, the hex file that is needed for programming the PSD311 is created. That is done during

“Compile”. “Compile” reads the code written for the microcontroller (in Intel hex format) and concatenates or merges it with the PSD311 configuration data to produce the desired output file for downloading to a programmer for programming.

That is all there is to programming the PSD311 which is now supported on industry-standard programmers like the Data I/O, BP Microsystems, Bytek, and Logical Devices programmers as well as the low-cost WSI MagicPro programmer.

The 68HC11/ PSD311 System Software

The software for the 68HC11 was written with a word processor and assembled using a cross assembler. A portion of the cable tester design code which is programmed into the PSD311 is listed in Appendix B. Here the register and RAM memory locations are set up within the first 64 clock cycles from reset of the 68HC11 and located at 0000H to enable easy Direct Addressing and Bit manipulations of often used registers.

Initialization of the Option Register, Timer prescaler, Stack and Serial Communications Interface complete the basic set up for the 68HC11 operation. Other initialization operations include: Ports A and B of the PSD311 which are set up as outputs for display control and data transfer operations, and the LCD display which is set up to display the first screen. Final initialization is achieved by setting several internal registers and clearing any pending interrupts. Now, the IRQ mask bit can be cleared and the main program loop entered.

Included in the code is a demonstration of some useful routines which will illustrate how to easily work with the Latch and Buffer expansion from the 68HC11/PSD311. Remember that these extended addresses off the 68HC11 can be accessed in several ways. The example code shown uses the Bit Set and Bit Clear instructions in the indexed addressing mode. With these Bit Set and Bit Clear instructions, which are read-modify-write instructions, an additional register should be set up in the internal RAM, not on the latched (write-only) address, so the instructions will function properly. Data can then be manipulated and stored as a complete byte to the latch enabling data to be read and the current value in the latch to be checked. (Bit manipulation on the latched addresses using the indexed addressing mode will result in a correct bit change. However, the rest of the byte will be unusable as data on the bus will be scrambled at the rising edge of the chip select signal.) The latch and buffer expansion keeps software algorithms simple.

**The 68HC11/
PSD311 System
Software
(Cont.)**

Regarding the software for the keypad, no debounce software is necessary because the 74C922 has a built in debounce circuit. Actually, direct access from Port E to the keypad data and the AND instruction allows easy compare and execution of the correct routine.

The remaining subroutines in the program are straightforward and basic to most microcontrollers and microprocessors. Those used by the 68HC11 are found in previously published handbooks and articles which can be obtained through your local Motorola sales office.

**Putting the
System to
Work**

The 68HC11/PSD311 cable tester design could be expanded very easily with software to learn many different wiring configurations and to check several cables against a good one. Its usefulness can also be increased by making it battery operated for field use because of the low current draw of the tester.

The cable tester, as designed, will display the test results and step through the program to show the pin by pin connections of the cable. Results are then stored and later fed into a computer through the RS232 communications port of the tester.

Summary

Requirements for microcontroller-based designs are continually changing and to be able to adapt to these changes means being flexible. Of course, flexibility in hardware is sometimes hard to achieve, while flexibility in software is mostly a given. One of the goals of the PSD3XX family of products is to bridge the gap in flexibility between hardware and software.

By that, it is meant that hardware will not be a gating item when developing a new design that needs to be introduced to market quickly. And the PSD311, as illustrated in this cable tester design, addresses that issue perfectly by providing

a user-configurable peripheral solution for hardware designers. So, if an application is modified and the I/O configuration changes, or design fixes are required, the P.C. board does not have to be re-engineered. The PSD3XX can just be reprogrammed to reflect the new changes.

The flexibility provided by the PSD311 solution in this design is crucial in that it enabled development to be completed quickly and successfully using a "core" approach which can handle many different cable applications, including applications for telephone interconnections, printers, and local area networks.

**Appendix A.
PSD311 Part
Configuration
Listed in .SV1
File**

ALIASES

A16/CS8 = CS8
A17/CS9 = IRQ
A18/CS10 = DA
A19/CSI = CSI

GLOBAL CONFIGURATION

Address/Data Mode: MX
Data Bus Size: 8
CSI/A19: CSI
Reset Polarity: LO
ALE Polarity: HI
WRD/RWE: RWE
A16-A19 Transparent or Latched by ALE: T
Using different READ strobes for SRAM and EPROM: N

PORT A CONFIGURATION (Address/IO)

Bit No.	Ai/IO.	CMOS/OD.
0	IO	CMOS
1	IO	CMOS
2	IO	CMOS
3	IO	CMOS
4	IO	CMOS
5	IO	CMOS
6	IO	CMOS
7	IO	CMOS

PORT B CONFIGURATION

Bit No.	CS/IO.	CMOS/OD.
0	IO	CMOS
1	IO	CMOS
2	IO	CMOS
3	IO	CMOS
4	IO	CMOS
5	IO	CMOS
6	IO	CMOS
7	CS7	CMOS

CHIP SELECT EQUATIONS

$$/CS7 = /A15 * /A14 * A13 * /A12 * E * R/W$$

+

PORT C CONFIGURATION

Bit No.	CS/Ai.
0	CS8
1	CS9
2	A18

CHIP SELECT EQUATIONS

$$/CS8 = /A15 * /A14 * A13 * /A12 * E * / R/W$$

$$/IRQ = DA$$

ADDRESS MAP

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	EPROM	EPROM	File Name
	19	18	17	16	15	14	13	12	11	STRT	STOP	START	STOP	
ES0	N	X	N	N	0	1	1	0	N	6000	6FFF	6000	6fff	BASE301.OBJ
ES1	N	X	N	N	0	1	1	1	N	7000	7FFF			

1



**Appendix A.
PSD311 Part
Configuration
Listed in .SV1
File (Cont.)**

```

ES2  N X N N 1 0 0 0 N 8000 8FFF
ES3  N X N N 1 0 0 1 N 9000 9FFF
ES4  N X N N 1 1 0 0 N C000 CFFF c000 cfff BASE301.OBJ
ES5  N X N N 1 1 0 1 N D000 DFFF d000 dfff BASE301.OBJ
ES6  N X N N 1 1 1 0 N E000 EFFF e000 efff BASE301.OBJ
ES7  N X N N 1 1 1 1 N F000 FFFF f000 ffff BASE301.OBJ
RS0  N X N N 0 1 0 1 0 5000 57FF

```

```

CSP  N X N N 0 1 0 0 0 4000 47FF
***** END *****

```

```

CDATA          = 0
CADDRDAT      = 1
CRRWR         = 1
CA19/(/CSI)   = 0
CALE          = 0
CRESET        = 0
COMB/SEP      = 0
CADDHLT       = 0

```

```

CPAF2         = 0

```

```

CPAF1 [0] = 0
CPAF1 [1] = 0
CPAF1 [2] = 0
CPAF1 [3] = 0
CPAF1 [4] = 0
CPAF1 [5] = 0
CPAF1 [6] = 0
CPAF1 [7] = 0

```

```

CPACOD [0] = 0
CPACOD [1] = 0
CPACOD [2] = 0
CPACOD [3] = 0
CPACOD [4] = 0
CPACOD [5] = 0
CPACOD [6] = 0
CPACOD [7] = 0

```

```

CPBF [0] = 1
CPBF [1] = 1
CPBF [2] = 1
CPBF [3] = 1
CPBF [4] = 1
CPBF [5] = 1
CPBF [6] = 1
CPBF [7] = 0

```

```

CPBCOD [0] = 0
CPBCOD [1] = 0
CPBCOD [2] = 0
CPBCOD [3] = 0
CPBCOD [4] = 0
CPBCOD [5] = 0
CPBCOD [6] = 0
CPBCOD [7] = 0

```

```

CPCF [0] = 1
CPCF [1] = 1
CPCF [2] = 0

```

Appendix B. Core System Software for Cable Tester Design

```

0000          CPU      "6811.TBL"
0000          HOF      "INT8"
;
;*****
;*   THE 68HC11 IN CONJUNCTION WITH THE PSD301   *
;*   ARE USED IN DEVELOPEMENT OF SOFTWARE FOR   *
;*   DISPLAY, KEYBOARD FUNCTION, AND OTHER APPL. *
;*   MEMORY MAP:EPROM(1)  C000-FFFF (PROGRAM)   *
;*                   EPROM    B600-BFFF (68HC11) *
;*                   EPROM(2) 6000-9FFF (DATA)   *
;*                   RAM      5000-5FFF (PSD301) *
;*                   I/O      4000-4007 (PSD301) *
;*                   LAT      2000 (LATCH & BUFFER) *
;*                   RAM      1000-10FF (68HC11) *
;*                   I/O & REG 0000-003F (68HC11) *
;*
;*                   BY TIM DUNAVIN              *
;*                   ANTEC                       *
;*                   ANIXTER MANUFACTURING      *
;*****
;
6000          ORG      06000H      ;DATA MEMORY
;
;*****
;*   LOOKUP TABLES   *
;*****
;
6000 3638484331DATTAB: DFB      "68HC11/PSD311 UP",00H
;
6011 54494D4F54CREDITS: DFB      "TIMOTHY E. DUNAVIN"
6023 414E544543      DFB      "ANTEC - ANIXTER MFG."
6037 524F434B20      DFB      "ROCK FALLS, ILL. 61071"
;
;*****
;
C000          ORG      0C000H      ;PROGRAM MEMORY
;
103D =      INIT:    EQU      103DH      ;RAM AND I/O MAPPING REGISTER
4000 =      PORTBC:  EQU      04000H     ;I/O BASE ADDRESS OF THE 301
2000 =      LAT:     EQU      02000H     ;LATCH AND BUFFER
0000 =      KEY1:    EQU      00H        ;KEYPAD 1
0001 =      KEY2:    EQU      01H        ;KEYPAD 2
0002 =      KEY3:    EQU      02H        ;KEYPAD 3
0003 =      KEY4:    EQU      03H        ;KEYPAD A
0004 =      KEY5:    EQU      04H        ;KEYPAD 4
0005 =      KEY6:    EQU      05H        ;KEYPAD 5
0006 =      KEY7:    EQU      06H        ;KEYPAD 6
0007 =      KEY8:    EQU      07H        ;KEYPAD B
0008 =      KEY9:    EQU      08H        ;KEYPAD 7
0009 =      KEY0:    EQU      09H        ;KEYPAD 8
000A =      KEY1:    EQU      0AH        ;KEYPAD 9
000B =      KEY2:    EQU      0BH        ;KEYPAD C
000C =      KEY3:    EQU      0CH        ;KEYPAD *
000D =      KEY4:    EQU      0DH        ;KEYPAD 0
000E =      KEY5:    EQU      0EH        ;KEYPAD #
000F =      KEY6:    EQU      0FH        ;KEYPAD D
;

```

1

Appendix B. Core System Software for Cable Tester Design (Cont.)

```

;*****
;*      INITIALIZATION ROUTINE      *
;*****
;
;NOTE: OPTION and TMSK2 must be programed in first 64 E
;      cycles out of RESET
;
;
C000 0F      START:      SEI          ;SET IRQ MASK
C001 8610    LDAA        #010H      ;SET RAM AT 1000 AND
C003 B7103D  STAA        INIT        ;SET REGISTERS AT 0000
;*****
;
;***** 64 BYTES OF REGISTER AREA *****
0000 =      PORTA:      EQU         0000H      ;PORT A DATA REGISTER
;0001 IS RESERVED
0002 =      PIOC:       EQU         0002H      ;PARALLEL I/O CONTROL REGISTER
0003 =      PORTC:      EQU         0003H      ;PORT C DATA REGISTER (AD0 - AD7)
0004 =      PORTB:      EQU         0004H      ;PORT B DATA REGISTER (A8 - A15)
0005 =      PORTCL:    EQU         0005H      ;PORT C LATCHED DATA REGISTER
;0006 IS RESERVED
0007 =      DDRC:       EQU         0007H      ;DATA DIRECTION REG FOR PORT C
0008 =      PORTD:      EQU         0008H      ;PORT D DATA REGISTER (RxD, TxD, AND I/O)
0009 =      DDRD:       EQU         0009H      ;DATA DIRECTION REG FOR PORT D
000A =      PORTE:      EQU         000AH      ;PORT E DATA REGISTER
000B =      CFORC:      EQU         000BH      ;TIMER COMPARE FORCE REGISTER
000C =      OC1M:       EQU         000CH      ;OUTPUT COMPARE 1 MASK REGISTER
000D =      OC1D:       EQU         000DH      ;OUTPUT COMPARE 1 DATA REGISTER
000E =      TCNT:       EQU         000EH      ;TIMER COUNTER REGISTER (16 BIT)
;000F LSB TCNT
0010 =      TIC1:       EQU         0010H      ;TIMER INPUT CAPTURE REGISTER 1 (16 BIT)
;0011 LSB TIC1
0012 =      TIC2:       EQU         0012H      ;TIMER INPUT CAPTURE REGISTER 2 (16 BIT)
;0013 LSB TIC2
0014 =      TIC3:       EQU         0014H      ;TIMER INPUT CAPTURE REGISTER 3 (16 BIT)
;0015 LSB TIC3
0016 =      TOC1:       EQU         0016H      ;TIMER OUTPUT COMPARE REG 1 (16 BIT)
;0017 LSB TOC1
0018 =      TOC2:       EQU         0018H      ;TIMER OUTPUT COMPARE REG 2 (16 BIT)
;0019 LSB TOC2
001A =      TOC3:       EQU         001AH      ;TIMER OUTPUT COMPARE REG 3 (16 BIT)
;001B LSB TOC3
001C =      TOC4:       EQU         001CH      ;TIMER OUTPUT COMPARE REG 4 (16 BIT)
;001D LSB TOC4
001E =      TOC5:       EQU         001EH      ;TIMER OUTPUT COMPARE REG 5 / INPUT CAPTURE
;REGISTER 4 (16 BIT) - 001F LSB TOC5/TIC4
0020 =      TCTL1:      EQU         0020H      ;TIMER CONTROL REGISTER 1
0021 =      TCTL2:      EQU         0021H      ;TIMER CONTROL REGISTER 2
0022 =      TMSK1:      EQU         0022H      ;MAIN TIMER INT MASK REGISTER 1
0023 =      TFLG1:      EQU         0023H      ;MAIN TIMER INT. FLAG REG 1
0024 =      TMSK2:      EQU         0024H      ;MAIN TIMER INT MASK REGISTER 2
0025 =      TFLG2:      EQU         0025H      ;MAIN TIMER INT. FLAG REG 2
0026 =      PACTL:      EQU         0026H      ;PULSE ACCUMULATOR CONTROL REG
0027 =      PACNT:      EQU         0027H      ;PULSE ACCUMULATOR COUNT REG
0028 =      SPCR:       EQU         0028H      ;SPI CONTROL REGISTER
0029 =      SPSR:       EQU         0029H      ;SPI STATUS REGISTER
002A =      SPDR:       EQU         002AH      ;SPI DATA REGISTER
002B =      BAUD:       EQU         002BH      ;SCI BAUD RATE CONTROL REGISTER
002C =      SCCR1:      EQU         002CH      ;SCI CONTROL REGISTER 1
002D =      SCCR2:      EQU         002DH      ;SCI CONTROL REGISTER 2
002E =      SCSR:       EQU         002EH      ;SCI STATUS REGISTER
002F =      SCDR:       EQU         002FH      ;SCI DATA REGISTER
0030 =      ADCTL:      EQU         0030H      ;A/D CONTROL/STATUS REGISTER
0031 =      ADRL:       EQU         0031H      ;A/D RESULT REGISTER 1
0032 =      ADRL:       EQU         0032H      ;A/D RESULT REGISTER 2
0033 =      ADRL:       EQU         0033H      ;A/D RESULT REGISTER 3
0034 =      ADRL:       EQU         0034H      ;A/D RESULT REGISTER 4
;0035 - 0038 RESERVED

```

Appendix B. Core System Software for Cable Tester Design (Cont.)

```

0039 = OPTION: EQU 0039H ;SYSTEM CONFIGURATION OPTIONS
003A = COPRST: EQU 003AH ;ARM/RESET COP TIMER CIRCUITRY
003B = PPROG: EQU 003BH ;EEPROM PROGRAMMING REGISTER
003C = HPRIO: EQU 003CH ;HIGHEST PRIORITY INTERRUPT
;INIT: EQU 003DH ;RAM AND I/O MAPPING REGISTER (NEW ADD.)
003E = TEST1: EQU 003EH ;FACTORY TEST REGISTER
003F = CONFIG: EQU 003FH ;CONFIGURATION CONTROL REGISTER
;
;***** 256 BYTES OF INTERNAL RAM *****
1000 = FLAGS: EQU 1000H ;FLAG REGISTER
1001 = LAL: EQU 1001H ;LATCH DATA REGISTER
1002 = STOR: EQU 1002H ;BASIC RAM STORAGE AREA
10FF = STACK: EQU 10FFH ;STACK AREA
;
;***** 2K X 8 EXTERNAL RAM *****
5000 = MASSTOR: EQU 05000H ;MASS STORAGE RAM IN PSD301
;
;***** EEROM AREA, 512 BYTES *****
B600 = EROM: EQU 0B600H ;DATA RETENTION AREA
;
;*****
C006 01 NOP ;SLIGHT DELAY TO ALLOW REGISTER SET UP
C007 86E3 LDAA #0E3H ;SET UP OPTION REG. - ADPU =1, CSEL = 1,
; IRQE = 1
C009 9739 STAA OPTION ;(ENABLE EEPROM CHARGE PUMP, IRQ EDGE
; SENSITIVE)
C00B 8602 LDAA #002H ;SET TIMER PRESCALER TO 8
C00D 9724 STAA TMSK2 ;AND DISABLE TIMER INTERRUPTS
C00F 7F0028 CLR SPCR ;DISABLE ALL SPI INT.
C012 8E10FF LDS #STACK ;SET UP STACK
C015 8680 LDAA #080H
C017 9726 STAA PACTL ;PA7 OUTPUT
;***** INITIALIZE THE SCI TO 9600 BAUD AT 8MHZ (DISABLED)
C019 86FC ONSCI: LDAA #0FCH ;INIT. PORT D DDR (02H)
C01B 9709 STAA DDRD ;PD0, PD1 - INPUT, PD2-PD5 - OUTPUT
C01D 8600 LDAA #000H ;SET UP PORT D
C01F 9708 STAA PORTD
C021 7F002C CLR SCCR1 ;SET UP SER. COM. CON. REG. 1
C024 7F002D CLR SCCR2
C027 962E LDAA SCSR ;TO CLEAR TDRE AND TC OF SCSR
C029 4F CLRA ;READ STATUS REG., LOAD TRANS. DATA REG.
C02A 972F STAA SCDR
;***** INITIALIZE THE 301 FOR DISPLAY INTERFACE
C02C CFFFFFF ONPIA: LDX #0FFFFH ;SET UP PORTS B & C AS OUTPUTS
C02F FF4004 STX PORTBC+4
;***** DISPLAY SET UP (NEW REV. 15 MAY 91) *****
C032 CE2710 DISINIT: LDX #02710H ;100ms DELAY (POWER UP DELAY FOR DISPLAY)
C035 BDC0E1 JSR TDELAY ;TIME DELAY
C038 8630 LDAA #030H ;SET UP DISPLAY
C03A BDC0F4 JSR SENDI ;SEND INSTRUCTION (30 1ST TIME)
C03D CE0300 LDX #00300H ;6.1ms DELAY
C040 BDC0E1 JSR TDELAY ;TIME DELAY
C043 BDC0F4 JSR SENDI ;SEND INSTRUCTION (30 2ND TIME)
C046 BDC0DE JSR TD40 ;TIME DELAY
C049 BDC0F4 JSR SENDI ;SEND INSTRUCTION (30 3RD TIME)
C04C BDC0DE JSR TD40 ;TIME DELAY
C04F 8638 LDAA #038H ;FUNCTION SET (8 BIT-SINGLE LINE)
C051 BDC0F4 JSR SENDI ;SEND INSTRUCTION
C054 CE0280 LDX #00280H ;5ms DELAY
C057 BDC0E1 JSR TDELAY ;TIME DELAY
C05A 860C LDAA #00CH ;DISPLAY ON - NO CURSOR

```

**Appendix B.
Core System
Software for
Cable Tester
Design (Cont.)**

```

C05C BDC0F4      JSR      SENDI      ;SEND INSTRUCTION
C05F CE0280      LDX      #00280H     ;5mS DELAY
C062 BDC0E1      JSR      TDELAY     ;TIME DELAY
C065 8606        LDAA     #006H      ;ENTRY MODE SET
C067 BDC0F4      JSR      SENDI      ;SEND INSTRUCTION
C06A CE0280      LDX      #00280H     ;5mS DELAY
C06D BDC0E1      JSR      TDELAY     ;TIME DELAY
C070 BDC0EC      JSR      HOME      ;DISPLAY CURSOR HOME!
C073 CE0190      LDX      #00190H     ;4.0mS DELAY
C076 BDC0E1      JSR      TDELAY     ;TIME DELAY
C079 18CE6000    LDY      #DATTAB     ;TOP OF DATA TABLE
C07D BDC0CC      JSR      PDOD      ;SEND MESSAGE TO DISPLAY
;***** FINAL INIT. *****
C080 9629        LDAA     SPSR      ;CLEAR ANY SPI INT.
C082 962A        LDAA     SPDR      ;
C084 86FF        LDAA     #0FFH     ;CLEAR ANY TIMER INT.
C086 9723        STAA     TFLG1    ;
C088 9725        STAA     TFLG2    ;
C08A 962E        LDAA     SCSR      ;CLEAR ANY SCI INT.
C08C 962F        LDAA     SCDR      ;
;
;EXAMPLES OF WORKING WITH LATCH AND BUFFER
C08E 7F2000      CLR      LAT      ;CLEAR LATCH
C091 CE1001      LDX      #LA1      ;SET INDEX
C094 1C0200      BSET     2,X,00H   ;SET BIT 2 OF LA1
C097 A600        LDAA     0,X      ;GET LATCH REGISTER
C099 B72000      STAA     LAT      ;STORE DATA TO LATCH
C09C B62000      LDAA     LAT      ;GET DATA FROM BUFFER
;
C09F BDC0B0      JSR      BEEP      ;SOUND OFF!
;
C0A2 0E          CLI      ;CLEAR IRQ MASK
;
;*****
;*      MAIN LOOP      *
;*****
;
C0A3 01          LOOP:   NOP
C0A4 7EC0A3      JMP      LOOP      ;RETURN
;
;*****
;*      SUBROUTINES    *
;*****
;
;***** WATCHDOG SERVICE ROUTINE *****
C0A7 8655        DOG:    LDAA     #055H     ;RESET WATCHDOG TIMER
C0A9 973A        STAA     COPRST
C0AB 86AA        LDAA     #0AAH
C0AD 973A        STAA     COPRST
C0AF 39          RTS      ;RETURN FROM SUB.
;
;***** HOOTER OSC. ROUTINE *****
C0B0 18CE01FF    BEEP:   LDY      #001FFH ;SET COUNT
C0B4 8640        BEEP1:  LDAA     #040H   ;BEEPER ON
C0B6 9700        STAA     PORTA
C0B8 CE0014      LDX      #00014H
C0BB BDC0E1      JSR      TDELAY     ;DELAY
C0BE 4F          CLRA     ;BEEPER OFF
C0BF 9700        STAA     PORTA
C0C1 CE0014      LDX      #00014H
C0C4 BDC0E1      JSR      TDELAY     ;DELAY
C0C7 1809        DEY      ;COUNT -1
C0C9 26E9        BNE     BEEP1   ;IF NOT DONE, KEEP GOING
C0CB 39          RTS      ;RETURN FROM SUB.

```



Appendix B. Core System Software for Cable Tester Design (Cont.)

```

;***** PUT DATA ON DISPLAY *****
C0CC 18A600 PDOD: LDAA 0,Y ;GET BYTE
C0CF 2707 BEQ PDOD1 ;IF END, GOTO NEXT1
C0D1 BDC100 JSR SENDD
C0D4 1808 INY ;NEXT BYTE
C0D6 20F4 BRA PDOD ;RETURN TO NEXT
C0D8 39 PDOD1: RTS ;RETURN FROM SUB.
;
;***** TIME DELAY ROUTINE *****
C0D9 CE0002 TD20: LDX #00002H ;20us DELAY
C0DC 2003 BRA TDELAY
C0DE CE000F TD40: LDX #0000FH ;150us DELAY
C0E1 09 TDELAY: DEX ;DECAMENT COUNT
C0E2 8C0000 CPX #00000H ;COUNT = 0?
C0E5 26FA BNE TDELAY ;IF NOT DONE, GOTO TDELAY
C0E7 39 RTS ;RETURN FRO SUB.
;
;***** CLEAR SCREEN, CURSOR HOME, AND SEND INSTRUCTION *****
C0E8 8601 CSCREEN: LDAA #001H ;CLEAR DISPLAY
C0EA 2008 BRA SENDI ;SEND INSTRUCTION
C0EC 8602 HOME: LDAA #002H ;CURSOR HOME
C0EE 2004 BRA SENDI ;SEND INSTRUCTION
C0F0 86C0 LINE2: LDAA #0C0H ;SET CURSOR TO LINE 2
C0F2 2000 BRA SENDI ;SEND INSTRUCTION
C0F4 CE4000 SENDI: LDX #PORTBC ;SET UP DATA TRANSFER
C0F7 A706 STAA 6,X ;STORE AT PIA PORT A
C0F9 1C0702 BSET 7,X,02H ;DISPLAY E HIGH
C0FC 1D0702 BCLR 7,X,02H ;DISPLAY E LOW
C0FF 39 RTS ;RETURN FROM SUB.
;
;***** SEND DATA TO DISPLAY *****
C100 CE4000 SENDD: LDX #PORTBC ;SET UP DATA TRANSFER
C103 A706 STAA 6,X ;SEND DATA
C105 1C0701 BSET 7,X,01H ;DISPLAY RS HIGH
C108 1C0702 BSET 7,X,02H ;DISPLAY E HIGH
C10B 1D0702 BCLR 7,X,02H ;DISPLAY E LOW
C10E 1D0701 BCLR 7,X,01H ;DISPLAY RS LOW
C111 BDC0DE JSR TD40 ;150us TIME DELAY
C114 39 RTS ;RETURN FROM SUB.
;
;*****
;* ROUTINE TO CHANGE BYTE IN EEROM *
;* PRELOADED X = ADDRESS IN EEROM (B600 - B7FF) *
;* DATA TO BE STORED, IS IN "STOR" *
;* (THIS IS A MOTOROLA ROUTINE) *
;*****
C115 A600 CHGBYT: LDAA 0,X ;GET DATA AT ADDRESS TO BE CHANGED
C117 81FF CMPA #0FFH ;CHECK IF ERASED
C119 2717 BEQ CHGBYT1 ;JUMP IF BYTE ERASED
C11B 8616 LDAA #016H ;SET BYTE, ERASE, AND EELAT
C11D 973B STAA PPROG
C11F 86FF LDAA #0FFH
C121 A700 STAA 0,X
C123 8617 LDAA #017H ;SET EEPRG
C125 973B STAA PPROG
C127 3C PSHX ;SAVE X
C128 CE0300 LDX #00300H
C12B BDC0E1 JSR TDELAY ;20ms TIME DELAY
C12E 38 PULX ;RESTORE X
C12F 4F CLRA ;CLEAR BYTE, ERASE, EELAT, AND EEPRG
C130 973B STAA PPROG ;END OF BYTE ERASE
C132 8602 CHGBYT1: LDAA #002H ;SET EELAT - DO BYTE PROGRAM

```

**Appendix B.
Core System
Software for
Cable Tester
Design (Cont.)**

```

C134 973B          STAA    PPROG
C136 B61002       LDAA    STOR          ;GET DATA TO BE STORED
C139 A700         STAA    0,X          ;STORE IN NEW LOCATION IN EEROM
C13B 7C003B       INC     PPROG
C13E 3C          PSHX          ;SAVE X
C13F CE0300       LDX    #00300H
C142 BDC0E1       JSR    TDELAY          ;20ms DELAY
C145 38          PULX          ;RESTORE X
C146 7A003B       DEC     PPROG          ;CLEAR EEPRG
C149 7F003B       CLR    PPROG          ;CLEAR EELAT, END OF BYTE PROGRAM
C14C 39          RTS          ;RETURN FROM SUB.

;
;*****
;* ROUTINE TO SET UP A/D CONVERTER *
;* ACC A = VALUE TO INITIATE CONVERSION *
;* BEFORE ENTRY TO THIS ROUTINE *
;*****
C14D 9730 CONV: STAA    ADCTL          ;SET UP A/D CONVERTER
C14F 133080FC CONV1: BRCLR  ADCTL,80H,CONV1 ;WAIT HERE TILL CONVERSION COMPLETE
C153 39          RTS          ;RETURN FROM SUB.

;
;*****
;* INTERRUPT ROUTINES *
;*****
;
;*****
;* SERIAL COMMUNICATIONS INTERFACE - IRQ *
;*****
;
C154 3B          SCOM:  RTI          ;RETURN FROM INT.
;
;*****
;* SERIAL TRANSFER COMPLETE *
;*****
;
C155 3B          TRANC: RTI          ;RETURN FROM INT.
;
;*****
;* PULSE ACCUMULATOR INPUT EDGE *
;*****
;
C156 3B          PULSEE: RTI          ;RETURN FROM INT.
;
;*****
;* PULSE ACCUMULATOR OVERFLOW *
;*****
;
C157 3B          PULSEO: RTI          ;RETURN FROM INT.
;
;*****
;* TIMER OVERFLOW *
;*****
;
C158 3B          TIMEO:  RTI          ;RETURN FROM INT.
;
;*****
;* TIMER OUTPUT COMPARE 5 *
;*****
;
C159 3B          COMP5:  RTI          ;RETURN FROM INT.
;

```

Appendix B. Core System Software for Cable Tester Design (Cont.)

```

;*****
;* TIMER OUTPUT COMPARE 4 *
;*****
C15A 3B COMP4: RTI ;RETURN FROM INT.
;
;*****
;* TIMER OUTPUT COMPARE 3 *
;*****
C15B 3B COMP3: RTI ;RETURN FROM INT.
;
;*****
;* TIMER OUTPUT COMPARE 2 *
;*****
C15C 3B COMP2: RTI ;RETURN FROM INT.
;
;*****
;* TIMER OUTPUT COMPARE 1 *
;*****
C15D 3B COMP1: RTI ;RETURN FROM INT.
;
;*****
;* TIMER INPUT COMPARE 3 *
;*****
C15E 3B ICOMP3: RTI ;RETURN FROM INT.
;
;*****
;* TIMER INPUT COMPARE 2 *
;*****
C15F 3B ICOMP2: RTI ;RETURN FROM INT.
;
;*****
;* TIMER INPUT COMPARE 1 *
;*****
C160 3B ICOMP1: RTI ;RETURN FROM INT.
;
;*****
;* REAL TIME INT. ROUTINE *
;*****
C161 3B REALT: RTI ;RETURN FROM INT.
;
;*****
;* IRQ INT. ROUTINE *
;*****
C162 960A DOIT: LDA PORTC ;GET KEYBOARD DATA
C164 840F ANDA #00FH ;FILTER DATA
;
C166 8100 CMPA #KEY1 ;1 KEY?
C168 2601 BNE DOIT10 ;IF NOT GOTO DOIT10
C16A 3B RTI ;RETURN FROM INT.
;
C16B 8101 DOIT10: CMPA #KEY2 ;2 KEY?
C16D 2601 BNE DOIT20 ;IF NOT GOTO DOIT20
C16F 3B RTI ;RETURN FROM INT.

```


**Appendix B.
Core System
Software for
Cable Tester
Design (Cont.)**

```

;
C170 8102 DOIT20: CMPA #KEY3 ;3 KEY?
C172 2601 BNE DOIT30 ;IF NOT, GOTO DOIT30
C174 3B RTI ;RETURN FROM INT.
;
C175 8103 DOIT30: CMPA #KEYA ;A KEY?
C177 2601 BNE DOIT40 ;IF NOT GOTO DOIT40
C179 3B RTI ;RETURN FROM INT.
;
C17A 8104 DOIT40: CMPA #KEY4 ;4 KEY?
C17C 2601 BNE DOIT50 ;IF NOT GOTO DOIT50
C17E 3B RTI ;RETURN FROM INT.
;
C17F 8105 DOIT50: CMPA #KEY5 ;5 KEY?
C181 2601 BNE DOIT60 ;IF NOT GOTO DOIT60
C183 3B RTI ;RETURN FROM INT.
;
C184 8106 DOIT60: CMPA #KEY6 ;6 KEY?
C186 2601 BNE DOIT70 ;IF NOT GOTO DOIT70
C188 3B RTI ;RETURN FROM INT.
;
C189 8107 DOIT70: CMPA #KEYB ;B KEY?
C18B 2601 BNE DOIT80 ;IF NOT GOTO DOIT80
C18D 3B RTI ;RETURN FROM INT.
;
C18E 8108 DOIT80: CMPA #KEY7 ;7 KEY?
C190 2601 BNE DOIT90 ;IF NOT GOTO DOIT90
C192 3B RTI ;RETURN FROM INT.
;
C193 8109 DOIT90: CMPA #KEY8 ;8 KEY?
C195 2601 BNE DOIT100 ;IF NOT GOTO DOIT100
C197 3B RTI ;RETURN FROM INT.
;
C198 810A DOIT100: CMPA #KEY9 ;9 KEY?
C19A 2601 BNE DOIT110 ;IF NOT GOTO DOIT110
C19C 3B RTI ;RETURN FROM INT.
;
C19D 810B DOIT110: CMPA #KEYC ;C KEY?
C19F 2601 BNE DOIT120 ;IF NOT GOTO DOIT120
C1A1 3B RTI ;RETURN FROM INT.
;
C1A2 810C DOIT120: CMPA #KEYZ ;* KEY?
C1A4 2601 BNE DOIT130 ;IF NOT GOTO DOIT130
C1A6 3B RTI ;RETURN FROM INT.
;
C1A7 810D DOIT130: CMPA #KEY0 ;0 KEY?
C1A9 2601 BNE DOIT140 ;IF NOT GOTO DOIT140
C1AB 3B RTI ;RETURN FROM INT.
;
C1AC 810E DOIT140: CMPA #KEYY ;# KEY?
C1AE 2601 BNE DOIT150 ;IF NOT GOTO DOIT150
C1B0 3B RTI ;RETURN FROM INT.
;
C1B1 810F DOIT150: CMPA #KEYD ;D KEY?
C1B3 2600 BNE DOIT160 ;IF NOT GOTO DOIT160
C1B5 3B DOIT160: RTI ;RETURN FROM INT.
;
;*****
;* XIRQ SERVICE ROUTINE *
;*****

```

Appendix B. Core System Software for Cable Tester Design (Cont.)

```

C1B6 3B      NOMASK: RTI                ;RETURN FROM INT.
;
;*****
;* SWI SERVICE ROUTINE                *
;*****
C1B7 3B      INTER: RTI                ;RETURN FROM INT.
;
;*****
;* RESET AND INTERRUPT VECTORS        *
;*****
;
FFC0          ORG      0FFC0H
;
FFC0          RES:     DFS      11*2      ;NOT USED
FFD6 C154     SERCOM:  DWM      SCOM      ;SERIAL COMM. INT.
FFD8 C155     SPISTC: DWM      TRANC     ;SERIAL TRANSFER COMPLETE
FFDA C156     PAIE:   DWM      PULSEE    ;PULSE ACCUMULATOR INPUT EDGE
FFDC C157     PAOV:   DWM      PULSEO    ;PULSE ACCUMULATOR OVERFLOW
FFDE C158     TOV:    DWM      TIMEO     ;TIMER OVERFLOW
FFE0 C159     TOCP5:  DWM      COMP5     ;TIMER OUTPUT COMPARE 5
FFE2 C15A     TOCP4:  DWM      COMP4     ;TIMER OUTPUT COMPARE 4
FFE4 C15B     TOCP3:  DWM      COMP3     ;TIMER OUTPUT COMPARE 3
FFE6 C15C     TOCP2:  DWM      COMP2     ;TIMER OUTPUT COMPARE 2
FFE8 C15D     TOCP1:  DWM      COMP1     ;TIMER OUTPUT COMPARE 1
FFEA C15E     TICP3:  DWM      ICOMP3    ;TIMER INPUT COMPARE 3
FFEC C15F     TICP2:  DWM      ICOMP2    ;TIMER INPUT COMPARE 2
FFEE C160     TICP1:  DWM      ICOMP1    ;TIMER INPUT COMPARE 1
FFF0 C161     RTIME:  DWM      REALT     ;REAL-TIME INT.
FFF2 C162     IRQ:    DWM      DOIT      ;TIMER/VIA INT.
FFF4 C1B6     XIRQ:   DWM      NOMASK    ;NON-MASKABLE INT.
FFF6 C1B7     SWI:    DWM      INTER     ;SOFTWARE INT.
FFF8 C000     IOT:    DWM      START     ;ILLEGAL OPCODE TRAP (START OVER)
FFFA C000     COPS:   DWM      START     ;COP FAILURE (RESET)
FFFC C000     COPS1:  DWM      START     ;COP CLOCK MONITOR FAIL (RESET)
FFFE C000     RESET:  DWM      START     ;RESET
;
;*****
0000          END                ;THE END!!!!

```




Programmable Peripheral Application Note 020 Benefits of 16-Bit Design with PSD3XX

By Ching Lee

Introduction

Embedded controller architecture has been evolving from 4-bit, 8-bit to 16-bit through the years. The increase in the data bus bandwidth is a natural progression for microcontrollers to achieve higher performance. Today, 16-bit embedded controllers such as the 80C196 and 683XX families provide excellent performance at reasonable cost. Yet many designers are weary of the cost of higher chip count, more board space and power consumption in 16-bit applications and prefer to stay with 8-bit designs. Some microcontroller manufacturers tackle this problem by introducing processors with 16-bit internal architectures but have 8-bit external data busses. Later additional enhancements such as dynamic bus sizing provide the choice of selecting either an 8 or 16-bit bus for further cost reduction. This compromise

certainly increases the performance; it is still not as good as a true 16-bit implementation.

With the introduction of the PSD3XX family of field programmable microcontroller peripherals from WSI, there is no reason not to use 16-bit microcontrollers. The PSD3XX provides an integrated solution in a single chip, which includes user configurable I/O ports, Chip Select outputs, logic replacement, Page Register, Programmable Address Decoder (PAD), EPROM and SRAM. The PSD3XX is a perfect match for 16-bit microcontroller applications. In this application note, we will look at some of the advantages of 16-bit designs, and how PSD3XX interfaces to microcontrollers such as the 80C196 and 68302.

1

Typical 16-Bit Microcontroller System Architecture

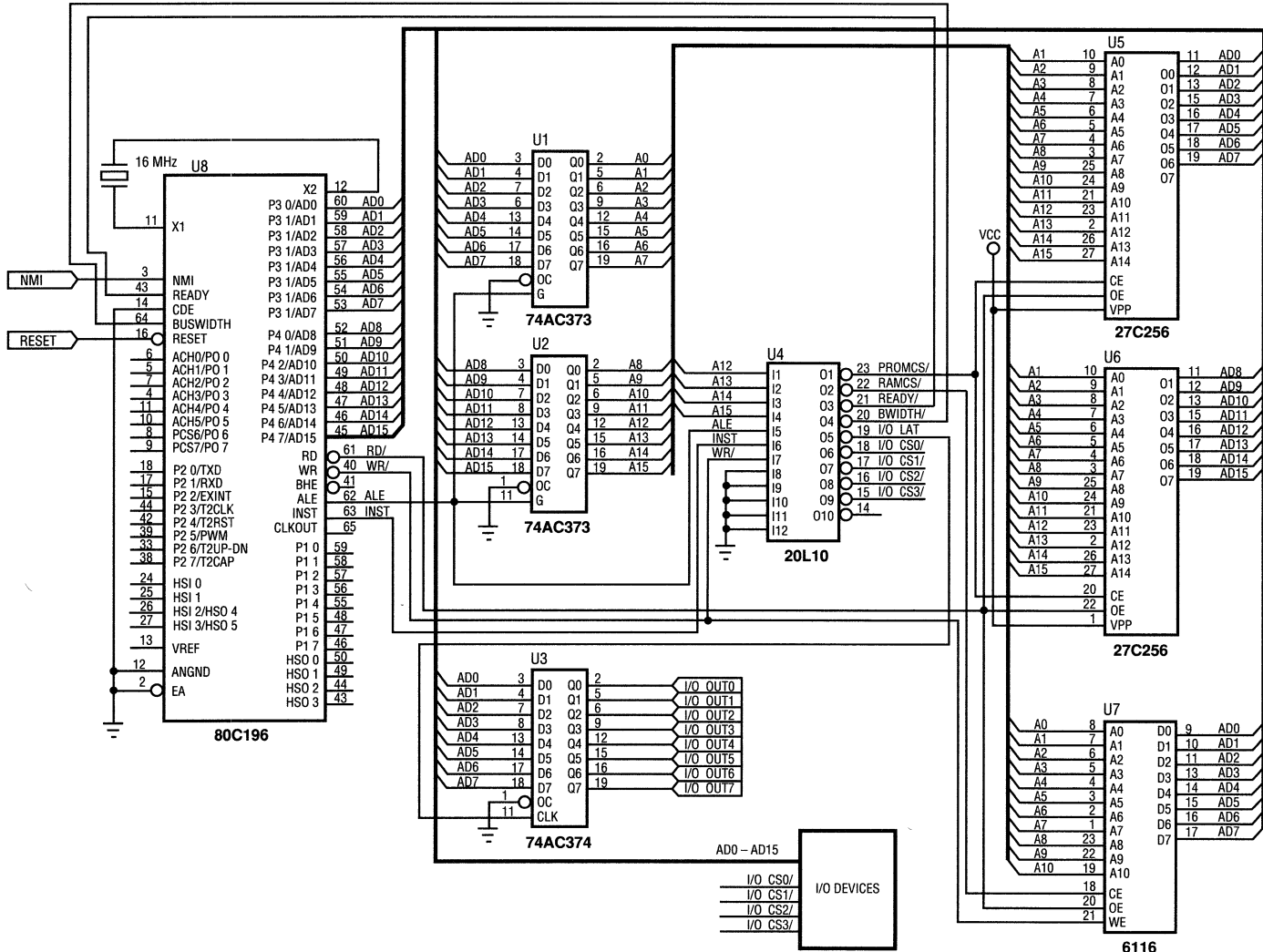
There is no one standard 16-bit architecture, especially in the field of embedded controller applications. For a typical 80C196 design, the basic building block consists of two address latches (74AC373), address decoding logic (with PAL or discrete logic), program memory (EPROMs), data memory (one or more SRAM), and I/O devices.

Figure 1 is the schematic of such a system. In this design, 64K bytes of program memory/EPROM, and a 2K byte SRAM for scratch pad are required. Since the 80C196 has only 64K byte memory space, the INST signal provides the paging capability, with program memory residing in the first 64K page while SRAM and I/O devices occupy the second page. The I/O section consists of one output port (74AC374) and other peripheral devices. The chip select signals for the I/O devices and memory are connected directly from the decoding PAL outputs. The processor's data bus width is determined by the type of

bus cycle. EPROM accesses are 16-bits wide, SRAM is 8-bits while I/O bus cycles can be 8 or 16-bits, depending on the device being accessed. The BWIDTH output from the PAL informs the processor what type of bus width is to be expected for that particular cycle.

An I/O device usually takes longer time to complete the bus cycle. Let us assume, in this case, I/O devices require 3 wait states with the exception of the I/O latch. The configuration register of the 80C196 is then programmed to insert 3 wait states. Whenever there is an I/O bus cycle, the READY output signal from the PAL goes low to activate the processor's wait state control to insert the programmed amount of wait state. For memory bus cycles, no wait state is inserted.

Figure 1. Typical 16-Bit Microcontroller System Architecture



**Typical 16-Bit
Microcontroller
System
Architecture
(Cont.)**

Table 1 is the memory address map of the 80C196 microcontroller, and the addresses of the I/O devices. Address locations 0000H through 00FFH and 1FFEh through 207FH are reserved for the microcontroller. The remaining locations can be used for program/data memory or memory mapped I/O devices. EPROM occupies the first 64K bytes, where program codes start from 2080H to FFFFH, and a 2K look-up table

resides inside the EPROM from location 1000H to 17FFH. The 2K scratch RAM and I/O starts from 4000H in the second page.

The address map requires the following PAL equations to be programmed to the decoder PAL. The IO_CS lines are enabled after ALE goes low.

$$\begin{aligned} \text{EPROMCS} &= \text{INST} \\ &+ \text{INST/} * \text{A15/} * \text{A14/} \\ \\ \text{RAMCS} &= \text{INST/} * \text{A15/} * \text{A14} * \text{A13/} * \text{A12/} \\ \\ \text{BWIDTH} &= \text{RAMCS} \\ &+ \text{IO_CS0} \\ &+ \text{IO_CS1} \\ &+ \text{IO_LAT} \\ \\ \text{READY} &= \text{IO_CS0} \\ &+ \text{IO_CS1} \\ &+ \text{IO_CS2} \\ &+ \text{IO_CS3} \\ \\ \text{IO_LAT} &= \text{INST/} * \text{WR} * \text{A15/} * \text{A14} * \text{A13/} * \text{A12} \\ \text{IO_CS0} &= \text{INST/} * \text{ALE/} * \text{A15/} * \text{A14} * \text{A13} * \text{A12/} \text{ "/O DEV.#0} \\ \text{IO_CS1} &= \text{INST/} * \text{ALE/} * \text{A15/} * \text{A14} * \text{A13} * \text{A12} \text{ "/O DEV.#1} \\ \text{IO_CS2} &= \text{INST/} * \text{ALE/} * \text{A15} * \text{A14/} * \text{A13/} * \text{A12/} \text{ "/O DEV.#2} \\ \text{IO_CS3} &= \text{INST/} * \text{ALE/} * \text{A15} * \text{A14/} * \text{A13/} * \text{A12} \text{ "/O DEV.#3} \end{aligned}$$

1

**Table 1.
80C196
Memory Map**

<i>Device</i>	<i>INST (Page)</i>	<i>Address (Hex)</i>	<i>Buswidth (Bit)</i>
EPROM (Code)	1	2080 – FFFF	16
EPROM (Table + Data)	X	1000 – 27FF	16
RAM	0	4000 – 47FF	8
I/O_LATCH	0	5000	8
I/O_CS0	0	6000	8
I/O_CS1	0	7000	8
I/O_CS2	0	8000	16
I/O_CS3	0	9000	16

16-Bit Performance Advantages

It is obvious that a 16-bit bus provides more performance than an 8-bit bus, at least the data bus bandwidth will double. The following factors contribute to the performance improvement:

Program Code Fetch

Instructions such as ANDB of the 80C196 consists of 4 bytes. In an 8-bit bus system it takes 4 bus cycles to fetch the instruction, while in 16-bit bus designs it takes only 2 bus cycles.

Data Fetch

For applications with high data transfer rate, where indexed or indirect references are frequently used, a 16-bit bus takes much less time to accomplish the same job.

Queue Flush for Branch/Jump Instructions

A pre-fetch queue usually speeds up instruction execution time by providing instructions to the Execution Unit in a timely manner. However there is a penalty which goes with the queue when a successful branch or jump instruction is executed. The queue has to be flushed, Program Counter to be reloaded, and new instructions to be fetched. A 16-bit bus helps to fill up the queue much faster. This is critical to system performance since Branch/Jump instructions are the most frequently used instructions in general.

Free Up The System Bus

The microcontroller reduces its number of operand fetches in a 16-bit bus, freeing the bus for other devices which share the same bus. In system which has a DMA Controller or Slave Processor sharing the same memory space with the microcontroller, the less usage of the memory bus will enhance system performance.

Let us look at a sample program to calculate the differences in execution time between an 8 and a 16-bit bus. In the typical 16-bit design example above, there is a look-up table residing in the EPROM. A look-up table is a quick way for the program to provide an output to an I/O device based on the input value without getting into complex mathematical operations. The following program, which is published in Intel application note AP-248, does table look-up and interpolation.

Assuming the 80C196 queue is always full, to execute the following code takes 128 state times in a 16-bit bus. In an 8-bit bus, it takes 32 more state times just to fetch the codes and data, not including the time the microcontroller waits for the queue to be filled. The estimated performance penalty for an 8-bit bus in this application is at least 25%, and will certainly be more in the actual run time environment. The published statement from Intel is that it is difficult to measure the 8-bit bus performance penalty, but has shown to be up to 30%, depending on the instruction mix.

The 16-bit bus design will increase the system performance, especially for microcontrollers which usually don't have internal program cache or a pre-fetch pipeline queue to lessen the penalty caused by the bottle neck on the memory bus. The 80C196 has an internal 4 byte queue. This helps execution time but bus width still remains the critical factor.

Table Look-up and Interpolation

RSEG at 22H

```

IN-VAL:      dsb      1      ;Actual Input Value
TABLE_LOW:   dsw      1
TABLE_HIGH:  dsw      1
IN_DIF:      dsw      1      ;Upper Input-Lower Input
IN_DIFB:     equ      IN_DIF  ;byte
TAB_DIF:     dsw      1      ;Upper Output- Lower Output
OUT:         dsw      1
RESULT:      dsw      1
OUT_DIF:     dsl      1      ;Delta Out

```

CSEG at 2080H

LD SP, #100h

Look:

```

LDB AL, IN_VAL      ;Load temp with Actual Value
SHRB AL, #3         ;Divide the byte by 8
ANDB AL, #1111110B ;Insure AL is a word address
                    ;This effectively divides AL by 2
                    ;so AL = IN_VAL/16
LDBZE AX, AL        ;Load byte AL to word AX
LD TABLE_LOW, TABLE [AX]
                    ;TABLE_LOW is loaded with the value
                    ;in the table at table location AX
LD TABLE_HIGH, (TABLE+2) [AX]
                    ;TABLE_HIGH is loaded with the value
                    ;in the table at table loc. AX+2
                    ;(The next value in the table)
SUB TAB_DIF, TABLE_HIGH, TABLE_LOW
                    ;TAB_DIF=TABLE_HIGH - TABLE_LOW
ANDB IN_DIFB, IN_VAL, #0FH
                    ;IN_DIFB=least significant 4 bits of
                    ;IN_VAL
LDBZE IN_DIF, IN_DIFB
MUL OUT_DIF, IN_DIF, TAB_DIF
                    ;Output_difference =
                    ;Input_difference * Table_difference
SHRAL OUT_DIF, #4   ;Divide by 16 (2**4)
ADD OUT, OUT_DIF, TABLE_LOW
                    ;Add output difference to output
                    ;generated with truncated IN_VAL as
input
SHRA OUT, #4        ;Round to 12-bit answer
ADDC OUT, ZERO      ;Round up if Carry = 1

```

No_Inc:

```

ST OUT, RESULT      ;Store OUT to RESULT
BR Look             ;Branch to "Look"

```

CSEG at 2100h

Table:

```

DCW 0000H, 2000H, 3400H, 4C00H ;A random function
DCW 5D00H, 6A00H, 7200H, 7800H
DCW 7B00H, 7D00H, 7600H, 6D00H
DCW 5D00H, 4B00H, 3400H, 2200H
DCW 1000H

```


**PSD3XX
Solution
for 16-Bit
Microcontroller**

In this section, we will see how a single PSD302 is able to replace all the basic building blocks as shown in the design example in Figure 1. As seen from the block diagram (Figure 2.), the PSD302 provides the following functional blocks:

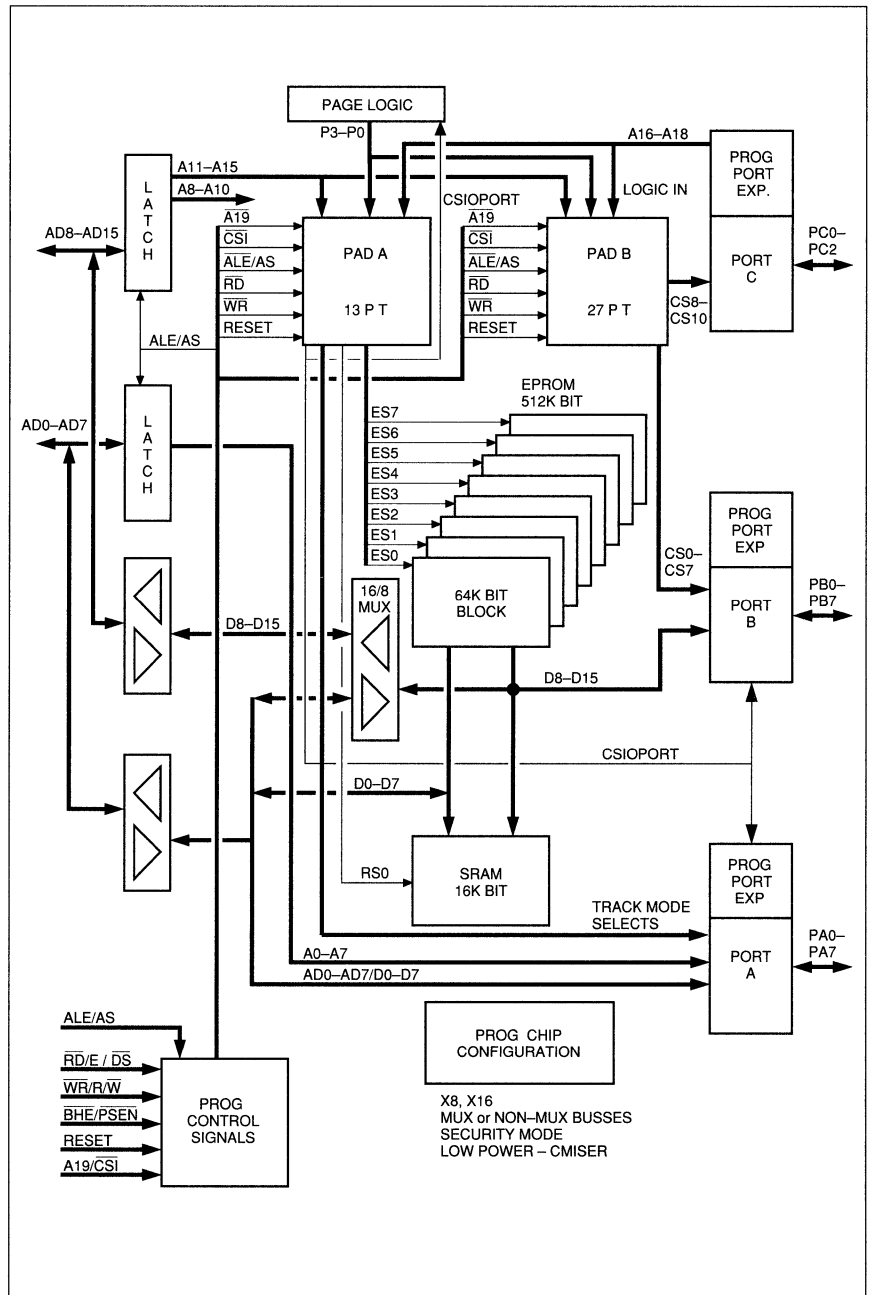
- ❑ 64K bytes EPROM, as 64K x 8 or 32K x 16
- ❑ 2K bytes SRAM, as 2K x 8 or 1K x 16, expanding the microcontroller's internal scratch SRAM
- ❑ Address latches/data buffers, bus interface to most microcontrollers.
- ❑ Programmable Address Decoder (PAD); provides PAL type function: 18 inputs, 24 outputs and 40 product terms.
- ❑ Port A: an 8-bit port, each bit can be configured as :
 - I/O line
 - latched address output (A0–A7)
 - track AD0/AD7 as I/O lines in track mode for shared access.
 - data port D0/D7 in non-multiplexed mode
 - CMOS or open drain output
- ❑ Port B: an 8-bit port, each bit can be configured as :
 - I/O line
 - chip select or logic replacement output from the PAD
 - D8–D15 in non-multiplexed mode
 - CMOS or open drain output

- ❑ Port C: 3-bit port, each bit can be configured as input to or output from the PAD
- ❑ Page Register: a 4-bit Page Register for bank switching
- ❑ A19/ $\overline{\text{CSI}}$ input pin for power down configuration

Figure 3 is the schematic of the design example with the PSD302. Not all the functions of the PSD3XX are utilized in this example. The Page Register is not used since the INST signal from the 80C196 can be easily included in the PAD for page decoding (for design with the Page Register, see WSI Application Note 015). The internal EPROM and SRAM of the PSD302 replaces U5, U6, and U7 in Figure 1. Port A is configured as an I/O port to replace U3, the I/O latch. The PAD provides decoding functions for all the chip selects, as well as the READY and BWIDTH inputs to the microcontroller. Please note the PSD302 is able to provide a 16-bit SRAM for faster data accesses.

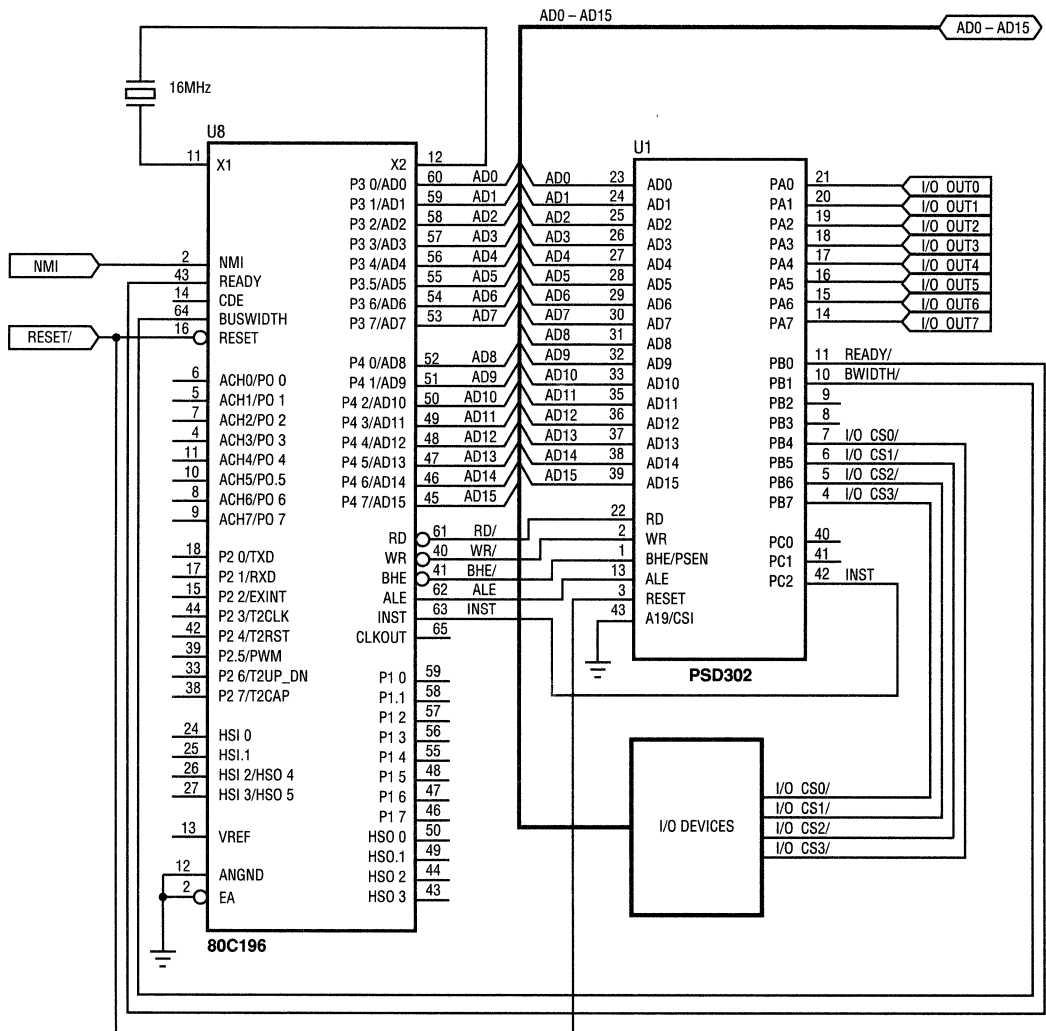
In this application the PSD302 is configured to operate in a 16-bit, multiplexed mode. The PAL equations are programmed into the PAD. Depending on the particular bus cycle, the PSD302 latches the microcontroller address, determines which device is to be enabled, and provides data output for a read cycle. If it is an I/O bus cycle, either Port A is enabled or one of the I/O_CS lines are activated. At the same time, the appropriate READY and BWIDTH signals are generated.

Figure 2.
PSD302
Block
Diagram



1

Figure 3.
Design Example
with PSD302



**PSD3XX
Solution
for 16-Bit
Microcontroller
(Cont.)**

WSI supplies PSD users with easy to use software tools and programming devices. MAPLE software, which is PC based, enables designers to configure the PSD3XX. Some of the computer screen configuration displays for this design

example are shown in Figure 4. Figure 4A is the address map decode for the EPROM, SRAM and Port A. Figure 4B is the truth table input for the READY signal.

**Figure 4A.
PSD3XX
Address
Map**

ADDRESS MAP														
	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMT START	SEGMT STOP	FILE START	FILE STOP	FILE NAME
ES0	N	0	X	X	0	0	0	N	N			0	1FFF	TEST.HEX
ES1	N	X	X	X	0	0	1	N	N			2000	3FFF	TEST.HEX
ES2	N	1	X	X	0	1	0	N	N			4000	5FFF	TEST.HEX
ES3	N	1	X	X	0	1	1	N	N			6000	7FFF	TEST.HEX
ES4	N	1	X	X	1	0	0	N	N			8000	9FFF	TEST.HEX
ES5	N	1	X	X	1	0	1	N	N			A000	BFFF	TEST.HEX
ES6	N	1	X	X	1	1	0	N	N			C000	DFFF	TEST.HEX
ES7	N	1	X	X	1	1	1	N	N			E000	FFFF	TEST.HEX
RS0	N	0	X	X	0	1	0	0	0			N/A	N/A	N/A
CSP	N	0	X	X	0	1	0	1	0			N/A	N/A	N/A

ALIAS: A18 = INST →

Fill in A19 – A12 (Binary) or SEGMT START (Hex); and FILE (START, STOP)

FILE NAME, P3..P0, and ALE/AS. Use SPACEBAR to erase any field value.

F1 – Return to Main Menu F2 – Temporary Exit to DOS F3 – Go to Help

Cursor – UP: ↑ Down: ↓ Left Col: ← Right Col: → Right – F4 Left – F5

PART NAME: PSD302

C:\WSI\OLDMAP

**Figure 4B.
READY
Signal
Truth Table**

PORT B			CHIP SELECT DEFINITION READY											
PIN	CS//O	CMOS/OD	A18	A17	A16	A15	A14	A13	A12	A11	RD	WR	ALE	P3
PB0	CS0	CMOS												
PB1	CS1	CMOS												
PB2	CS2	CMOS												
PB3	CS3	CMOS												
PB4	CS4	CMOS												
PB5	CS5	CMOS												
PB6	CS6	CMOS												
PB7	CS7	CMOS												

ALIAS: A18 = INST →

CS definition is the NOR of the product terms (rows). Enter 1 to select High signal, 0 to select Active Low signal, X to mean "don't care", SPACEBAR to erase. Enter values in columns relevant to your application; leave other columns untouched.

F1 – Return to PORT B Menu

Cursor – Up: ↑ Down: ↓ Left: ← Right: →

PART NAME: PSD302

C:\WSI\OLDMAP

**PSD3XX
Solution
for 16-Bit
Processor with
Non-Multiplexed
Bus**

A PSD3XX can be configured to operate in the 16-bit mode with a non-multiplexed bus. In this case, the microcontroller address lines A0–A15 are tied to AD0–AD15 inputs of the PSD3XX; Port A and B of the PSD3XX are then configured as data ports, connecting to data bus D0–D15. In applications where the EPROM space in a PSD3XX is not enough, or a large amount of I/O lines and chip selects are needed, two PSD3XXs will provide a viable solution. Connecting two PSD3XXs to a microcontroller needs special consideration.

Figure 5 shows a basic design of a 68302 microcontroller interfacing to two PSD312s. The implementation is fairly straightforward; the two PSD302's are configured to work in 8-bit non-multiplexed mode. The first PSD302 (U2) occupies the even bank of the memory space of the 68302; the second PSD302 (U3) occupies the odd bank. The 68302 has no A0 in the address bus; it depends on signals UDS/ and LDS/ (Upper and Lower Data Strobe) to control the flow of data on the data bus as shown in Table 2.

**Table 2.
68302 Byte
Enable**

<i>UDS/</i>	<i>LDS/</i>	<i>D8–D15</i>	<i>D0–D7</i>
Low	Low	Enabled	Enabled
Low	High	Enabled	Disabled
High	Low	Disabled	Enabled
High	High	Disabled	Disabled

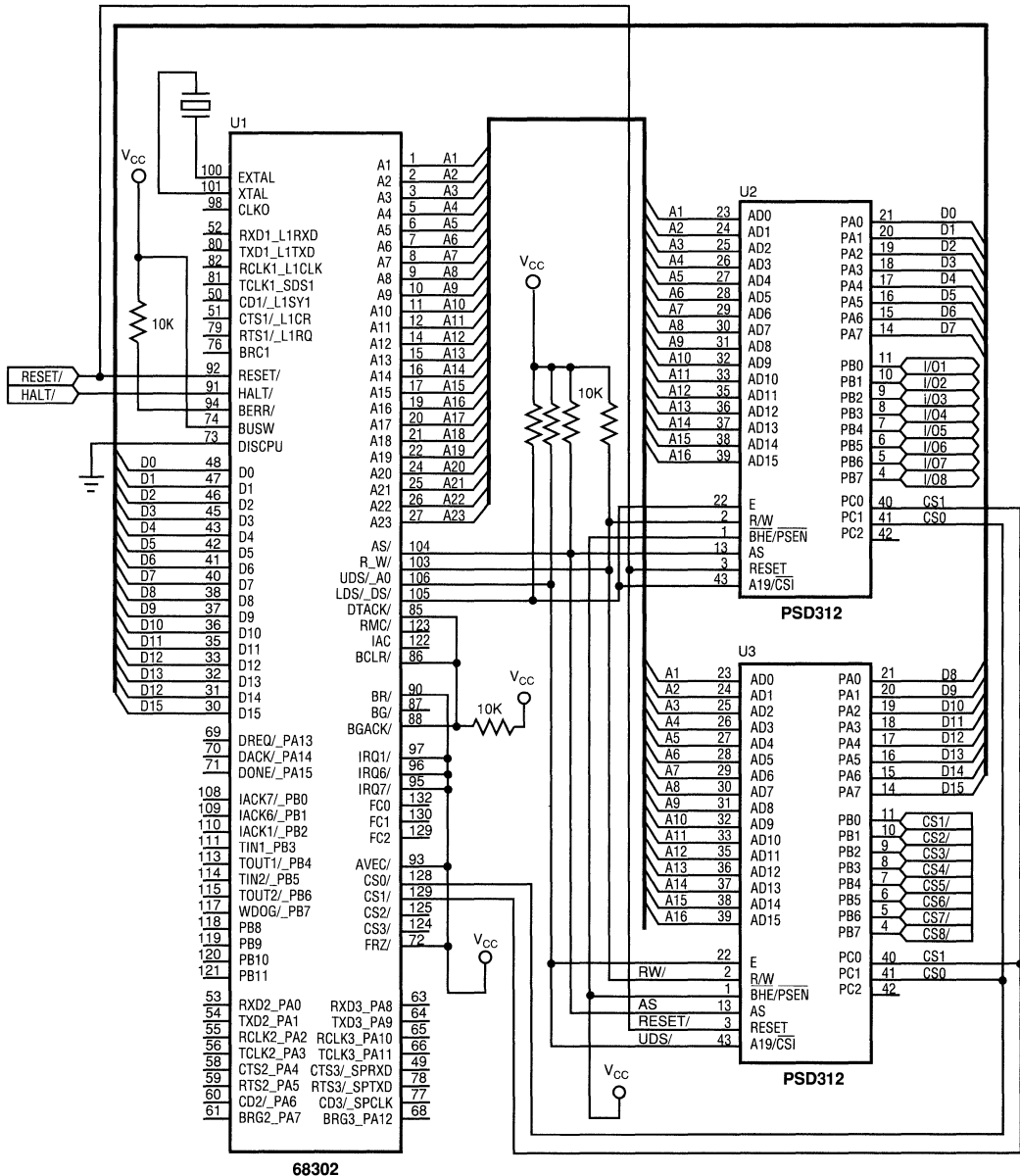
The above table is also true for most other microcontrollers. Some use different signal names, such as HBE/ for UDS/ and A0 is equivalent to LDS/. The decoding for bank select is the same for both cases. The following points must be considered when configuring the PSD3XX for this type of application:

- Address inputs to the PSD3XX have to shift right by one. Address line A1 connects to AD0 pin of the PSD3XX and so on. For processors which have A0, A0 is no longer used as address input.
- Provide bank select signals to the appropriate PSD3XX for proper bank decoding. The even bank PSD3XX must include signal LDS/ as input to the PAD, and the odd bank PSD3XX requires the signal UDS/. These signals can be connected to Port C or the A19/CSI pin in order to be routed as PAD inputs.

- While inside the MAPLE software during PSD3XX configuration, the address map decode of the EPROM, SRAM, I/O port must also reflect the shift of the address inputs.
- The codes of the user's program have to be split into two files, one for the even bank PSD3XX and one for the odd bank PSD3XX.

Figure 5.
PSD3XX
Interface
to 68302

1



**PSD3XX
Solution
for 16-Bit
Microcontroller
with Non-
Multiplexed
Bus**

Figure 6 shows the address map of the odd bank PSD3XX. In the map table, the columns A19, A17, A16 are input signals of UDS, CS0 and CS1. Since this is the decoding for the odd bank, UDS (column A19) has to be low for any of the PSD3XX devices to be enabled.

Furthermore, the CS0 selects the EPROM and CS1 selects SRAM and I/O Port. The chip select logic of the 68302 also generates the programmed amount of wait state internally.

The columns A15, A14, A13 in the address map are actually A16, A15 and A14 after the input address lines to the PSD3XX are shifted by one. Entries to the SEGMENT START/STOP or FILE START/STOP columns must also change to reflect the shift of the address lines. For example, the top address of the EPROM (128K bytes) was 1FFFF, and is now 0FFFF after the shift.

**Figure 6.
Address Map,
Odd Bank
PSD3XX**

ADDRESS MAP

	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMT START	SEGMT STOP	FILE START	FILE STOP	FILE NAME
ES0	0	X	0	1	0	0	0	N	N			0	1FFF	ODD.HEX
ES1	0	X	0	1	0	0	1	N	N			2000	3FFF	ODD.HEX
ES2	0	X	0	1	0	1	0	N	N			4000	4FFF	ODD.HEX
ES3	0	X	0	1	0	1	1	N	N			6000	7FFF	ODD.HEX
ES4	0	X	0	1	1	0	0	N	N			8000	9FFF	ODD.HEX
ES5	0	X	0	1	1	0	1	N	N			A000	BFFF	ODD.HEX
ES6	0	X	0	1	1	1	0	N	N			C000	DFFF	ODD.HEX
ES7	0	X	0	1	1	1	1	N	N			E000	FFFF	ODD.HEX
RS0	0	X	1	0	0	0	0	0	0			N/A	N/A	N/A
CSP	0	X	1	0	1	0	0	0	0			N/A	N/A	N/A

ALIAS: A19 = UDS →

Fill in A19 – A12 (Binary) or SEGMENT START (Hex); and FILE (START, STOP) FILE NAME, P3..P0, and ALE/AS. Use SPACEBAR to erase any field value.

F1 – Return to Main Menu F2 – Temporary Exit to DOS F3 – Go to Help
Cursor – UP: ↑ Down: ↓ Left Col: ← Right Col: → Right – F4 Left – F5

Conclusion

After going through the design examples with the PSD3XX, it is not difficult to see the advantages the PSD3XX family offers over designs with discrete ICs. Besides providing 16-bit performance, PSD3XX devices are able to replace 7 ICs in the 80C196 example. This not only reduces the board size dramatically but also provides benefits such as cost reduction in board manufacturing, higher product reliability, lower power consumption and reduced component cost.

Other PSD3XX advantages over the discrete component design include the power down mode to reduce power consumption when the microcontroller is idle. The security feature protects the code stored in the EPROM from illegal copy. The flexibility, programmability, and ease of use which come with the PSD3XX truly make it an optimal solution for 16-bit embedded applications.





Programmable Peripheral Application Note 021

Interfacing The PSD3XX To The MC68HC16 and The MC68300 Family of Microcontrollers

By Ching Lee

1

Introduction

The PSD3XX devices are user-configurable microcontroller peripherals which offer an ideal solution for embedded control applications. The PSD3XX family provides basic building blocks to microcontroller based designs including I/O ports, logic replacement, programmable address decoder (PAD), Memory Page Register,

2K bytes of SRAM and up to 128K bytes of EPROM. Please consult Application Note 011 for detail features and operations.

In this Application Note we will demonstrate how the PSD3XX interfaces to the 16-bit MC68HC16 and the MC68300 family of 32-bit microcontrollers from Motorola.

Typical MC68331 Design

The MC68300 family includes micro-controllers such as the MC68330, MC68331, MC68332 and MC68340. These devices share a common MC68020 CPU core. Although the MC68HC16 is not a member of the family, it does have a MC68300 type bus interface and timing. The MC68331 will be chosen as the microcontroller used with the PSD3XX

device in this Application Note, but the description applies to other members of the group as well.

A typical MC68331 design consists of two EPROMs, two SRAMs, an I/O block and glue logic. The complexity of the I/O block and glue logic depends on the application. Figure 1 shows the block diagram of such a design.

The MC68331 Bus Interface

Before interfacing the PSD3XX to the MC68331, we have to understand the MC68331 bus and the bus features. Area's of interest to the PSD3XX interface are discussed in the following sections.

Address Bus

The MC68331 bus has 24 non-multiplexed address lines (A0 – A23). Address lines A19 – A23 can be selected either as address lines or as chip select signals ($\overline{CS6}$ – $\overline{CS10}$) at reset time.

Data Bus

The data bus (D15 – D0) is a non-multiplexed bus which transfers 8 or 16 bits of data. The processor supports byte, word, and long word operands.

The MC68331 transfers the even data byte (with A0 = 0) through D15 – D8, and the odd byte (with A0 = 1) through D7 – D0. Table 1 shows the different data transfer cases for a 16-bit bus, and the status of the control signals involved in the bus cycles.

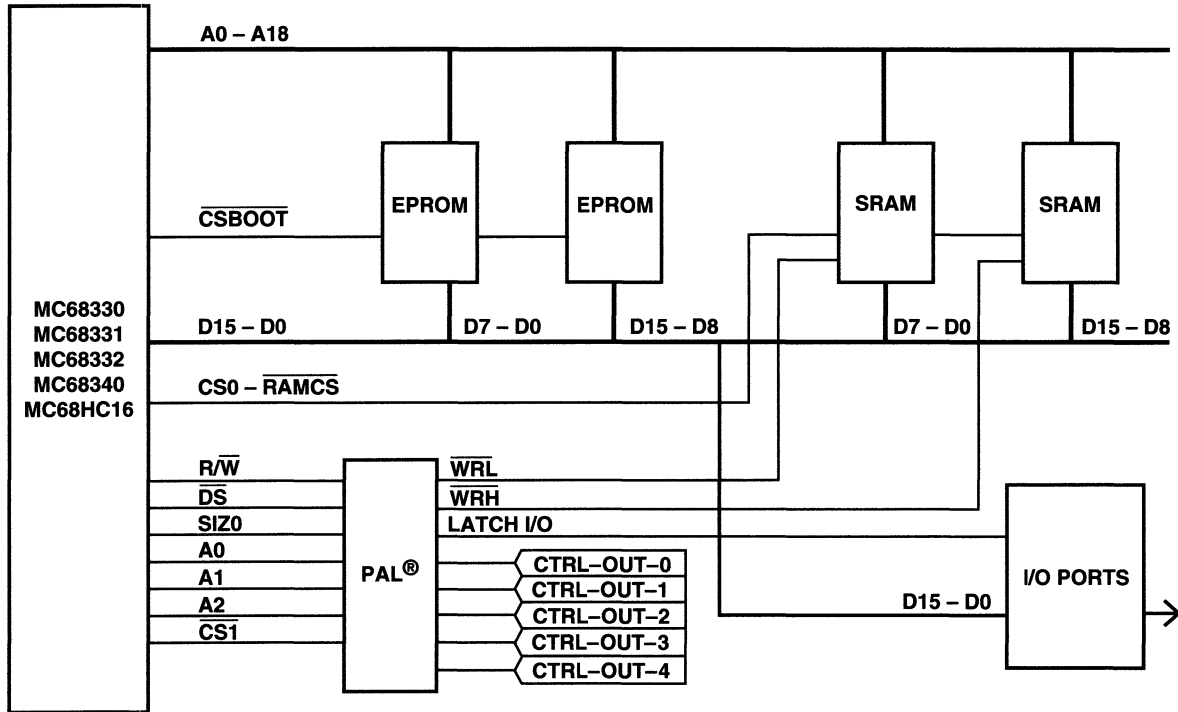
For byte transfer, the positioning of the byte is determined by address A0. OP0 refers to the most significant byte of a word operand, and OP1 is the least significant byte. Operands in parentheses are ignored during read cycles, but are driven by the processor during write cycles. Misaligned words are not supported by the MC68331.

**Table 1.
MC68331 Data
Transfer On
16-Bit (Word)
Port**

<i>Transfer</i>	<i>SIZ1</i>	<i>SIZ0</i>	<i>A0</i>	<i>DSACK1</i>	<i>DSACK2</i>	<i>D15 – D8</i>	<i>D7 – D0</i>
Upper Byte (even)	0	1	0	0	X	OP0	(OP0)
Lower Byte (odd)	0	1	1	0	X	(OP0)	OP0
Word (aligned)	1	0	0	0	X	OP0	OP1
Long Word (aligned)	0	0	0	0	X	OP0	OP1



Figure 1. MC68331 System Block Diagram



The MC68331 Bus Interface (Cont.)

Chip Select Logic

The MC68331 provides 12 chip select output signals (CSBOOT and CS0 – CS10). CS0 – CS10 are multiplexed with other signals and default to chip select mode during reset.

The chip select signals are user-programmable, flexible and powerful. The following list outlines some of the options/features which can be programmed into any chip select signal:

1. **Base Address:** Specify base address and block size.
2. **Mode Option:** Select asynchronous/synchronous bus mode.
3. **Byte Option:** Specify upper byte, lower byte, or both.
4. **Read/Write Option:** Specify read or write bus cycle.

5. **Strobe Option:** Specify CS signal to be synchronized with the DS or AS signal.
6. **DSACK Option:** Specify internal/external source of the DSACK signal. If internal DSACK is specified, then select the number of wait states.

After system reset, $\overline{CS0}$ – $\overline{CS10}$ lines are disabled since they should not select any device until the system is configured. But \overline{CSBOOT} has a default reset value such that it can be used to enable a boot PROM, or PSD3XX in this case. The other chip selects are then initialized by the boot program. CSBOOT may or may not be re-programmed to a different value. The reset value of the CSBOOT signal is listed in Table 2.

Table 2.
CSBOOT
Reset
Value

Fields/Option	Reset Values
Base Address	0000 0000
Block Size	1M Byte
Asynchronous/Synchronous Mode	Asynchronous Mode
Upper/Lower Byte	Both Bytes
Read/Write	Read/Write
Strobe	AS
DSACK	13 Wait States
Address Space	Supervisor/User
Interrupt Priority Level	Any Level
Autovector	Interrupt Vector Externally

A Typical MC68331 Design With PSD3XX

As seen in the typical MC68331 design block diagram in Figure 1, the basic building blocks include EPROMs, SRAMs, a PAL®, and I/O port. The PSD3XX will replace all or most of these blocks. Depending on the amount of EPROM and

SRAM space and the I/O port requirement, one or two PSD3XX devices can be used. The two PSD3XX system does have a different interface to the MC68331 than a single PSD3XX design. The two designs will be discussed separately.

The Two PSD3XX Design

Figure 2 shows a two PSD3XX implementation of a typical MC68331 system. In this design the PSD312s replace the EPROMs, the SRAMs, the PAL[®], and the I/O port in the typical design. The two PSD312 devices provide 128K bytes of EPROM and 4K bytes of SRAM. Two PSD313's can be used if more EPROM space is needed.

The configurations of the two PSD312's in this design are as follow:

Bus Width

Each PSD312 is configured to operate in 8-bit non-multiplexed mode. PSD#1 supplies the even data byte to the processor while PSD#2 supplies the odd data byte. Together they appear to the MC68331 as a single 16-bit port.

Port A

Port A in a PSD3XX can be an I/O port, address output latch or as a data port with a non-multiplexed bus. In this design example, Port A is configured as a data port. Port A of PSD#1 is connected to D15 – D8 and PSD#2 is connected to D7 – D0 of the processor.

Port B

Port B can be an I/O port, chip select outputs, or as a data port in a 16-bit non-multiplexed bus. In this design, half of Port B is used to replace the I/O port in the typical design, and the other half is used as logic replacement for the PAL[®]. The two B ports together provide two 4-bit I/O ports, and 8 output control signals.

Port C

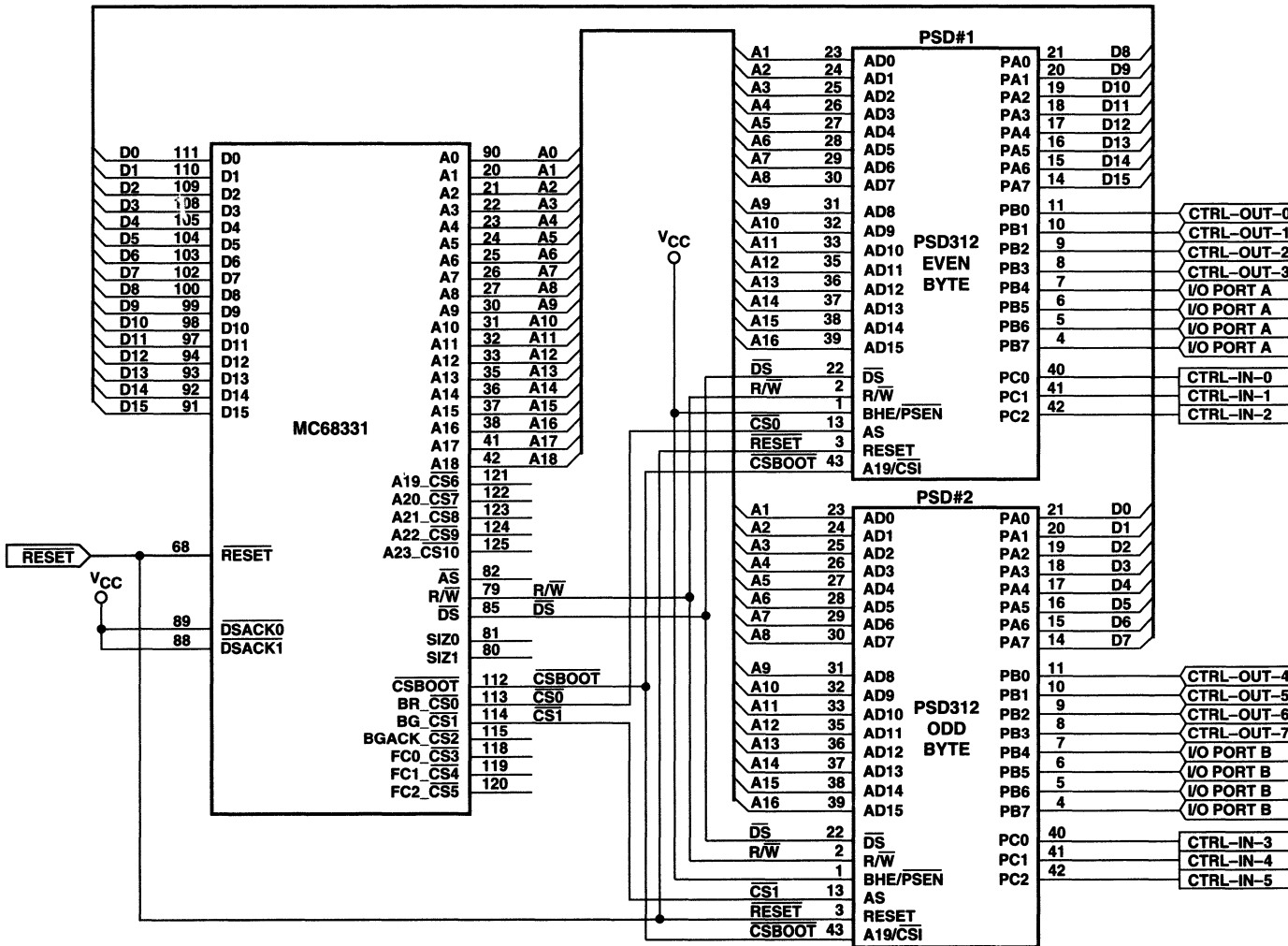
Port C can be used as an address/logic input port to the PAD, or as chip select outputs. In our case, Port C is used as logic input.

Bus Interface

The PSD312's are configured to interface to a bus with a read/write signal ($\overline{R/W}$) and a data strobe signal (\overline{DS}). Since the MC68331 has a non-multiplexed bus, the address strobe (\overline{AS}) input signal is not required by the PSD312 to latch the address lines internally. Instead, the \overline{AS} and the CS1/A19 pins are used as address input pins to select the internal PSD312 sections. The 3 signals which select the PSD312's are \overline{CSBOOT} , $\overline{CS0}$ and $\overline{CS1}$. The \overline{CSBOOT} selects the EPROMs. The $\overline{CS0}$ selects the even byte SRAM and I/O Port B; $\overline{CS1}$ selects the odd byte only. No wait states are required if the MC68331 is running at a 16 MHz system clock. Please refer to Figure 2 for the bus interface connections.

After reset, \overline{CSBOOT} selects the PSD312's to run the boot program. During this time, the 3 chip select lines can be re-programmed to reflect the correct address range and wait state. This has to be done before the SRAMs can be accessed. Table 3 shows some typical option values for this application. There is no 4 KB block size option, so 8 KB is assigned to the SRAM.

Figure 2. MC68331/Two PSD312 Interfacing



**Table 3.
Chip Select
Option Value**

<i>Option</i>	<i>CSBOOT</i>	<i>CS0</i>	<i>CS1</i>
Base Address	000000	040000	040000
Block Size	256 KB	8 KB	8 KB
Asynchronous/Synchronous Mode	Asynchronous	Asynchronous	Asynchronous
Upper/Lower Byte	Both Bytes	Upper Byte	Lower Byte
Read/Write	Read	Read/Write	Read/Write
Strobe	AS	AS	AS
DSACK	0 Wait State	0 Wait State	0 Wait State

**The Two
PSD3XX Design
(Cont.)**

Bus Interface (Cont.)

Figure 3 shows the Address Map from the PSD#1 configuration file generated by the WSI Maple configuration software. In the Address Map truth table, the A19 column is the CSBOOT input, and the Q.F. AS column is the CS0 input. EPROM is selected when CSBOOT is low; SRAM or

I/O Port B is selected when CS0 is low. The addresses in FILE STRT/FILE STOP columns are shifted right by one to reflect the fact that the A15 – A13 input pins on the PSD312's are connected actually to the A16 – A14 of the MC68331.

**Figure 3.
Address Map
Truth Table**

ADDRESS MAP														
	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMT STRT	SEGMT STOP	FILE STRT	FILE STOP	Q. F. AS
ES0	0	X	X	X	0	0	0	N	N			0	1fff	1
ES1	0	X	X	X	0	0	1	N	N			2000	3fff	1
ES2	0	X	X	X	0	1	0	N	N			4000	5fff	1
ES3	0	X	X	X	0	1	1	N	N			6000	7fff	1
ES4	0	X	X	X	1	0	0	N	N			8000	9fff	1
ES5	0	X	X	X	1	0	1	N	N			a000	bfff	1
ES6	0	X	X	X	1	1	0	N	N			c000	dfff	1
ES7	0	X	X	X	1	1	1	N	N			e000	ffff	1
RS0	1	X	X	X	0	0	0	0	0			N/A	N/A	0
CSP	1	X	X	X	0	0	0	0	1			N/A	N/A	0

END

The Two PSD3XX Design (Cont.)

Power Down Mode

The PSD3XX's power down mode is particularly useful in a system which uses a PSD3XX mostly as a boot PROM. The power down mode is controlled by the input pin A19/CSI. This pin can be configured by MAPLE as address input (A19) or as chip select input (CSI). To implement the power down mode to the two-PSD312 design, the following changes are required:

- ❑ Configure the A19/CSI pin by MAPLE as the $\overline{\text{CSI}}$ pin. Connect the pin to the $\overline{\text{CSBOOT}}$ signal. The PSD3XX's are normally in power down mode except when $\overline{\text{CSBOOT}}$ is asserted.
- ❑ The PSD3XX has a 10 ns hold time requirement on the CSI input with reference to the trailing edge of the $\overline{\text{DS}}$ signal. The MC68331 does not provide this hold time; designers have to delay the $\overline{\text{CSBOOT}}$ signal to meet this requirement.

- ❑ The $\overline{\text{CSBOOT}}$ signal is programmed to have a block size of 512 KB. This will cover both the EPROM and SRAM space. When accessing the SRAM, the $\overline{\text{CSBOOT}}$ signal is asserted to take the PSD3XX out of power down mode. In power down mode, the Port B I/O ports will maintain their output values but the chip select output signals will be inactive.
- ❑ When the $\overline{\text{CSBOOT}}$ is used as CSI input to the PSD3XX, the access time from CSI valid to data out is 130 ns (PSD302-12), 10 ns more than the normal address valid to data out time. This requires $\overline{\text{CSBOOT}}$ and CS0 – CS1 to be programmed with one wait state.

1

The Single PSD3XX Design

The single PSD3XX design is for applications which need less EPROM and SRAM space. Figure 4 illustrates the schematic of a PSD302 interfacing to the MC68331. The PSD302 provides 64 KB of EPROM and 2 KB of SRAM; Port A and Port B are configured as data ports for the MC68331. Port C generates the I/O latch signal for the I/O port and two chip select signals.

The single PSD3XX interface to the MC68331 is different from the two PSD3XX design. In order for the PSD302 to operate in the 16-bit mode, it needs a Byte High Enable ($\overline{\text{BHE}}$) signal. If the design has a $\overline{\text{BHE}}$ signal available (generated from decoding to A0 and SI20 signals from the MC68331), connect it directly to the PSD302's $\overline{\text{BHE}}$ pin. $\overline{\text{CSBOOT}}$ or other high address bits are then used to select the PSD302.

If there is no $\overline{\text{BHE}}$ signal available, the $\overline{\text{CSBOOT}}$ signal can be programmed to provide this function. After reset, the $\overline{\text{CSBOOT}}$ signal operates as a chip select signal with initial values shown in Table 2. It will serve temporarily as the $\overline{\text{BHE}}$ input to

the PSD302 until the $\overline{\text{CSBOOT}}$ is programmed as a $\overline{\text{BHE}}$ signal. The programming should be done shortly after reset with the value listed in Table 4.

Now that the $\overline{\text{CSBOOT}}$ signal is used as the $\overline{\text{BHE}}$ signal, the PSD302 needs other means to select the internal device. Other chip select signals from the MC68331 cannot be used since they are inactive at the time of boot up. Address lines A18 – A19 are used instead as decoding address inputs to the PSD302 as shown in Figure 4. For example, the EPROM is enabled if A18 – A19 are equal to 00H; SRAM and I/O Ports are enabled if A18 – A19 are equal to 01H. Depending on the application, any other high address bits can be used for this purpose.

The PSD302 is configured to operate in the 16-bit mode. The high byte is coming from Port B and is connected to D7 – D0 on the MC68331 data bus. Table 5 shows which port and byte it is driving for different bus cycles. Please note the low byte/high byte definition in the PSD302 Data Book is the opposite to that defined in the MC68331 User's Manual.

Figure 4. MC68300/PSD302 Interface

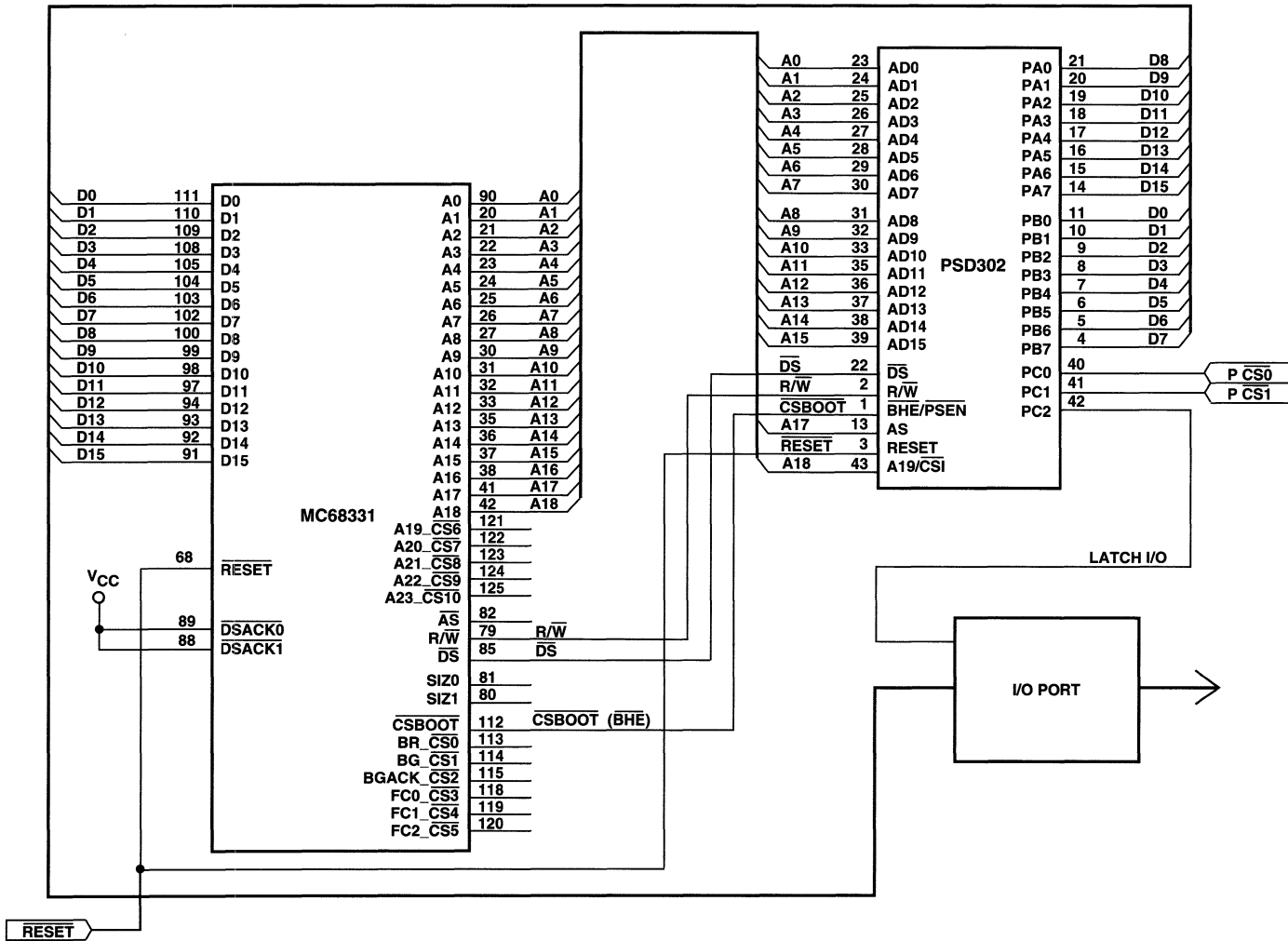


Table 4.
CSBOOT as
BHE Signal

Fields/Option	CSBOOT Values as BHE
Base Address	000000
Block Size	128 K Byte
Asynchronous/Synchronous Mode	Asynchronous Mode
Upper/Lower Byte	Lower Byte (Odd Byte)
Read/Write	Read/Write
Strobe	AS
DSACK	0 Wait State

1

Table 5.
PSD302
Data Ports

A0	BHE (CSBOOT)	Port A (Even Byte)	Port B (Odd Byte)
0	0	D15 – D8	D7 – D0
0	1	D15 – D8	Tri-State
1	0	Tri-State	D7 – D0
1	1	Tri-State	Tri-State

Conclusion

As we have seen from the two PSD3XX design examples, the PSD3XX's are able to replace at least 6 ICs in the typical 16-bit MC68331 design. The PSD3XX programmable microcontroller peripheral not only provides a cost effective solution to embedded applications, but it also reduces printed circuit board space, power consumption and offers code protection from unauthorized access.



Programmable Peripheral Application Note 022

Using WSI's PSD3XX Programmable Microcontroller Peripheral Family with 80C31/80C51 Microcontrollers

By Dan Kinsella

Introduction

The 80C51 microcontroller family is composed of several versions from different manufacturers that are variations of the basic 80C51 architecture. Different functions on-chip, a variety of speeds, packages, etc. are all available while the 80C51 instruction set is maintained on all variations. Because of the wide variety of features found with the 80C51 devices, as well as the usefulness of the general architecture itself, the 80C51 product group has become one of the most widely used microcontroller families in the world. The purpose of this application note is to provide some background and implementation ideas to designers using the 80C51 microcontroller family along with the WSI PSD3XX family. A simple 80C31 system will be examined, as well as more complex systems where memory map issues become important.

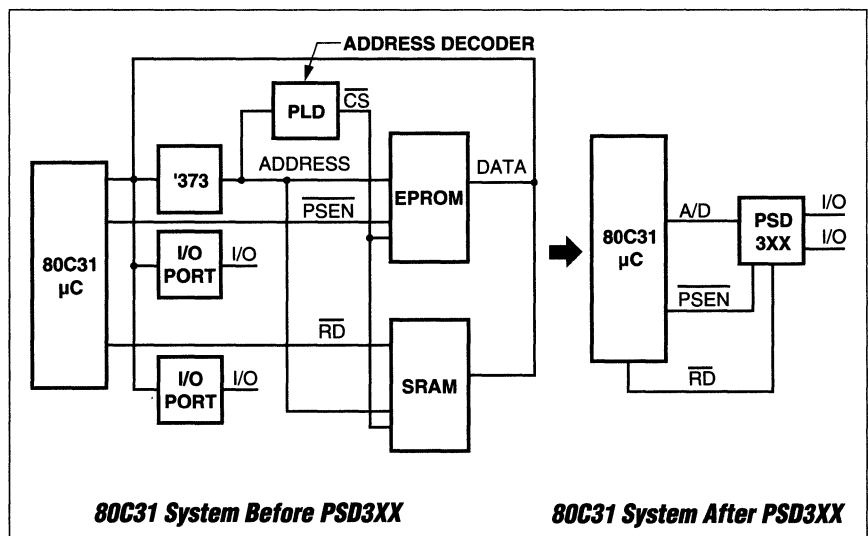
The 80C51 family can be divided into two basic memory usage types: internal memory usage only (80C51, 8751, etc.)

and external memory accesses needed (80C31, 80C51 with external accesses, etc.). The PSD3XX family helps solve the problem of system size and cost by integrating the typical devices used with external accesses (logic, SRAM, EPROM, etc.) onto a single chip. Figure 1 illustrates a typical 80C31 system and how it is dramatically simplified and improved by using a PSD3XX device.

The PSD3XX device simplifies the system by enabling the designer to integrate I/O ports, the microcontroller to peripherals or memories interface logic, EPROM code storage, and scratch pad SRAM data storage into a single chip. Design changes that are sometimes required due to changing market conditions are often easily and quickly implemented in the reprogrammable PSD3XX without adding extra chips to the system or making board layout changes.

1

Figure 1.



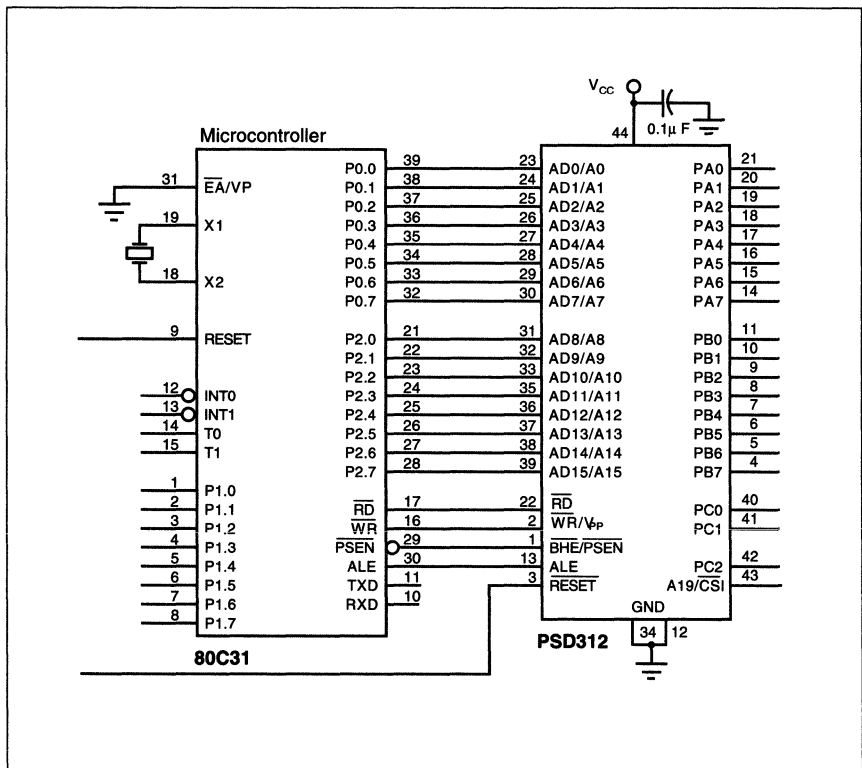
The 80C31 Family

A number of 80C31 versions are offered from several suppliers that include different functions on-chip, but versions such as the 80C32 have different internal memory sizes as well. Each of these versions has retained the core architecture and external memory access requirements of the basic 80C31.

The 80C51 family memory model is composed of two separate memory spaces or allocations. Program space is intended for storage of the program control code (usually in EPROM) and data space is intended for storage of temporary or changeable information (usually in SRAM). Data space is also where most memory mapped I/O is located. In external memory operation, the 80C31 uses the PSEN signal to access program memory and the RD signal accesses the data memory. The PSD3XX's programmability provides for this memory model (called "separate space") as well as other memory models.

Program and data space do not always need to be separated. In fact, most controller architectures make no distinction between program and data space. If this memory model is desired, the PSD3XX will internally OR the PSEN and RD signals together and EPROM, SRAM and I/O will be available in the same memory space (called "combined space"). In "combined space", program code can be stored in EPROM or SRAM. This can be very important if the user has program code that is downloaded into SRAM, or if the system uses lookup table contents that depend on system parameters not known until the system is operating. Anytime the program code needs to be easily changeable, SRAM is a convenient way to store it. Figure 2 illustrates how the PSD3XX would be connected to the 80C31 and tables 1 and 2 convey some of the choices made via the PSD Development System Software to configure the PSD3XX as "separate" or "combined" address space.

Figure 2.
80C31/PSD312
Interface
Simplicity



**The 80C31
Family
(Cont.)**

Table 1 shows an EPROM segment starting at address 0 and covering the entire 64K program space. The SRAM block and memory mapped I/O are shown in the same memory space. This is

acceptable since they will be controlled by the \overline{RD} signal so there will be no conflict with the EPROM which is controlled by the \overline{PSEN} signal.

**Table 1.
"Separate"
Mode PSD3XX
Choices**

```

***** MAPLE 5.00 *****
PSD PART USED: PSD312
*****PROJECT INFORMATION*****
Project Name :      = 8031 app. note - Separate address space
Your Name :       = Dan Kinsella
Date :           = May, 1992
Host Processor :  = 8031
*****GLOBAL CONFIGURATION*****
Address/Data Mode :      MX
Data Bus Size :        8
Reset Polarity :       HI
Security :            OFF
ALE Polarity :        HI
Using Different READ strobes for Data and Program: Y
Separate Data and Program Address spaces : Y  ----->  Separate Mode

*****READ WRITE CONTROL*****
/RD and /WR
*****ADDRESS MAP*****

      A  A  A  A  A  A  A  A  A  SEGMENT SEGMENT FILE   FILE  File      Page Reg  Q.F
      19 18 17 16 15 14 13 12 11  STRT  STOP  STRT  STOP  Name      3210  ALE

ES0 N  0  0  0  0  0  0  0  N  N  0      1FFF  0000  1FFF  TEST.HEX      N
ES1 N  0  0  0  0  0  0  1  N  N  2000  3FFF  2000  3FFF  TEST.HEX      N
ES2 N  0  0  0  0  0  1  0  N  N  4000  5FFF  4000  5FFF  TEST.HEX      N
ES3 N  0  0  0  0  0  1  1  N  N  6000  7FFF  6000  7FFF  TEST.HEX      N
ES4 N  0  0  0  0  1  0  0  N  N  8000  9FFF  8000  9FFF  TEST.HEX      N
ES5 N  0  0  0  0  1  0  1  N  N  A000  BFFF  A000  BFFF  TEST.HEX      N
ES6 N  0  0  0  0  1  1  0  N  N  C000  DFFF  C000  DFFF  TEST.HEX      N
ES7 N  0  0  0  0  1  1  1  N  N  E000  FFFF  E000  FFFF  TEST.HEX      N
RS0 N  0  0  0  0  0  1  1  0  0  6000  67FF  N/A    N/A    N/A      N
CSP N  0  0  0  0  1  1  0  0  0  C000  C7FF  N/A    N/A    N/A      N

*****END*****

*****ADDRESSES OF I/O PORTS*****
Pin Register of Port A :      C002  Page (Binary):
Direction Register of Port A :  C004
Data Register of Port A :     C006
Pin Register of Port B :      C003
Direction Register of Port B : C005
Data Register of Port B :     C007
Page Register :              C018
*****

```

**The 80C31
Family
(Cont.)**

Table 2 shows the I/O, EPROM and SRAM blocks located throughout the memory map but not overlapping as in the case of the "separate" mode.

If SRAM or I/O is overlapped on the EPROM block in the memory map in the "combined mode", the SRAM or I/O will be accessed in that address range. The EPROM portion that is overlapped becomes inaccessible. In the "separate" mode, EPROM and SRAM can be

overlapped and both are accessible because the PSEN and RD separately enable these blocks to the output. In the "combined" mode, these enables are gated together and SRAM and I/O have priority over the EPROM (for example, if the SRAM is mapped at the same start address as EPROM block 7 (ES7), the first 2K of ES7 will be SRAM access and the rest of the 8K block will be EPROM access).

**Table 2.
"Combined"
Mode PSD3XX
Choices**

```

***** MAPLE 5.00 *****
PSD PART USED: PSD312
*****PROJECT INFORMATION*****
Project Name :      = 8031 APP NOTE - Combined address space
Your Name   :      = DAN KINSELLA
Date       :      = MAY, 1992
Host Processor :    = 8031

*****GLOBAL CONFIGURATION*****
Address/Data Mode :  MX
Data Bus Size   :    8
Reset Polarity  :    HI
Security       :    OFF
ALE Polarity    :    HI
Using Different READ strobes for Data and Program: Y
Separate Data and Program Address spaces : N -----> Combined Mode
*****READ WRITE CONTROL*****
/RD and /WR
*****ADDRESS MAP*****

```

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	FILE	FILE	File	Page	Reg	Q.F
	19	18	17	16	15	14	13	12	11	STRT	STOP	STRT	STOP	Name	3210	ALE	
ES0	N	0	0	0	0	0	0	N	N	0	1FFF	0000	1FFF	TEST.HEX			N
ES1	N	0	0	0	0	1	1	N	N	6000	7FFF	2000	3FFF	TEST.HEX			N
ES2	N	0	0	0	1	0	0	N	N	8000	9FFF	4000	5FFF	TEST.HEX			N
ES3	N	0	0	0	1	0	1	N	N	A000	BFFF	6000	7FFF	TEST.HEX			N
ES4	N	0	0	0	1	1	1	N	N	E000	FFFF	8000	9FFF	TEST.HEX			N
ES5	N							N	N								N
ES6	N							N	N								N
ES7	N							N	N								N
RS0	N	0	0	0	0	0	1	1	0	3000	37FF	N/A	N/A	N/A			N
CSP	N	0	0	0	1	1	0	0	0	C000	C7FF	N/A	N/A	N/A			N

```

*****END*****

*****ADDRESSES OF I/O PORTS*****
Pin Register of Port A :      C002 Page (Binary):
Direction Register of Port A : C004
Data Register of Port A :    C006
Pin Register of Port B :      C003
Direction Register of Port B : C005
Data Register of Port B :    C007
Page Register :              C018
*****

```

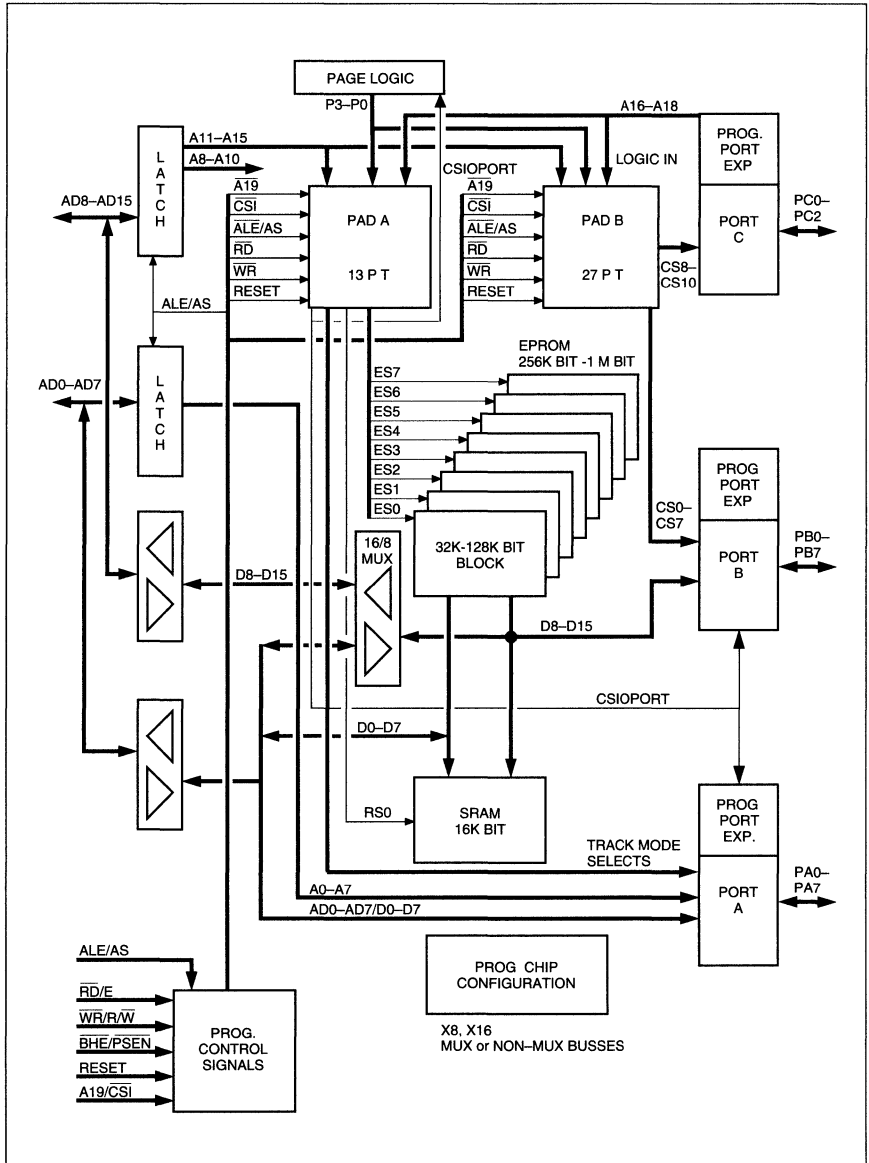


**PSD3XX
Architecture**

Figure 3 illustrates a more detailed block diagram of the PSD3XX family. Since the PSD30X can be configured to operate either x8 or x16, it has the flexibility to

operate with 8 or 16-bit microcontrollers. The PSD31X devices are used in 8-bit-only applications and are available at lower cost.

**Figure 3.
PSD3XX
Architecture**



1

**PSD3XX
Architecture
(Cont.)**

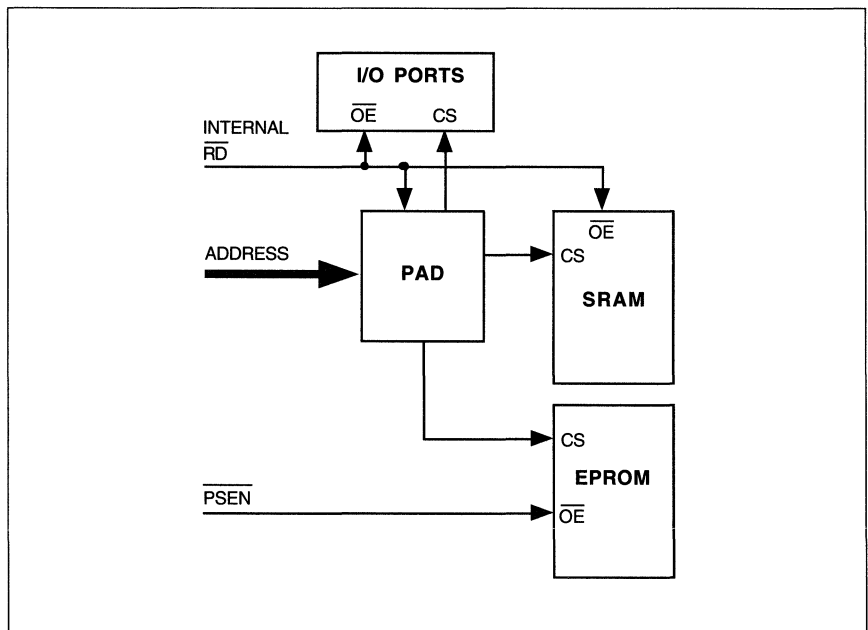
Various EPROM sizes are offered in the PSD3XX series. The PSD3X1 contains 256K bits of EPROM, the PSD3X2 contains 512K bits and the PSD3X3 provides 1 megabits. The PSD3X2 and PSD3X3 devices contain a 4-bit page register to enable additional memory map flexibility. This page register enables memory expansion by a factor of 16. For example, 8-bit microcontrollers like the 80C31 that address only 64K of memory space can now access over 1 meg bytes with either the PSD3X2 or PSD3X3.

Figure 3 illustrates the block configuration of the EPROM. It can be selected as separate blocks that can be scattered throughout the memory space or concatenated into a single block. This feature provides the designer with a great deal of flexibility and efficiency by placing EPROM segments throughout the memory where they are most needed. The SRAM block can also be programmed to appear in any part of the

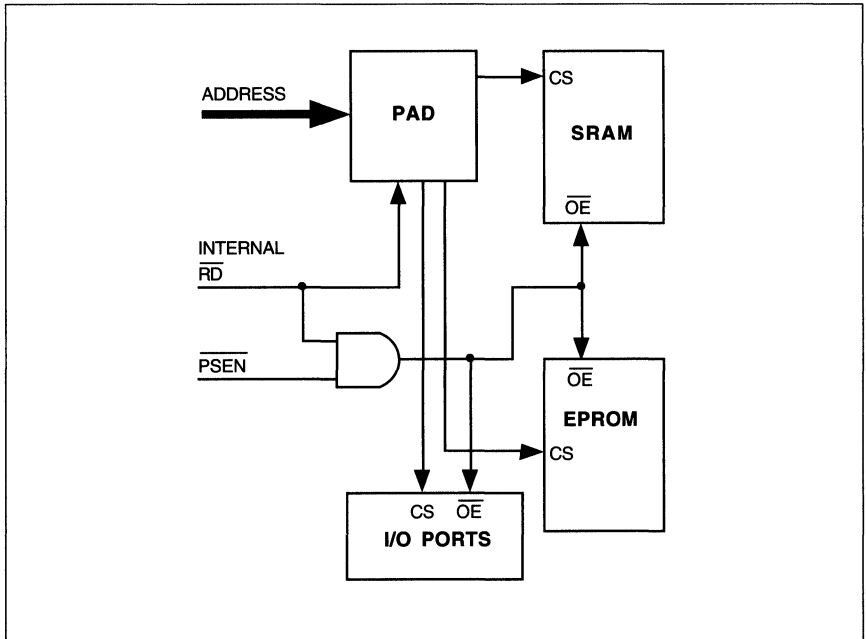
memory map. Some possible examples are shown in Tables 1 and 2. The Table 1 memory map shows the EPROM blocks concatenated together and starting at address 0 as they might look in a simple 80C31 system. Table 2 illustrates the EPROM blocks separated and spread throughout the memory map with the SRAM segment and the memory mapped I/O between EPROM blocks.

Internally the PSD3XX resolves the issue of Combined vs. Separate address spaces by ORing the PSEN and RD signals when Combined space is specified. The EPROM and SRAM segments will both be enabled if either of these signals is present. The address decoder (PAD) will determine which of these is actually active. In the Separate mode, the PSEN will enable only the EPROM output and the RD will enable only the SRAM and I/O ports. Figures 4 and 5 illustrate how this is done.

**Figure 4.
Separate Code
and Data
Address Spaces**



**Figure 5.
Combined
Address Space**



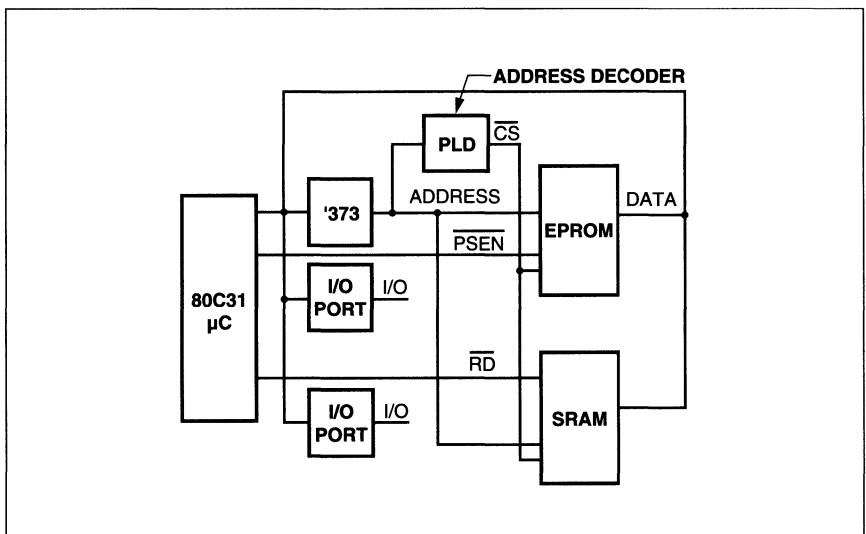
1

**Simple
80C31
Design**

Figure 6 shows the block diagram of a typical 80C31 design. The '373 latch is required to demultiplex the address and data busses. EPROM is used for program control memory and the SRAM is required

for stack extension or scratch pad memory. The Separate memory model is being used in the design. In addition, two I/O ports and a PLD based address decoder are incorporated.

**Figure 6.
A Typical
80C31 Design**



**Simple
80C31
Design**

The '373 latch, I/O ports, PLD address decoder, EPROM and SRAM can be easily replaced by a single PSD31X device, connected as shown in Figure 2. (the EPROM complement for the PSD31X series is 32K x 8 for the PSD311, 64K x 8 for the PSD312 and 128K x 8 for the PSD313). All are pin and function compatible.

A PSD3XX device is capable of replacing up to 6 devices in this fairly routine design.

If more complex memory maps are required, as in the system below, the system savings can be even greater. In addition, the hardware design can be easily reused in the future since I/O ports, chip selects, EPROM addresses and contents, etc., are all programmed into the PSD3XX. As a result, the printed circuit board layout will not need to be changed for significant system design changes.

**Adding
Capability...**

For designs that require more program and data space than the 80C31 can directly address (128K in Separate address space and 64K in Combined space), the PSD312 and PSD313 include a paging register to expand the usable memory space. For a more complete discussion, see WSI Application Note 015, "Using Memory Paging with the PSD3XX". This 4-bit page register enables the 80C31 to address up to 16 pages of 64K memory. To change from one page to the next requires only that the microcontroller write the page number of the memory page desired to the page register.

Although the page register is a good solution for systems requiring more address space than provided by the 80C31, there are times when more memory space is desired and using the page register may not be appropriate. Table 3 exhibits a system address map for a Combined memory space design that requires more than 64K of memory space. We do not want to use the page register in this case because the data memory must reside in the same page as the program memory, such that data can be accessed by the program without switching pages.

We can see from this memory map that the Separate mode memory model is not usable because some EPROM addresses are in the SRAM data space. Also, the total memory space required is 98K which exceeds the 64K memory space normally available in the Combined mode memory model.

The solution using the PSD3XX makes use of a feature included in all PSD3XX devices. Every PSD3XX can support up to 20 address inputs, directly enabling up to 1 meg of address space to be accommodated. Port C pins can be configured either as chip select outputs from the PAD or as address inputs that can be used in the memory map. Port C pins can be configured on an individual pin basis so chip select outputs can be provided at the same time as address inputs. By using the RD signal as an external input to the PAD via Port C, the RD signal can be used as an extra address bit to enable up to 128KB of address space to be accessed. See Figure 7.

When using this solution, the timing requirements are more stringent since the RD signal is valid later than the addresses are valid. Therefore, the 80C31/PSD3XX timing that must be satisfied is from RD valid to PSD3XX data out instead of the more usual 80C31 address valid to PSD3XX data out. See Figure 8.

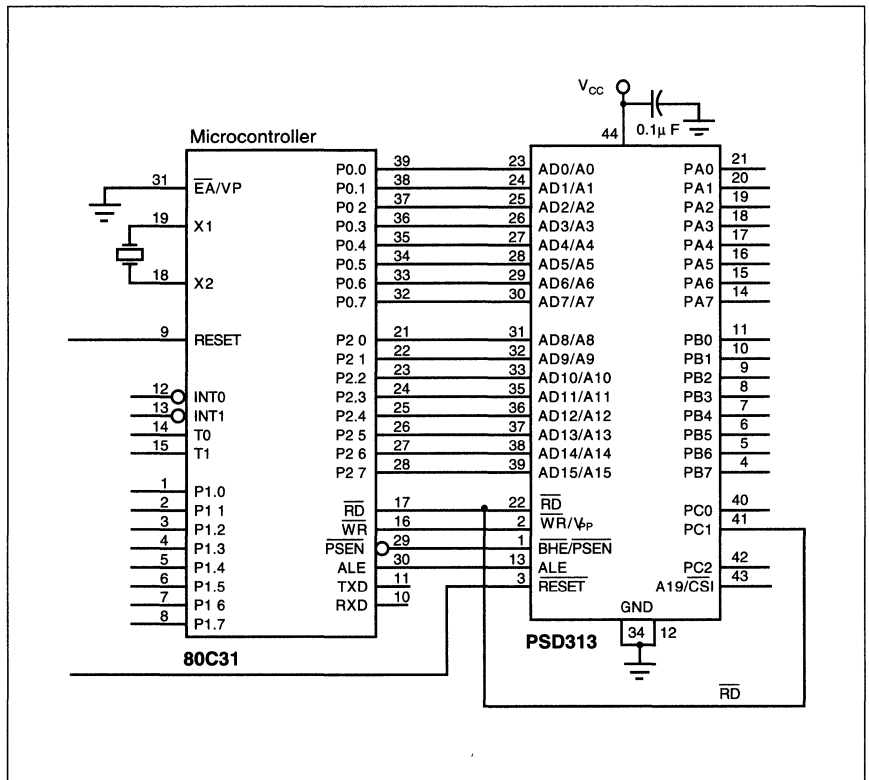
This tighter timing might require a higher speed PSD31X device. Since RD is now part of the input address, T2 as shown above must be used as the time required for memory access instead of T1. For example, in a 16 MHz 80C31 design, T1 is 257.5 ns (max.) and T2 is 222.5 ns (max.). In this case, a higher speed PSD3XX is not required since a PSD31X-20 would be fast enough to meet T1 and T2 timings.

**Table 3.
Combined
Memory Space
System Address
Map**

Address	Program Space	Data Space
0000 – 7fff	EPROM (32 KB)	SRAM (2 KB)
8000 – ffff	EPROM (32 KB)	EPROM Table (32 KB)

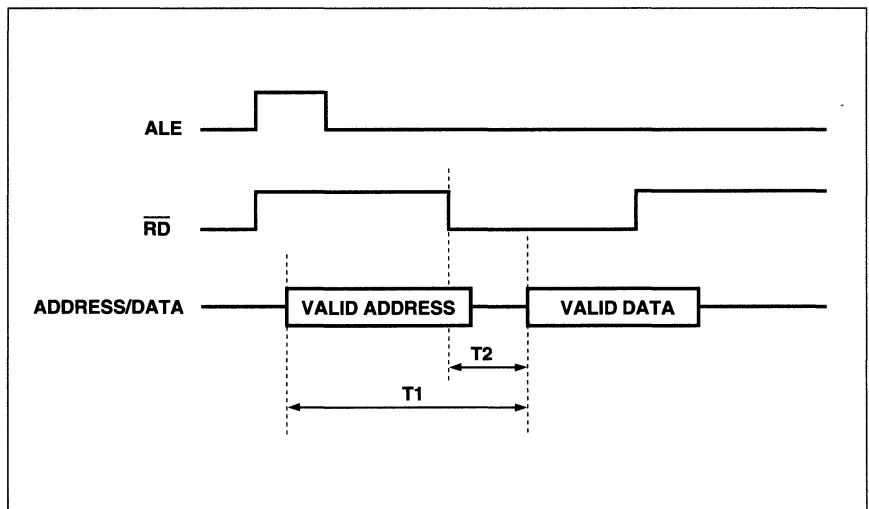


Figure 7.
 \overline{RD} Used
as an Address
Bit to Enable
128 KB
Addressing



1

Figure 8.
More Stringent
 \overline{RD} Valid
to Data Valid
Timing



**Adding
Capability...
(Cont.)**

Table 4 shows part of the MAPLE software solution for this problem which uses the Combined memory space configuration.

**Table 4.
Combined
Memory Space
MAPLE Software
Solution**

```

*****MAPLE 5.00*****
PSD PART USED: PSD313
*****PROJECT INFORMATION*****
Project Name :      = 8031 App. note
Your Name :       = Dan Kinsella
Date :           = May, 1992
Host Processor :  = 8031

*****ALIASES*****
/CS8/A16 = /RD
*****GLOBAL CONFIGURATION*****
Address/Data Mode :    MX
Data Bus Size :       8
Reset Polarity :      HI
Security :            OFF
ALE Polarity :        HI
Using Different READ strobes for Data and Program: Y
Separate Data and Program Address spaces: N
*****READ WRITE CONTROL*****
/RD and /WR
*****PORT C CONFIGURATION*****

Pin      CS/Ai      LOGIC/ADDR
PC0      A16        ADDR
PC1      CS9
PC2      CS10
A19      CSI

*****ADDRESS MAP*****
/RD = A16
   A  A  A  A  A  A  A  A  A  A  SEGM  SEGM  FILE  FILE  File  Page  Reg  Q.F
   19 18 17 16 15 14 13 12 11 STRT  STOP  STRT  STOP  Name  3210  ALE
ES0 N  N  N  0  1  0  N  N  N  18000 1BFFF 10000 13FFF TEST.HEX  N
ES1 N  N  N  0  1  1  N  N  N  1C000 1FFFF 14000 17FFF TEST.HEX  N
ES2 N  N  N  1  0  0  N  N  N  00000 03FFF 00000 03FFF TEST.HEX  N
ES3 N  N  N  1  0  1  N  N  N  04000 07FFF 04000 07FFF TEST.HEX  N
ES4 N  N  N  1  1  0  N  N  N  08000 0BFFF 08000 0BFFF TEST.HEX  N
ES5 N  N  N  1  1  1  N  N  N  0C000 0FFFF 0C000 0FFFF TEST.HEX  N
ES6 N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N
ES7 N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N
RS0 N  N  N  0  0  1  0  1  0  15000 157FF  N/A  N/A  N/A  N  N
CSP N  N  N  0  0  0  1  1  0  13000 137FF  N/A  N/A  N/A  N  N
*****END*****

*****ADDRESSES OF I/O PORTS*****
Pin Register of Port A :      3002  Page (Binary):
Direction Register of Port A : 3004
Data Register of Port A :     3006
Pin Register of Port B :      3003
Direction Register of Port B : 3005
Data Register of Port B :     3007
Page Register :              3018
*****

```



Reset Circuit Considerations

In a design where the 80C31 shares the same reset signal with a PSD3XX, a race condition exists if the reset circuit consists of RC components only. Due to slow fall time on the reset signal, the 80C31 could get out of reset mode and start fetching codes while the PSD3XX is still in reset.

The following three reset circuits are recommended for use with 80C31 and PSD3XX based designs:

1. Use a Reset Chip such as Dallas Semiconductor's DS1232, or Maxim's Max 699. Both have space saving, 8-pin mini-Dip packages. The reset output is connected directly to the reset pins on the 80C31 and PSD3XX.
2. Use two separate RC reset circuits: one which generates a high reset pulse to the 80C31 and the other one generates a low reset pulse to the PSD3XX. The RC constant of the PSD3XX reset circuit should be less than that of the 80C31 such that the PSD3XX reset signal has a shorter pulse. In this case, the PSD3XX is configured to have an active low reset input.
3. Use a buffered reset signal: The output of an RC reset circuit is buffered by a gate (such as 74HC14) before it is connected to reset pins on the 80C31 and PSD3XX.

1

Summary

Using the PSD3XX family of Programmable Microcontroller Peripherals in 80C51 type microcontroller designs can significantly reduce system design complexity and enhance the end product at the same time. Reusing hardware sections of existing designs in new designs becomes an easy process. Changing memory mapping and even memory sizes is easily accomplished by making simple adjustments in the PSD3XX development software, i.e. by

simply selecting where and how many of the blocks are required. The flexibility of the PSD3XX family is demonstrated by the equal ease of design if a Combined or Separate address space is needed. In addition, the 4-bit memory paging register in the PSD3X2 and PSD3X3 devices enables the microcontroller to address additional memory by a factor of 16.



Programmable Peripheral Application Note 023

PSD3XX Family Programmable Microcontroller Peripheral Design Tutorial

By Mark Elliott

Introduction

The PSD3XX family devices contain several commonly used microcontroller peripheral functions combined into one package. These include EPROM, SRAM, Chip Selects, and logic functions. Some of the advantages of the PSD3XX family are: board space is reduced, power consumption is reduced, cost is competitive – usually less, and board complexity is reduced. Design risk is also reduced because fewer traces are required on the PWB and the PSD3XX devices are more flexible than a discrete component design. Mistakes or design changes pose less of a potential problem. Additionally, the PSD3XX device includes a security option which, when implemented, protects the internal configuration data from duplication.

For a particular application, the designer should learn the PSD3XX family architecture, understand the configuration software, and understand the programming process. While none of these tasks are complicated, full knowledge of them is not required to understand the PSD3XX family.

This application note introduces the PSD3XX family design process by example. While going through the examples, you will learn about the entire design process including hardware, software, and programming. Those who do not have a strong background in the microcontroller field may also find themselves able to use the PSD3XX. This application note uses an 80C31 system as a model. Even if you intend to use a different microcontroller, you will find it useful to read on.

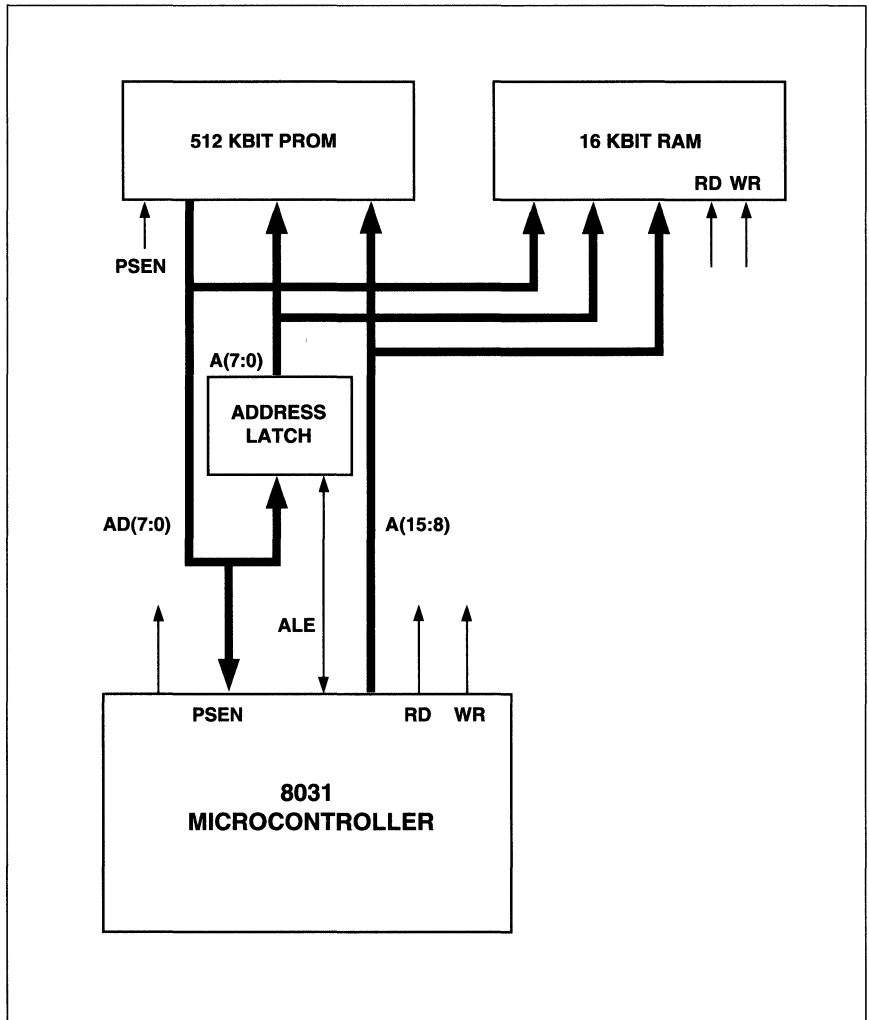
This application note demonstrates two designs using multiple packages which will be replaced by a design using the PSD312. The first example is a standard 80C31 board, one that does not make use of all the PSD312's potential. This will form a basis for understanding the product. In a second example, new functions are added to the standard design. By the end of the application note, you should have enough basic knowledge to understand the PSD312 device's use in your design.

PART I – Using the PSD312 with a Standard 8031 System.

Figure 1.1 illustrates a standard 80C31 microcontroller board design. The board contains a microcontroller, a 512 Kbit EPROM for program storage, a 16 Kbit static SRAM, and an address latch. In this example, all of the circuits on this board, excluding the microcontroller, will be replaced. Keep in mind that the PSD312 is not being used to its full advantage here. In the second example, Part 2 of this application note, you will see that the PSD312 is able to provide additional functions, replacing additional discrete packages.

The PSD312 was chosen for this task because it has the same SRAM and EPROM space as that used on the original design. A similar device, the PSD311, would be more suitable for replacing a smaller EPROM space of 256K bits or less. If a larger EPROM space is required, a PSD313 with its internal 1M bit UV EPROM could be used.

Figure 1.1:
8031
Microcontroller
Standard System
Block Diagram



NOTE: Each Block represents one IC package.

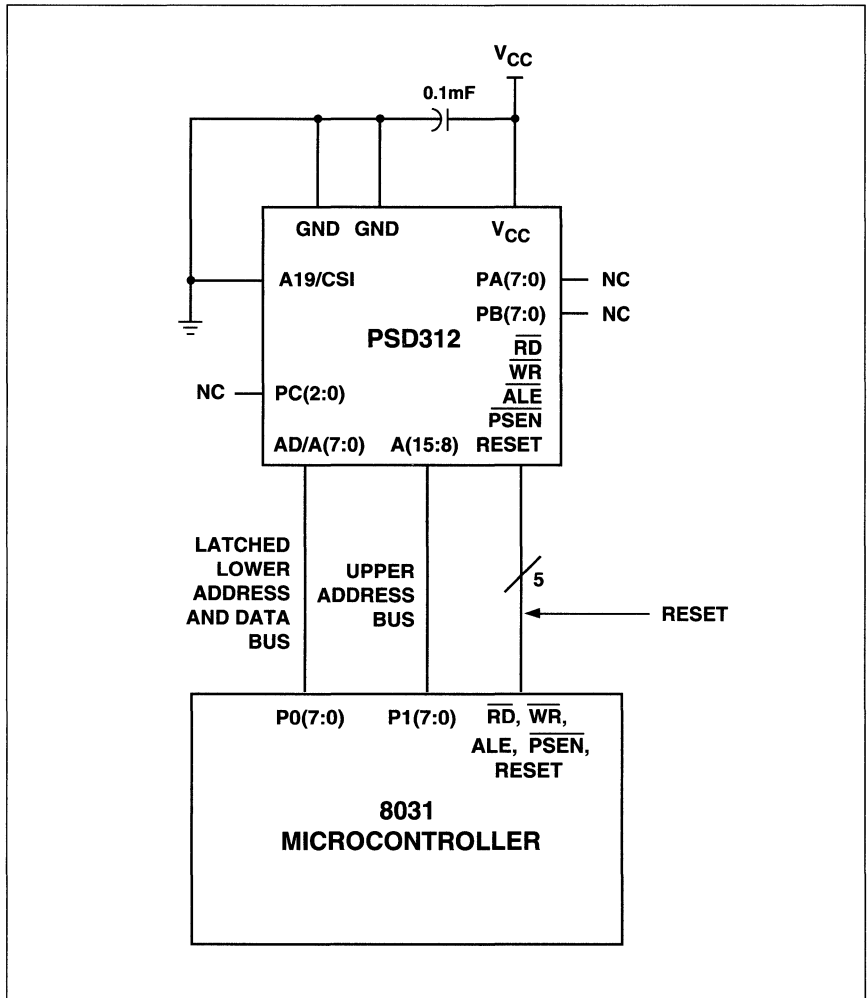
Physical Connections

The physical connections for the new board design using the PSD312 are illustrated in Figure 1.2. The multiplexed address/data pins, 0 through 7, from the 80C31 port 0 connect to the PSD312 pins AD/A(7:0). The upper address bits, 8 through 15 from port 1 connect to the pins A(15:8) on the PSD312. The 80C31 write line is connected to "WR.Vpp or R/W", the read line to RD/E, ALE to "ALE or AS". The connections for

PSEN and RESET are straightforward. The A19/CSI pin is an unused input in this application so it is tied low. Last of all, the power, grounds and the decoupling capacitor are connected. All other PSD312 pins will remain unconnected. These pins become useful for a more functional design such as in the second example.

1

**Figure 1.2:
Standard PSD312
Physical Connections**



NOTE: Additional Microcontroller connections are not shown.

Configuration Data Entry

Before the PSD312 can be programmed, the MAPLE software is used to define the PSD312 functionality. The MAPLE software is menu driven making it very easy to use. After the software is loaded, entering the command MAPLE accesses the software used to program the PSD3XX device. The main MAPLE menu will appear on the screen with its seven one-line options. Highlighting the PARTNAME option (or pressing F8), typing "PSD312" and pressing ENTER selects the part intended for programming. Next, the main menu will call up the correct subprogram used to configure the part selected. The subprogram looks very much like the main menu except that a second option box will appear on the right of the screen. See Figure 1.3 for an illustration of this menu as it appears on the computer display screen. In all illustrations, only the menus will be shown to enhance clarity. Other information may appear on the screen. For example, help information often appears to assist the designer. After an option is highlighted using the cursor controls, pressing ENTER will execute that option. The cursor controls enable the user to maneuver between option boxes and among the options within a box.

To configure the PSD312 you must focus on the option box to the right of the screen. The first option in this box is PROJECT INFORMATION which enables the user to store information about the software file.

The next option is ALIASES which provides the ability to name a pin with an alias name to help prevent mistakes. The alias name will show up in other menus that use the pin. This option will not be used in this example.

The next option is CONFIGURATION. After the CONFIGURATION option is selected, the CONFIGURATION menu, like that shown in Figure 1.4, will appear. Figure 1.4 displays the correct information for this example. When a line in the CONFIGURATION menu is highlighted, an explanation of the question is provided in a dialogue box at the bottom of the screen. The last question, "Separate Data and Address Spaces?" appears when Y is answered to "Using different READ strobes...". This question will not appear when the menu is first entered. Each question has two possible answers. Pressing the space bar will toggle between the two answers; no typing is required. Each answer to an option is explained here. See Figure 1.4 for reference.

This CONFIGURATION menu contains all of the necessary information to configure the PSD312 for use with the 80C31. Additional information is required in other menus to program the PSD312 logical functions. Pressing F1 returns to the main menu. All of the data entered in the CONFIGURATION menu (and all other menus) is automatically saved.

MX	The 80C31 uses a multiplexed address. We have made the correct connections in the hardware design so that the PSD312 will automatically know which pins to multiplex and how to multiplex them.
8	The 80C31 has an 8 bit data bus width. This is the only width supported by the PSD312.
A19	This option is arbitrary at this time because we are not using A19 as an input or CSI (the power down option).
HIGH	The reset polarity is high for the 80C31.
WR and RD	The 80C31 uses separate write and read strobes for SRAM.
HIGH	The ALE polarity is high for the 80C31.
Y	The separate read strobes are RD for SRAM and PSEN for EPROM. The Y answer enables the PSEN feature.
Y	The data and address space are separated by the read strobes. Data (SRAM) and program (EPROM) share the same address space. The different read strobes, RD for SRAM and PSEN for EPROM, enable them to share this space.

**Figure 1.3:
MAPLE
Main Menu**

F1	EXIT	PROJECT INFORMATION
F2	DOS	ALIASES
F3	MAPPRO	CONFIGURATION
F4	PARTLIST	PORT C AND A19/CSI
F5	LOAD	PORT A
F6	SAVE	PORT B
F7	COMPILE	ADDRESS MAP

1

**Figure 1.4:
Configuration
Menu**

<i>Configuration Bit</i>	<i>Value</i>
Address/Data Mode (Multiplexed: MX, Non-Multiplexed: NM)	MX
Data Bus Width (8/16 Bits)	8
Reset Polarity (Active Low: LO, Active High: HI)	HI
Security (ON/OFF)	OFF
To Select Read, Write Logic, Press SPACEBAR.	
ALE Polarity (Active Low: LO, Active High: HI)	HI
Using Different READ Strokes for Data and Program? (Y/N)	Y
Separate Data and Address Spaces? (Y/N)	Y

Configuration Data Entry (Cont.)

The next option in the option box is PORT C and A19/CSI. Although these pins are not used, they must be correctly programmed. Figure 1.5 shows the menu containing the correct information for this example. Port C consists of three pins PC(2:0). Each of these pins can be individually configured as a chip select (output) or an address (input). If ADDRESS is selected, there is the choice of using a LOGIC or ADDRESS input type for each pin. The space bar is used to toggle among the selections. (PC0 = CS8 or A16, PC1 = CS9 or A17, PS2 = CS10 or A18.) An address can exist out of sequence as in this example where A19 exists but A16, A17, and A18 do not.

The PORT C pins **must be tied low** if they are programmed as inputs and are unused. However, outputs are not required to have terminations and can remain untied. The easiest solution is to program all Port C pins as chip select outputs to avoid unnecessary board traces. The user could proceed to define the chip select equations for the pins defined as chip selects but it is not important since they will not be used. Pressing F1 will return the program to the main menu.

Port A is the next menu option on the right of the main menu screen. The PORT A ADDRESS/IO menu selections are shown in Figure 1.6. PORT A will not be used in this design but it should be programmed correctly. The default selections are the same as shown here. The selections are shown to help the user understand PORT A functions. After selecting the PORT A option you are given a choice between ADDRESS/IO and TRACK MODE. The ADDRESS/IO option, if chosen, enables each PORT A pin to output-buffer the address bits from AD/A(7:0) or transmit bits through the internal I/O ports. Each pin is individually programmed as either an ADDRESS or an I/O in the Ai/IO column of the menu. Each pin should be chosen as an output address. Pressing F1 twice returns the program to the main menu.

Port B is the next option in the menu. PORT B pins can be configured as chip select outputs or I/O ports just like PORT C. Since PORT B will not be used in this example, all of the pins should be configured as outputs for the same reason as given for the PORT C pins.

**Figure 1.5:
Port C Menu –
Standard**

PIN	CS/Ai	ADDR/LOGIC
PC0	CS8	
PC1	CS9	
PC2	CS10	
A19	A19	LOGIC

**Figure 1.6:
Port A Menu –
Standard**

PIN	Ai/IO	CMOS/OD
PA0	A0	CMOS
PA1	A1	CMOS
PA2	A2	CMOS
PA3	A3	CMOS
PA4	A4	CMOS
PA5	A5	CMOS
PA6	A6	CMOS
PA7	A7	CMOS

**Configuration
Data Entry
(Cont.)**

The last option is ADDRESS MAP. The address map lets the user select address ranges for EPROM, SRAM, and I/O in the same way that you would using separate packages. The difference is that, for separate packages, you would use chip selects to enable a chip when in its address range. For the PSD312, you just enter the conditions that will select each module. Figure 1.7 shows the correct address map entries for this design. Looking at addresses 16, 17, and 18, there is an N listed down the column. This means that the addresses can not be used as address inputs. They do not exist as addresses and can not be used as select bits. This is so because, in the configuration of PORT C, we chose all of these pins to be chip select outputs. If we had configured the PORT C pins to be logic inputs, then we would have had to enter an X for "don't care" in those same columns. The SEGMENT columns are usually filled in automatically by the program. They are left blank in this case because the N listed down the address columns make the segments undefinable.

Instead of having one contiguous EPROM space, the PSD3XX family EPROM is broken up into 8 Kbyte blocks. In this application, the selects for each block are ES0 through ES3. Notice that no two blocks of EPROM are selected under the same address conditions. After all, you can't look at two different EPROM addresses at once. The other selects, ES4 through ES8, will not be used in this example. An N is listed down the columns

for A11 and A12 because the blocks are bigger than the use of these address bits permit. The N stands for "not used" because the bits are not used for selection. That is, we "don't care" about these bit values. The EPROM file we will use is called DEMO.HEX and it exists in the MAP directory so that the MAPPRO software can find it when programming the PSD312. The size of the blocks for DEMO.HEX match the side of the blocks for the EPROM. RS0 is the select for SRAM. Notice that SRAM (RS0) and EPROM (ES0) can be selected under some of the same address conditions. This is not a problem because SRAM and EPROM can share address space since their data is selected by different read strobes (PSEN and RD). See the CONFIGURATION menu, Figure 1.4). CSP is the chip select for the PSD312 I/Os. I/O ports are selected by selecting each appropriate address which is an offset from the address of CSP. We are not using the I/O's so we can disable them by selecting a true condition for A19. A19 is tied low in the hardware so this condition will never occur. This hardware solution is not required in most cases. Usually, a CSP I/O base address can be set aside so that I/O data will not interfere with EPROM or SRAM data.

The ADDRESS MAP menu extends towards the right of the screen. Pressing the right cursor pans the menu to the right. This portion of the menu does not require input for this example. Leaving the inputs blank means "don't care". Pressing F1 returns the main menu.

Configuration Data Entry (Cont.)

At this point, all of the necessary menu items have been completed. On the left option block in the main menu is a SAVE option. You must type a name to store the file and press ENTER. The name given to this file will be DEMO. The extension is automatically appended. Saving is done so that next time the DEMO file is loaded, the menus will display the information entered. This is especially useful when modifying earlier designs. The file needs to be compiled to run on the MAPPRO software.

Selecting the compile option does this. The same name DEMO is typed and the correct extensions will be appended to the files generated. The user can look at a report file which is generated during the compilation. It can be used to verify the programming parameters. The COMPILER option takes a few minutes to run, depending on the speed of the PC used. The compiled configuration file can now be used in MAPPRO to program the PSD312.



Figure 1.7: Standard Address Map Menu

	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMT START	SEGMT STOP	FILE START	FILE STOP	FILE NAME
ES0	0	N	N	N	0	0	0	N	N			0	1FFF	DEMO.HEX
ES1	0	N	N	N	0	0	1	N	N			2000	3FFF	DEMO.HEX
ES2	0	N	N	N	0	1	0	N	N			4000	5FFF	DEMO.HEX
ES3	0	N	N	N	0	1	1	N	N			6000	7FFF	DEMO.HEX
ES4		N	N	N				N	N					
ES5		N	N	N				N	N					
ES6		N	N	N				N	N					
ES7		N	N	N				N	N					
RS0	0	N	N	N	0	0	0	0	0			N/A	N/A	
CSP	1	N	N	N	0	0	0	0	0			N/A	N/A	



Programming The PSD3XX

The MAPPRO software is used with the WSI MagicPro® PROM programmer to program the PSD312. MAPPRO can be started from the main menu or by entering the command MAP. When started, the MAPPRO menu appears as a list of options. Each option is selected by typing the first letter of the option.

The "Name of the Device" option selects the device to be programmed. Typing in PSD312 selects that device. The DEMO file is loaded by the "Load RAM from disk" option. After the PSD312 is plugged into the MagicPro device, program the PSD312 by selecting "Program". The program then asks for the starting addresses of SRAM and EPROM for which the default address is entered. Programming will take a few minutes.

The PSD3XX Family Device can be made secure by selecting the "Set Security" option. Make certain that this option is not selected until after the device configuration is programmed. If the security bit is programmed before the configuration, the PSD312 will fail the blank test and will require UV erasing.

The MAPPRO software can check a design by using the "verify" option. This option compares a PSD3XX series device, which is installed in the MagicPro® PROM programmer, to configuration and EPROM data which was previously loaded in the MAPPRO software. The "previously loaded" information may have been loaded during either the "Load RAM from disk" or "Upload data from device" option.

The "Upload data from device" option reads the PSD312 information and installs it into the MAPPRO RAM. With this option, PSD3XX family devices can easily be compared to one another or to the MAPPRO RAM. However, if the security bit has been set, selecting this option will not load the data.

The "Display RAM data" option can be used to display the PSD312 data which is contained in the MAPPRO RAM. This can be especially useful when you need to analyze EPROM data.



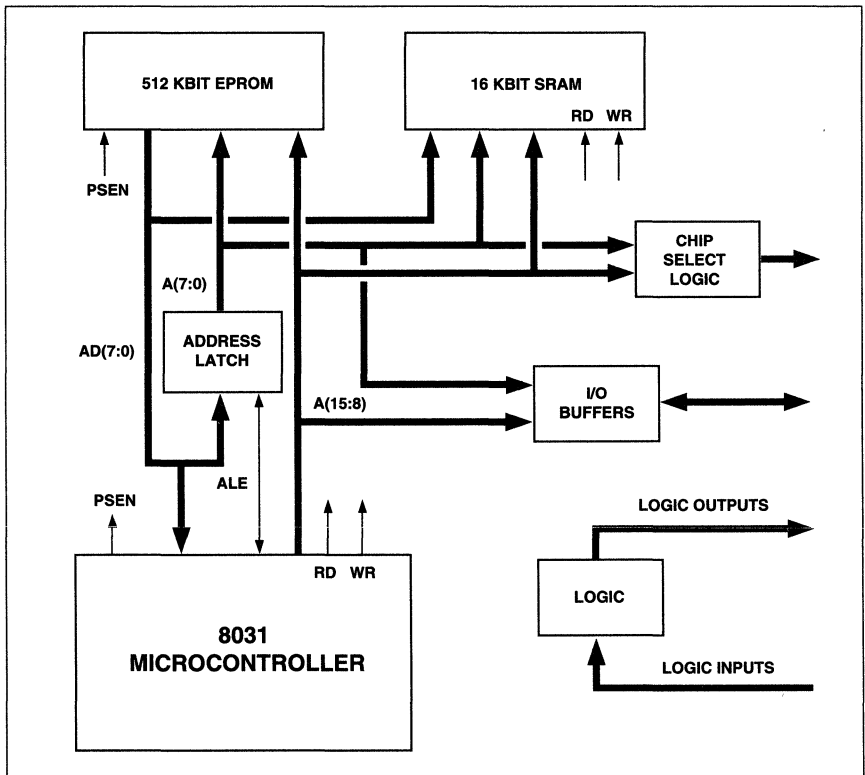
**PART II.
Advanced
PSD3XX Family
Design**

By now you should have a basic understanding of the PSD3XX device so it is time to introduce some additional features. This example will solve a slightly more complicated design problem. By going through this example you should understand how to use the PSD3XX device to realize functions for your own unique designs.

Assuming the design in Part 1 has been created and saved under the name DEMO, you can load that program in the MAPLE menu to begin this new design. This will keep you from having to enter redundant information. For example, the CONFIGURATION menu will not require any changes in this design. In this second example, restating the method of navigating through the menus will be avoided for purposes of brevity.

The diagram in Figure 2.1 illustrates the new microcontroller board design to be replaced by a design using the PSD312. In addition to functions previously replaced in the standard board design, this board includes chip select logic, I/O buffers, and a logic chip. The logic chip does not perform microcontroller functions but is included to show the flexibility of the PSD312. An additional change is there is now an SRAM chip select which is an input to the board. Its logic select function is defined elsewhere so it does not need to be recreated in the PSD312's chip select logic. This scenario would occur when some other device has a separate SRAM of its own. This other device will decide whether the microcontroller writes to the PSD312 SRAM or its own SRAM. The configuration of this design is mostly the same as in Part 1. We are now using PORT A as an I/O buffer, all PORT C pins are now used as logic inputs, and we must specify the chip selects in PORT B to conform to our logic and chip selects.

**Figure 2.1.
Advanced 8031
Microcontroller
System
Block Diagram**



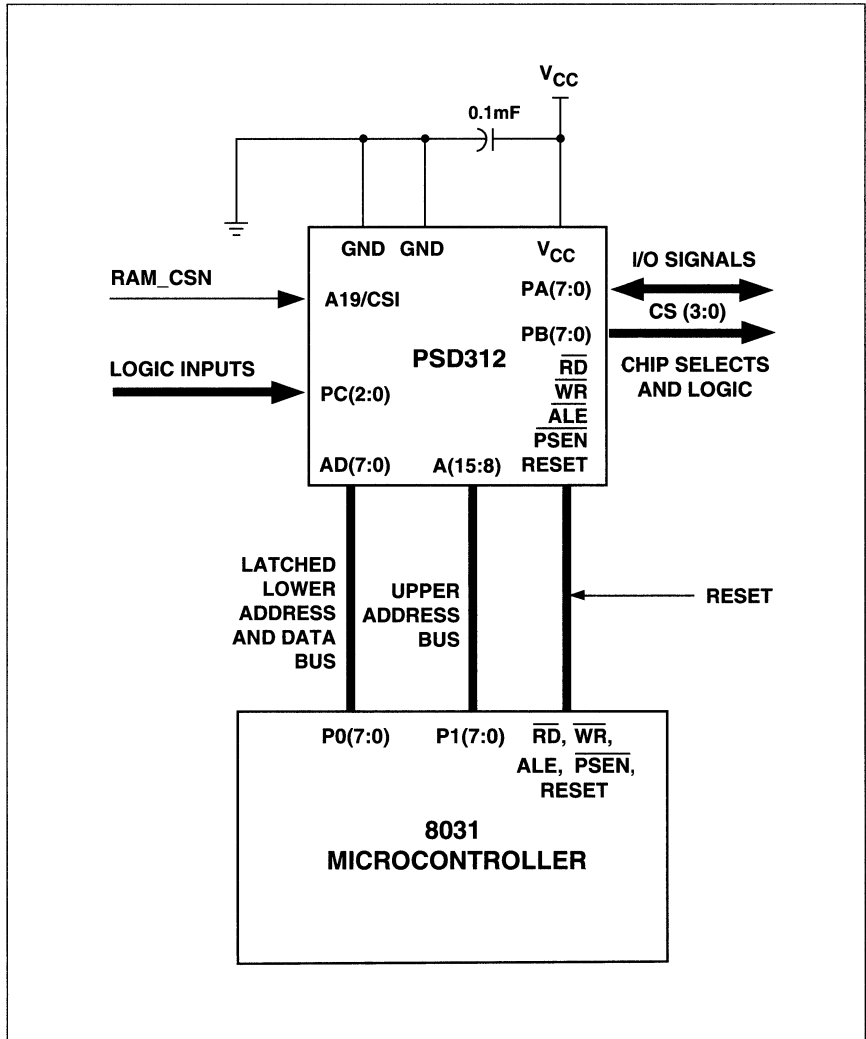
NOTE: Each Block represents one IC package.

**Advanced
PSD3XX Family
Design
(Cont.)**

Figure 2.2 illustrates the physical connections to the PSD312. The SRAM chip select is input to A19 (A19/CS1), and the logic inputs are input through A(18:16). These are arbitrary assignments among the address inputs. PORT C could not be used for chip select outputs because we

needed to make room for the inputs. See Figure 2.3 for PORT C menu selections. Notice that the address inputs can be used for logic. Although the name of the inputs are ADDRESS, they are really either logic or address inputs.

**Figure 2.2:
Advanced Design
PSD312/8031
Physical
Connections**



NOTE: Additional Microcontroller connections are not shown.

**Advanced
PSD3XX Family
Design
(Cont.)**

The I/O buffers connect to the PSD312 through PORT A. PA(3:0) are inputs, PA(7:4) are outputs. See Figure 2.4 for PORT A configuration.

Port B is used for the chip select and logic outputs. See Figure 2.5, the PORT B menu. These pins must be defined as chip select outputs. The CMOS/OD for CMOS or open drain output is the next option. From here we go to the chip selects for the PORT B pins. Table 2.1 describes the function of the logic chip. This type of logic might be used in a state machine. In the menu selections

for CS0 we look at the occurrences of a logic low from the table and enter that information into the chip select menu as shown in Figure 2.6. Notice that all other columns are left blank since they won't be part of the chip select equation. That is, we don't care about their levels. For example, we could use ALE or the page selects, P0 through P3, as logic inputs. Figures 2.7, 2.8, and 2.9 show the other chip select menus. These chip selects enable components on other boards when their address ranges are encountered.

**Figure 2.3:
Advanced
Port C
Menu**

PIN	CS/Ai	ADDR/LOGIC
PC0	A16	LOGIC
PC1	A17	LOGIC
PC2	A18	LOGIC
A19	A19	LOGIC

**Figure 2.4:
Advanced
Port A
Menu**

PIN	Ai/IO	CMOS/OD
PA0	IO	CMOS
PA1	IO	CMOS
PA2	IO	CMOS
PA3	IO	CMOS
PA4	IO	CMOS
PA5	IO	CMOS
PA6	IO	CMOS
PA7	IO	CMOS

**Figure 2.5:
Advanced
Port B
Menu**

PIN	Ai/IO	CMOS/OD
PA0	CS0	CMOS
PA1	CS1	CMOS
PA2	CS2	CMOS
PA3	CS3	CMOS
PA4	CS4	CMOS
PA5	CS5	CMOS
PA6	CS6	CMOS
PA7	CS7	CMOS

**Table 2.1:
Logic Truth
Table**

INPUTS			OUTPUT CONDITION
A	B	C	(CS0)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

**Table 2.2:
Chip Select
Address Map**

CHIP SELECT	ADDRESS RANGE	ADDRESS BITS				
		15	14	13	12	11
CS1	D000 – DFFF	1	1	0	1	X
CS2	C000 – C7FF	1	1	0	0	0
CS3	C800 – CFFF	1	1	0	0	1

**Figure 2.6:
Chip Select
Definition CS0**

A19	A18	A17	A16	A15	A14	A13	A12	A11	RD	WR	ALE	P3	P2	P1	P0
	0	0	0												
	0	1	1												
	1	1	0												
	X	X	X												

1

**Figure 2.7:
Chip Select
Definition CS1**

A19	A18	A17	A16	A15	A14	A13	A12	A11	RD	WR	ALE	P3	P2	P1	P0
				1	1	0	1								
				X	X	X	X								
				X	X	X	X								
				X	X	X	X								

**Figure 2.8:
Chip Select
Definition CS2**

A19	A18	A17	A16	A15	A14	A13	A12	A11	RD	WR	ALE	P3	P2	P1	P0
				1	1	0	0	0							
				X	X	X	X	X							
				X	X	X	X	X							
				X	X	X	X	X							

**Table 2.9:
Chip Select
Definition CS3**

A19	A18	A17	A16	A15	A14	A13	A12	A11	RD	WR	ALE	P3	P2	P1	P0
				1	1	0	0	1							
				X	X	X	X	X							
				X	X	X	X	X							
				X	X	X	X	X							

**Advanced
PSD3XX Family
Design
(Cont.)**

The address map is shown in Figure 2.10. Notice that the SRAM chip select, A19, must be enabled for the microcontroller to access the SRAM. This will enable another device to select which SRAM is enabled for a read or write. Address A19 is listed as a "Don't care" or "X" for ES0 through ES3 because the SRAM chip select, A19, may be enabled during an EPROM access. The

read strobes, PSEN and RD, are used to separate the EPROM and SRAM. The CSP is the enable for the I/O functions. It is selected for address 80 Hex. Since the EPROM address locations have been carefully mapped, the I/O address selections will not coincide with EPROM addresses.

**Figure 2.10:
Advanced
Address Map
Menu**

	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMT START	SEGMT STOP	FILE START	FILE STOP	FILE NAME
ES0	X	X	X	X	0	0	0	N	N			0	1FFF	DEMO.HEX
ES1	X	X	X	X	0	0	1	N	N			2000	3FFF	DEMO.HEX
ES2	X	X	X	X	0	1	0	N	N			4000	5FFF	DEMO.HEX
ES3	X	X	X	X	0	1	1	N	N			6000	7FFF	DEMO.HEX
ES4								N	N					
ES5								N	N					
ES6								N	N					
ES7								N	N					
RS0	0	X	X	X	0	0	0	0	0			N/A	N/A	
CSP	1	0	0	0	0	0	0	0	0			N/A	N/A	



Conclusion

A PSD3XX family device can implement many common microcontroller functions and it is flexible enough to be used on designs requiring special functions. Its use will reduce the component count, layout complexity, size, component cost, PCB cost, and power consumption of a design. Reliability is increased due to the reduced chip count. The risk of board redesign is minimal given the ease of design and the PSD3XX device's flexibility. The user friendly software makes it easy to use in any design.



Programmable Peripheral Application Note 024 Using the PSD311 with a High-Speed ADSP-2105 DSP

By Lane Hauck, Proxima Corp.

Introduction

Digital Signal Processor (DSP) chips are enjoying a surge in popularity with system designers, due to their high performance and dropping prices. Fueled by a large PC peripheral market, primarily in disk drive controller and "multimedia" signal processing applications, DSP offerings now include high-performance processors that

execute instructions in 100 ns or less for under \$10. To achieve a total solution the designer needs to add periphery circuitry. The WSI PSD3XX family of programmable peripherals implement the periphery functions by effectively integrating programmable logic, I/O ports and memory on a single chip.

The Processor

The Analog Devices ADSP-2101 family is a good example of the inexpensive DSP power available today. The lowest-cost member of the family, the ADSP-2105, has the following features:

- 16-bit multiplier/accumulator
- Hardware (zero time) looping
- 1K 24-bit internal program memory
- 512 16-bit internal data memory
- Two hardware address generator units
- Barrel shifter
- Synchronous serial port
- Timer

In one 100 ns cycle, the ADSP-2105 can fetch two operands, update the address units that pointed to the operands, multiply the two 16-bit operands and accumulate (add) the result to a 40-bit total. Program looping is done in hardware, so one of these fancy instructions can be executed every 100 ns.

The Periphery

The WSI PSD3XX family has found wide application in microcomputer systems. Because the PSD3XX chips offer all of the elements and "periphery" required for many applications in one package, they make possible very economical two-chip computer systems. Thumbing through the WSI PSD Data Book, it soon becomes apparent that the PSD3XX chips are great for 8051, 68HC11 and other microcontroller designs. But what about a chip for a DSP like the ADSP-2105?

Although a PSD100 chip is available expressly for DSP support, it's not the lowest cost choice because of its very high speed (access time of 45-55 ns). However, a feature of the ADSP-2101 family makes it possible to use the lowest cost PSD3XX chip available, the PSD311, to make a true 2-chip DSP system! In fact, the 100 ns ADSP-2105, which requires program memory access time of around 50 ns, can use the PSD311-12 (or even a slower version) for all of its system support, while still executing programs at a sustained 100 ns cycle time.

**ADSP
Memory
Organization**

**Program
Memory**

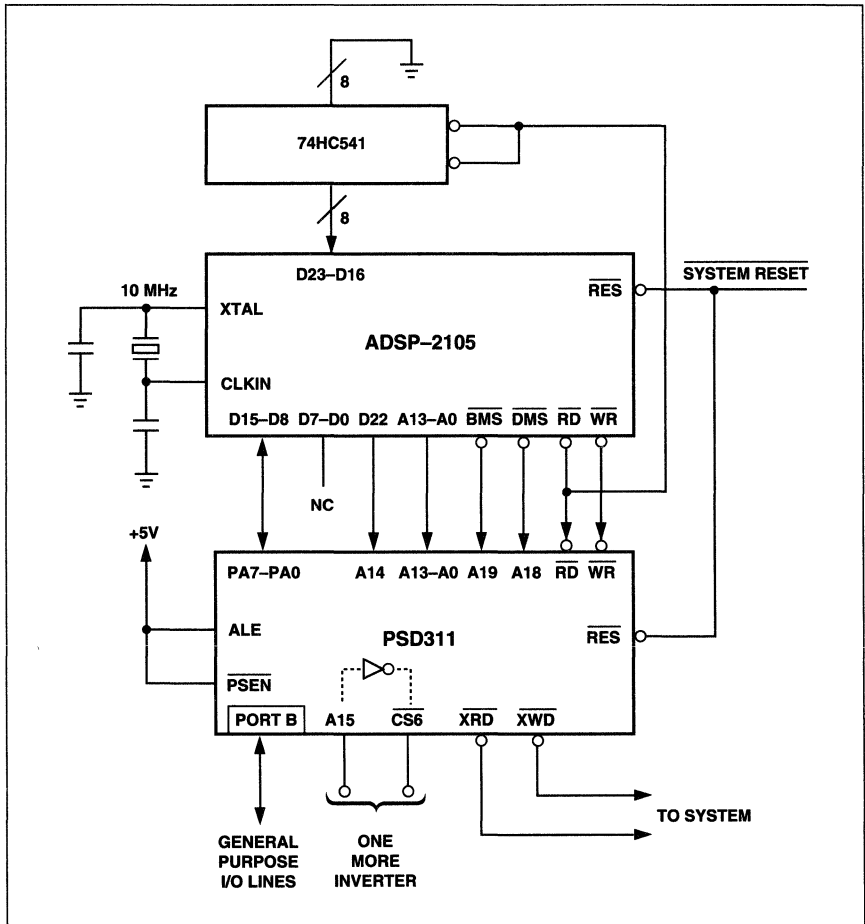
Program memory for the ADSP-2105 consists of 1024 24-bit words of RAM inside the DSP chip. A special external memory space, called "boot memory," is supported by the ADSP-2105 to enable connection of a byte-wide EPROM to the ADSP-2105 for loading the program memory at power-up, or subsequently under program control. A special active-low strobe signal, BMS (Boot Memory Select), simplifies the boot memory interface. All you need to interface a boot EPROM is to connect the EPROM address and data lines to the ADSP-2105 (see Figure 1), and the BMS signal to the PSD311 chip enable.

The two-chip design uses a 2K byte section of the PSD311's EPROM as boot memory. The connection to the PSD311 is almost as straightforward as connection to

a standard EPROM. Because of the way the busses are laid out inside the ADSP-2105, the eight PSD311 data lines are connected not to D7–D0, as you would expect, but to D15–D8 (see Figure 1). Also notice that the most significant "D22" line – there's no "A14" address line. The BMS signal acts as an EPROM chip select and is connected to the PSD311 "A19" input. A19 is programmed as a chip enable signal, as described later.

The ADSP-2105 generates active low read and write strobes, which are connected to the corresponding PSD311 RD and WR inputs. These strobes serve to enable transfers to and from the PSD311 EPROM and RAM.

**Figure 1.
Schematic
Diagram**



Data Memory

The ADSP-2105 has 512 16-bit words of internal data memory, located at \$3800. This is augmented by the 1024 bytes of RAM in the PSD311. Fortunately, the data bus is connected to the same ADSP-2105 data lines -- D15:D8 -- as the boot EPROM, so the address and data connections that were made for the boot EPROM are also valid for the RAM. All that is needed for RAM system integration is to program the PSD311 to place its RAM in the desired slot on the ADSP-2105 memory space.

Although using the 8-bit wide PSD311 RAM in a 16-bit system may be troublesome, remember the ADSP-2105 already has 512 16-bit words of high-speed internal RAM.

This design made good use of the "extra" RAM in the PSD311, and even incorporated an additional chip to gain some speed in using this RAM. Notice that the ADSP-2105 data lines D23–D15, which constitute the upper byte of the 16-bit data bus, are driven by an octal, tri-state buffer whose inputs are tied low. If this were not done, any 16-bit read of the 8-bit PSD311 RAM would contain garbage in the upper 8 bits (since these inputs are unconnected). If you can spare a couple of cycles to mask off the upper bits, or if your application doesn't look at the upper 8 bits, the buffer can be eliminated.

1

Logic Functions

The Programmable Address Decoder (PAD) is used to generate miscellaneous control signals, plus some I/O port bits. Two active-low strobes, \overline{XRD} (external read) and \overline{XWR} (external write), go to other I/O peripherals in the system. As with all

designs, this was one inverter short, so the A15 line was configured as an input, the $\overline{CS6}$ line as an output, and with the logic equation $\overline{CS6} = A15$ a chip was saved. Very handy, that PSD311 PAD!

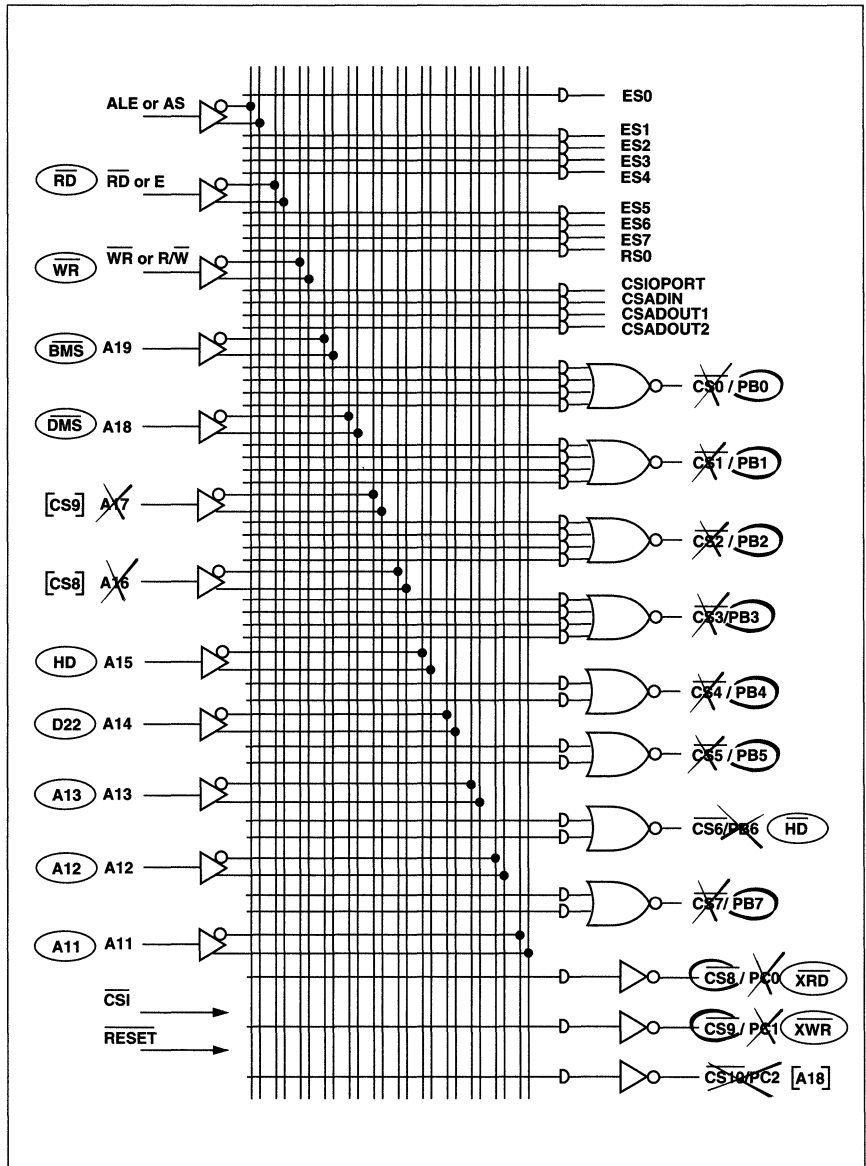
Configuring the PSD311

The WSI "PSD-Gold" development system was used to specify the PSD311 design file, and the MagicPro® Programmer (included in the PSD-Gold system) to program PSD311 parts. Appendix B shows the ".SV1" file that was created from the design. The ".SV1" file is a convenient summary of the PSD311 design, which includes aliases (named signals), global configuration information (how the part is set up), Port B configuration (assignment of port-B pins as I/O bits or chip selects outputs), Port C configuration, and logic equations for the two PADs.

Before using the PSD-Gold development system, it is very helpful to make a configuration worksheet as shown in Appendix A. This is a version of the PSD311 PAD description chart that appears in the WSI *Programmable Peripherals Design and Applications Handbook*. Some of the PSD311 pins can serve as either inputs or outputs. The worksheet helps to keep the pin assignments clear.

For example, $\overline{CS8}$ and A16 share the same pin, so if you use the pin as output $\overline{CS8}$, the input A16 pin is unavailable (shown crossed out in the worksheet). Likewise, the $\overline{CS9}$ -A17 pin is used as $\overline{CS9}$, so A17 is unavailable, and $\overline{CS10}$ -A18 is used as an input (A18), so the $\overline{CS10}$ -PC2 pin is crossed out.

Appendix A



Memory Space

The ADSP-2105 outputs a 14-bit address. The PSD311 has a 32KByte EPROM, organized as eight 4K byte "blocks," each with its own internal chip-enable signal that is programmed in one of the PADs. EPROM chip select signals are therefore developed from address lines A12–A14, with A0–A11 (accounting for each 4K block) going to each of the eight EPROM blocks. It was puzzling at first to see the signal A11 appear on the PSD311 PAD description since it does not participate in any of the chip select equations. Only after firing up the PSD-Gold software was it made clear that this input is not allowed in the EPROM chip select equations.

A12–A14 are connected to ADSP-2105 A12, A13 and D22 (D22 serves as "A14" as previously mentioned). The remaining PSD311 address inputs A15, A18, and A19 are available as general purpose inputs: A18 was used as $\overline{\text{DMS}}$, the ADSP-2105 Data Memory Select signal, and A19 as $\overline{\text{BMS}}$, the ADSP-2105 Boot Memory Select signal. As mentioned previously, the ADSP RD and WR signals are connected directly to the corresponding PSD311 signals.

The inverter consists of the unused A15 input and the $\overline{\text{CS6}}$ output. These signals are labeled "HD" and "HD" in the Appendix A worksheet.

Appendix B gives a summary of the questions asked by the configuration software, and the answers given for the DSP design. Line numbers have been added for discussion purposes. On lines 3–7, the signal name assignments are shown. Lines 12–19 show the configuration information, as follows:

12. Address/Data lines are NM: Non-Multiplexed.

This separates the low-8 address and data lines instead of multiplexing them onto one 8-bit port and separating them with an ALE (Address Latch Enable) signal. In this design the address and data lines are separate so NM is chosen.

13. Data Bus Size: 8

"8" is the only choice in the PSD311 (you can choose 8 or 16 in the PSD301 part).

14. CS/A19: A19

This double-duty pin can be used to power down the PAD when the CSI input is held high. The DSP design does not use this feature so A19 was selected, making this pin a general purpose input.

15. Reset Polarity: LO

This was made the same polarity as the ADSP-2105 reset so the two could be connected together.

16. ALE Polarity: HI

ALE is not used in a non-multiplexed design, but it must still be accounted for. ALE must be declared HI or LO, and then tied HI or LO to make the address latch "transparent". HI was chosen and tied the ALE pin HI.

17. WRD/RWE: WRD

Selects separate strobes for $\overline{\text{RD}}$ and $\overline{\text{WR}}$, rather than R/W and Enable pins. This enables direct interface to the ADSP-2105 RD and WR inputs.

18. A16-A19 Trans: T

A16–A19 are used as general purpose inputs so they are configured to be "transparent", i.e. non-latched.

19. Using different... N

The $\overline{\text{RD}}$ signal is used for both the PSD311 RAM and EPROM, so the answer to this question is "No." Basically, there are two ways to develop strobe signals for the PSD311's internal RAM and EPROM.

In the "Combined Address Space" option, the $\overline{\text{RD}}$ signal, qualified by the external $\overline{\text{PSEN}}$ signal, is used to enable both the RAM and EPROM. In the "Separate Code and Data Address Spaces" option, the $\overline{\text{RD}}$ signal enables the RAM, but the separate $\overline{\text{PSEN}}$ signal enables the EPROM. This is how 8031-type systems are hooked up.

Memory Space (Cont.)

Since the ADSP-2105 issues a single \overline{RD} strobe for all of its external memory, "NO" was answered to this option. IMPORTANT: Since the \overline{RD} signal is qualified by the \overline{PSEN} signal, the \overline{PSEN} signal must be tied HI (to V_{CC}) when using this option.

Lines 26–33 of Figure 2 show the output configurations for the Port B pins. "CMOS" was chosen over open drain.

Line 37 shows the logic equation for the inverter. The $\overline{CS6}$ signal is shown as it's alias name, "HD," but the A15 signal is shown as is (it was named HD in figure 1.)

Lines 46–48 show how the three pins that can be inputs or outputs were assigned, and lines 53 and 55 show the logic equations for the external read and write strobes.

Finally, lines 67–76 show the logic equations that select the EPROM and RAM. "ES0" is the EPROM Strobe for the first 2048-byte EPROM block. The file name "CYRM30.HEX" is entered to show where to find the Intel Hex format file for the data to be programmed into this EPROM block. It's important to remember to "re-compile" the design anytime the EPROM file list (lines 67–74) is changed. The compiler incorporates the hex files into its programming data only when the compiler is run.

Notice that the block size of the PSD311, 4 kilobytes, happen to be the same "block" size as the ADSP-2105 boot blocks, so the logical divisions of ADSP-2105 boot blocks and PSD311 chip selects are the same – one of those happy accidents that occur every so often in engineering design.

Only the first block was needed to begin the design. As the design progresses more or all of the EPROM blocks will probably be used. Blocks are added by entering File Names to the list, or using the same file name and giving different segment start and stop addresses within the file.

"RS0" (line 75) is the RAM chip select equation, and "CSP" (line 76) is the base address for the output port registers.

Note that the signal shown as "A19" is actually the ADSP-2105 \overline{BMS} signal, and thus is LO for the EPROM chip selects (the EPROM is used as boot memory). Also, the "A18" signal is actually the ADSP-2105 DMS signal, which is LO for the RAM and IO chip select equations. Aliases would have been handy for these signals, but the development software does not presently support aliases for all signals.

ADSP – 2105 Timing

The ADSP-2105 offers great timing flexibility in talking to outside peripherals such as the PSD311. Four separate memory spaces may be assigned individual numbers of wait states to accommodate wide timing differences.

The ADSP-2105 "Wait Register" was programmed for 1 wait state, i.e. a 200 ns cycle time, for EPROM, RAM, and external memory strobes. This allows comfortable margins with a PSD311 120 ns part.

Summary

At first glance, the lowest-cost member of the PSD family might not appear to be a match for a high-speed number cruncher like the ADSP-2105. After some analysis, however, it turns out to be a perfect match, making possible a very effective two-chip system. The eight-bit organization of the PSD311 EPROM makes it suitable for implementation as the ADSP-2105 boot memory, and the 8-bit RAM is easily interfaced into the ADSP-2105 data-memory space. The PSD311 I/O section provides welcome system interface bits,

and the programmable address decoders offer the required flexibility to place the various PSD311 resources where they are needed in the ADSP-2105 memory space. If there are pins left over, the PAD can even be used to implement some "stray" logic, if required. Because of the ability of the ADSP-2105 to insert a programmable number of wait states into external accesses, the designer may choose the PSD311 speed necessary for the most cost-effective design.

Appendix B

The .SV1 File

```

1  ALIASES
2
3  /CS8/A16 = XRD
4  /CS9/A17 = XWR
5  /CS10/A18 = DMS
6  /CSI/A19 = BMS
7  /CS6 = HD
8
9  *****
10         GLOBAL CONFIGURATION
11
12  Address/Data Mode:                NM
13  Data Bus Size:                    8
14  CSI/A19:                          A19
15  Reset Polarity:                   LO
16  ALE Polarity:                     HI
17  WRD/RWE:                          WRD
18  A16-A19 Transparent or Latched by ALE:  T
19  Using different READ strobes for SRAM and EPROM: N
20
21  *****
22
23         PORT B CONFIGURATION
24
25  Bit No.    CS/IO.    CMOS/OD.
26     0       IO       CMOS
27     1       IO       CMOS
28     2       IO       CMOS
29     3       IO       CMOS
30     4       IO       CMOS
31     5       IO       CMOS
32     6       CS6      CMOS
33     7       IO       CMOS

```

1

Appendix B
The .SV1
File (Cont.)

34

35

CHIP SELECT EQUATIONS

36

37

HD = /(A15)

38

39

40

41

42

43

PORT C CONFIGURATION

44

45

Bit No. CS/Ai.

46

0 CS8

47

1 CS9

48

2 A18

49

50

51

CHIP SELECT EQUATIONS

52

53

XRD = /(/DMS * /A13 * /A12 * A11 * / RD)

54

55

XWR = /(/DMS * /A13 * /A12 * A11 * / WR)

56

57

58

59



Programmable Peripheral Application Note 025

Interfacing The PSD3XX To The NEURON[®] 3150[™] CHIP

By Dan J. Friedman, WSI and Reza S. Raji, Echelon Corporation

Introduction

Interfacing the PSD3XX to the NEURON 3150 CHIP can increase the capability of the NEURON 3150 CHIP without significantly increasing the board space and power consumption. The PSD3XX enhances the capabilities of the NEURON 3150 CHIP by increasing both its I/O capability and memory capability. By using the PSD3XX, the I/O port capability can be expanded from 11 to 21 I/O ports. This two chip solution will also give the user up to 128K bytes of EPROM with built-in paging logic, 2K bytes of SRAM, and programmable logic for address decoding and integration of any glue logic. This application note describes the process of interfacing the PSD3XX to the NEURON 3150 CHIP.

The two chip solution discussed in this application note was implemented into a Series 100 Distributed Intelligence Controller developed by DMS Systems. Figure 1 shows a picture of the Series 100 board containing the PSD312 and the NEURON 3150 CHIP.

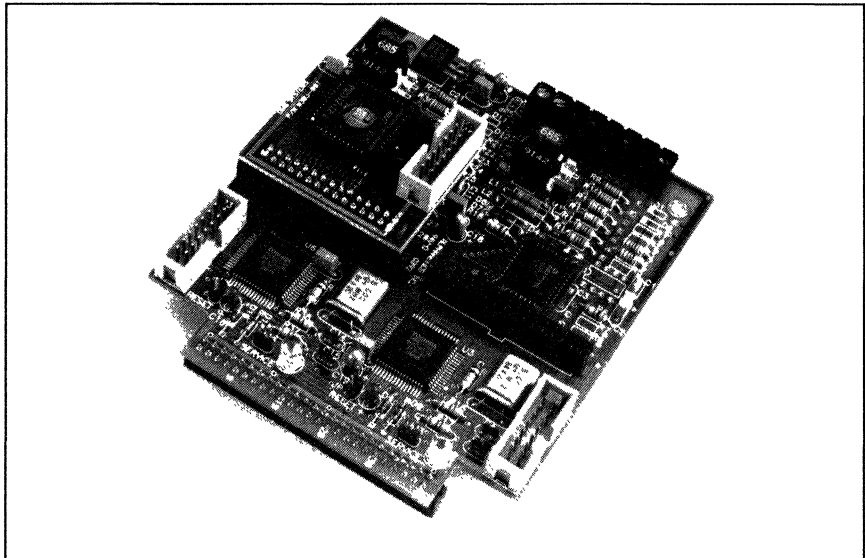
The Series 100 operates industry standard I/O modules and mounting racks. Unlike previous generations of master/slave control systems, the controller operates either as a stand-alone or in a parallel, peer-to-peer network. This allows each board to perform a number of difficult tasks autonomously, while still coordinating with the rest of the system.

Since the Echelon LONWORKS[™] network uses a high performance peer-to-peer protocol, there is no host necessary. Each controller can communicate with any others within the same network. Many networks can be linked to other networks through a router.

1

**Figure 1.
Series 100
Distributed
Intelligence
Controller**

(Courtesy of
DMS Systems)



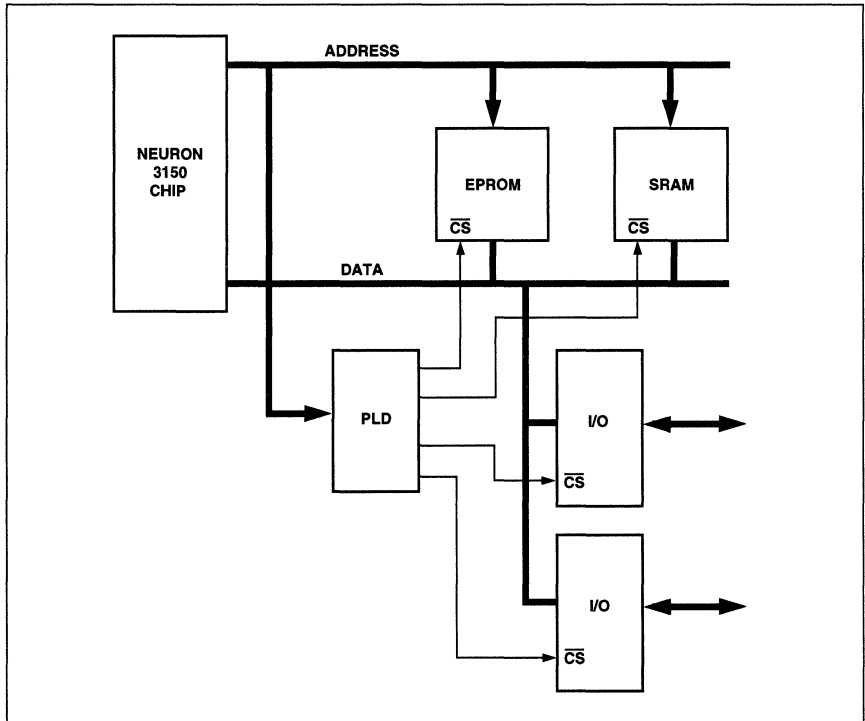
A Typical NEURON 3150 CHIP Design

Figure 2 shows a typical NEURON 3150 CHIP node design before and after the use of a PSD3XX. The Before design includes an EPROM, SRAM, decoder to generate external chip selects, and an I/O port. For applications where space is critical, this implementation may be unacceptable. In the NEURON 3150 CHIP, memory

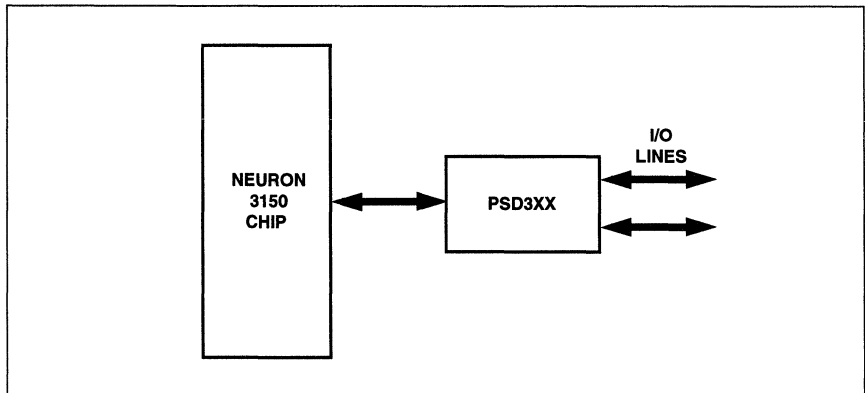
locations E800 through FFFF are reserved for internal use. All external memory must be mapped from 0000 to E7FF. In order to take advantage of the full memory space, an external address decoder to the external memory devices must be incorporated. The After drawing shows a simpler smaller design.

Figure 2. Before and After Interfacing to the WSI PSD3XX

Without PSD3XX



With PSD3XX



NEURON 3150 CHIP and the External Memory Interface

The NEURON 3150 CHIP provides an external memory bus to permit expansion of memory up to 58K bytes beyond the 512 bytes of EEPROM and 2K bytes of RAM resident on the chip. The NEURON 3150 CHIP requires 16K bytes of external non-volatile memory to store its firmware. The remaining 42K bytes of external memory are available for user application program and data.

Assessing Memory Requirements

LONWORKS™ nodes based on the NEURON 3150 CHIP use a combination of three different types of memory:

- Non-Volatile Memory for NEURON CHIP Firmware and, optionally, Application Code.
- Electrically Rewriteable Non-Volatile Memory for Network and Application Code and Data.
- Read/Write Memory for Packet Buffering, or, optionally, Application Code and Data.

A LONWORKS application node may include the external memory types described above by partitioning the available 58K byte memory space into three distinct regions aligned on 256-byte page boundaries. The different memory types do not need to map to contiguous address space. However, the LONBUILDER™ NEURON C compiler enforces the ordering of the types of memory to be ROM/EPROM first, EEPROM second, and finally RAM. The NEURON C compiler and LONBUILDER linker locate parts of an application in appropriate memory regions (see Chapter 6 of the NEURON C Programmer's Guide)

Memory Interface Logical Description

Figure 3 shows the memory map of the NEURON 3150 CHIP. Memory locations from 0 to E7FF are external to the NEURON 3150 CHIP. Access to this memory is through an external memory bus consisting of eight bi-directional three-state data lines, 16 unidirectional address lines driven by the NEURON 3150 CHIP, and two control lines.

The two control lines used for the external memory interface are:

\bar{E} – Enable Clock

This output is a strobe driven by the NEURON 3150 CHIP to synchronize the external bus. Its frequency is one-half that of the input clock or crystal. \bar{E} is low during the second half of the memory cycle, which indicates that the NEURON 3150 CHIP is actively reading or writing data. During write cycles, the NEURON 3150 CHIP drives the new data onto the data bus during the time \bar{E} is low. During read cycles, the NEURON 3150 CHIP clocks in the external data on the transition of \bar{E} .

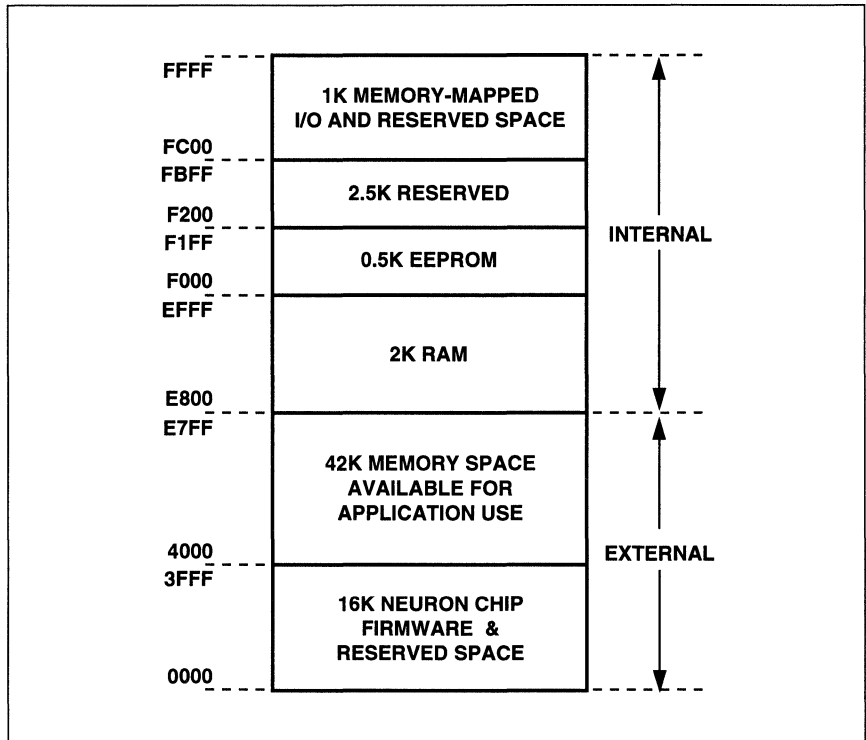
R/\bar{W} – Read/Write

This output indicates the direction of the data bus. It is set by the NEURON 3150 CHIP to high during a Read cycle, and low on a Write cycle. R/\bar{W} changes state during the time \bar{E} is high, and is stable during the time \bar{E} is low.

See the section on Special Timing Considerations for more information on the NEURON 3150 CHIP memory interface requirements.

1

Figure 3.
The NEURON
3150 CHIP
Memory
Map



PSD3XX
Architecture

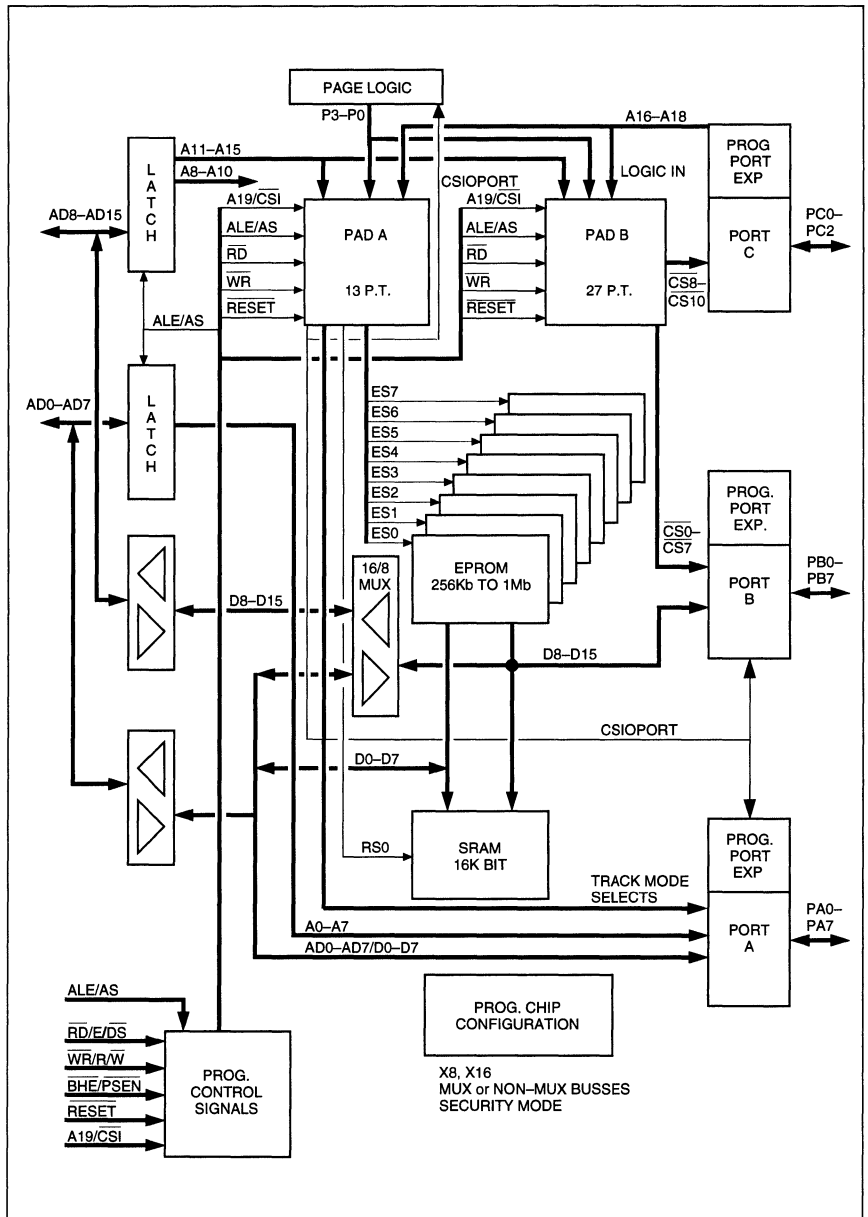
The PSD3XX integrates high performance user-configurable blocks of EPROM, SRAM, and programmable logic technology to provide a single chip microcontroller interface. The major functional blocks as shown in Figure 4 include two programmable logic arrays, Programmable Address Decoder (PAD A and PAD B), 256K bits to 1M bits of EPROM, 16K bits of SRAM, input latches, and output ports. The PSD3XX is ideal for applications requiring high performance, low power, and very small form factors.

The PSD3XX offers a unique single-chip solution for users of the NEURON 3150 CHIP that need more memory-mapped I/O, larger EPROM and SRAM size, external chip selects, and programmable logic. Table 1 summarizes the PSD3XX devices that can interface to the NEURON 3150 CHIP. The PSD3XXL devices can operate down to 3.0 V for low power applications.

As shown in Figure 5, WSI's PSD3XX can efficiently interface with, and enhance, the NEURON 3150 CHIP. This is the first solution that provides the NEURON 3150 CHIP with port expansion, page logic, two programmable logic arrays (PAD A and PAD B), 256K bits to 1M bits of EPROM, and 16K bits of SRAM on a single chip. The PSD3XX does not require any glue logic for interfacing to the NEURON 3150 CHIP.

The PSD3XX on-chip PAD A enables the user to map the I/O ports, eight segments of EPROM (8K x 8 each) and SRAM (2K x 8) anywhere in the address space of the NEURON 3150 CHIP. PAD B can implement up to 4 sum-of-product expressions based on address inputs, control signals, and other external input signals.

Figure 4.
PSD3XX
Architecture



1

**Table 1.
PSD3XX
Devices**

Device	I/O Ports	EPROM (Bits)	SRAM (Bits)	Data Path (Bits)	Supply Voltage
PSD312	19	512 K	16 K	8	5 V
PSD312L	19	512 K	16 K	8	3 V – 5 V
PSD313	19	1024 K	16 K	8	5 V
PSD313L	19	1024 K	16 K	8	3 V – 5 V

**PSD3XX
Architecture
(Cont.)**

The Page Register extends the accessible address space of the NEURON 3150 CHIP from 64K Bytes to 1 M Bytes. There are 16 pages that can serve as base address inputs to the PAD, thereby enlarging the address space of the NEURON 3150 CHIP by a factor of 16. Paging is not supported by the NEURON chip firmware or LONBUILDER tools and must therefore be managed entirely by the application program.

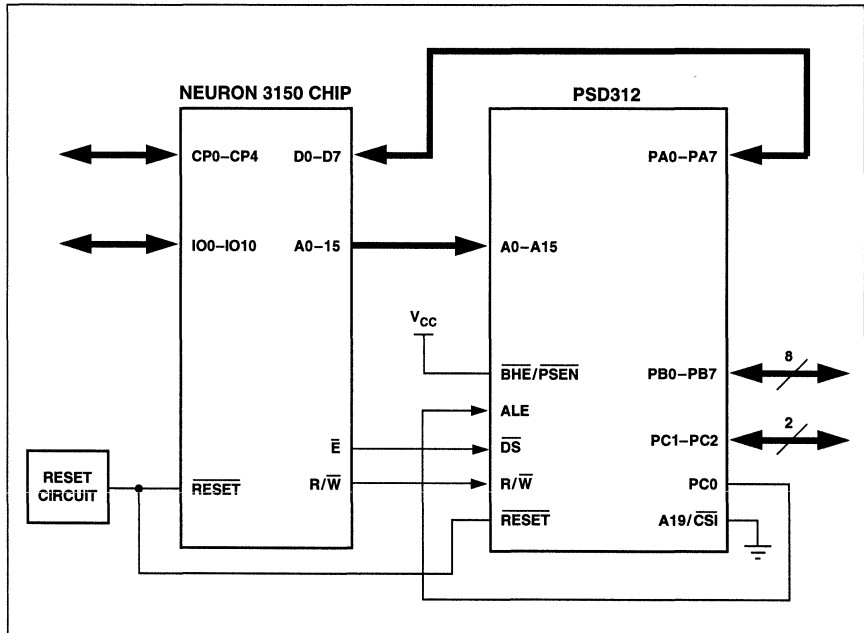
The 8-bit Data Bus. The low-order address/data bus (AD0/A0-AD7/A7) is the low-order address input bus. The high-order address/data bus (A8-A15) is the high-order address bus byte. Port A is the low-order data bus. External logic is required to interface with the PSD311. Therefore, it is recommended that the PSD312 or PSD313 be used.

Figure 5 shows how to interface the PSD312 or PSD313 to the NEURON 3150 CHIP. The PSD3XX is operated in the Non-Multiplexed Address/Data Mode with

Programmable Address Decoder (PAD)

The PSD3XX consists of two programmable arrays referred to as PAD A and PAD B. PAD A is used to generate chip select signals derived from the input address to the internal EPROM blocks, SRAM, and I/O ports.

**Figure 5.
Interfacing
The PSD312
To The NEURON
3150 Chip**



- Integrating the PSD312 to the NEURON 3150 CHIP adds:
- 10 Chip Selects or Data I/O Ports (in addition to the 11 I/O on the 3150).
 - 64K bytes of EPROM (expandable to 128K bytes).
 - 2K bytes of SRAM.
 - All Decode Logic for External Chip Selects and Internal Memory.



**PSD3XX
Architecture
(Cont.)**

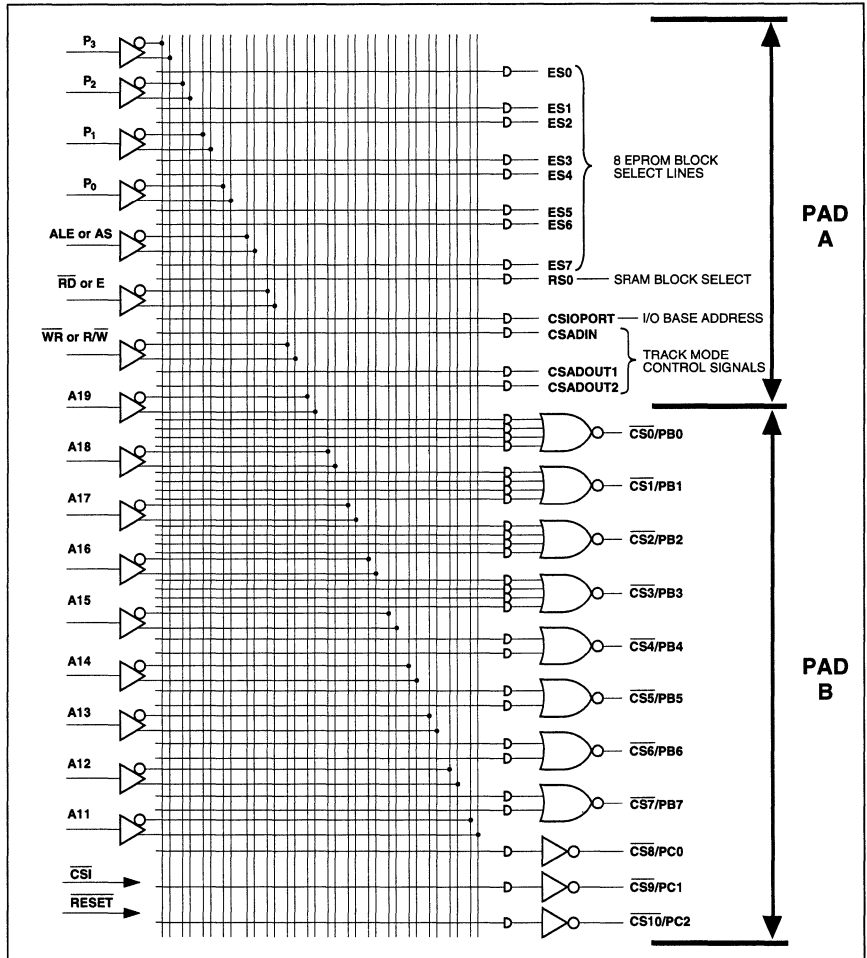
PAD B can be used to extend the decoding to select external devices or as a random logic replacement. The input bus to both PAD A and PAD B is the same. Using WSI's MAPLE software, each programmable bit in the PAD's array can have one of three logic states of 0, 1, and don't care (X). In a user's logic design, both PADs can share the same inputs using the X for input signals that are not supposed to affect other functions. The PADs use reprogrammable CMOS EPROM technology and can be programmed and erased by the user. Figure 6 shows the PSD3XX PAD description.

Port Functions

The PSD3XX has three I/O ports (Port A, B, and C) that are configurable at the bit level.

Port A – When interfacing to the NEURON 3150 CHIP, Port A is used for the lower order data bus.

**Figure 6.
PSD3XX
PAD
Description**



- NOTES:**
1. \overline{CSi} is a power-down signal. When high, the PAD is in stand-by mode and all its outputs become non-active.
 2. RESET deselects all PAD output signals
 3. A18, A17, and A16 are internally multiplexed with $\overline{CS10}$, $\overline{CS9}$, and $\overline{CS8}$, respectively. Either A18 or $\overline{CS10}$, A17 or $\overline{CS9}$, and A16 or $\overline{CS8}$ can be routed to the external pins of Port C. Port C can be configured as either input or output.



**PSD3XX
Architecture
(Cont.)**

Port B – 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 FF, in Figure 7). As an output, the pin level can be controlled by writing into the respective pin's data flip flop (DFF, in Figure 7). 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 level 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 Table 2.

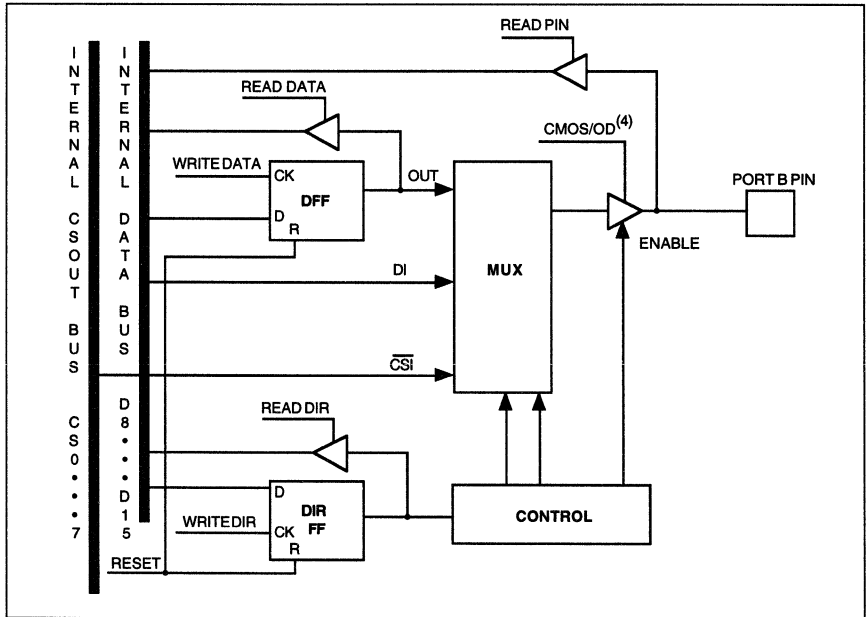
Alternatively, each bit of Port B can be configured to provide a chip-select output signal from PAD B, PB0 – PB7 can provide CS0 – CS7, respectively. Each of the signals CS0 – CS3 is comprised of four product terms. Thus, up to four ANDed expressions can be Ored while deriving any of these signals. Each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

Accessing the I/O Port – Table 2 shows the offset values with the respect to the base address defined by the CSIOPORT. They let the user access the corresponding registers.

Port C in all Modes – Each pin of Port C (shown in Figure 8) can be configured as an input to PAD A and PAD B or output from PAD B. 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 address lines or logic inputs to PAD A and PAD B. For example, A8-A10 can also be connected to those pins, improving the boundaries of CS0 – CS7 resolution to 256 bytes. As inputs, they can be individually configured to be logic or address inputs. A logic input uses the PAD only for Boolean equations that are implemented in any or all of the CS0 – CS10 PAD B outputs. Port C addresses can be programmed to latch the inputs by the trailing edge ALE or to be transparent.

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

**Figure 7
Port B
Pin Structure**



NOTE: 4. CMOS/OD determines whether the output is open drain or CMOS.

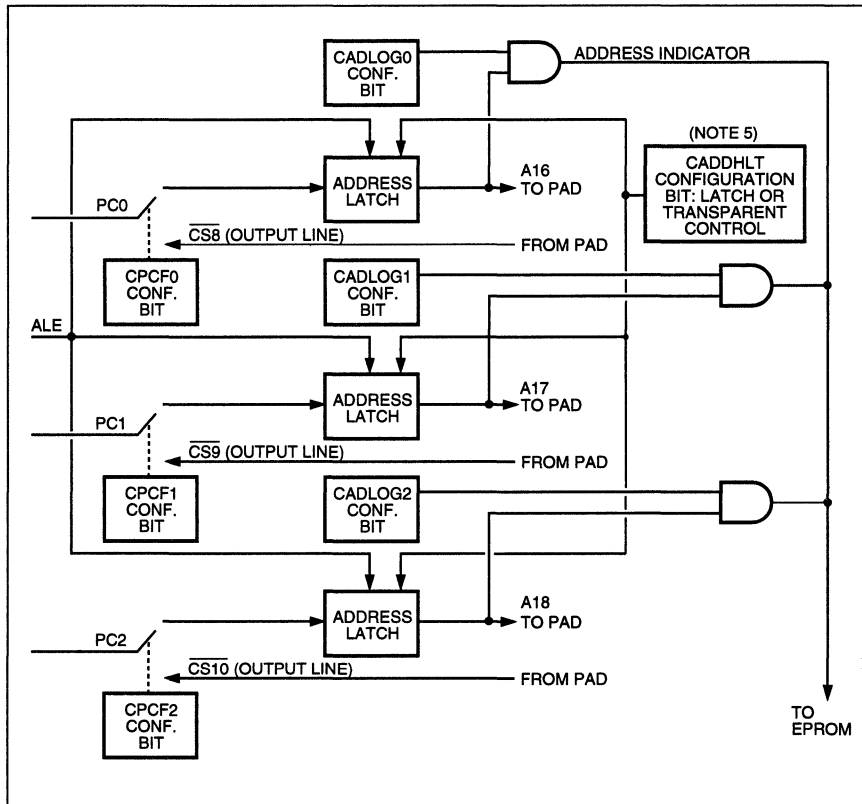


Table 2.
I/O Port
Addresses in
an 8-bit Data
Bus Mode

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

1

Figure 8.
Port C
Structure



NOTE: 5. 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.

PSD3XX Architecture (Cont.)

EPROM

The PSD3XX has 256K bits to 1M bits of EPROM and is organized from 32K x 8 to 128K x 8. 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 from 4K x 8 to 16K x 8.

SRAM

The PSD3XX has 16K bits of SRAM and is organized as 2K x 8. The SRAM is selected by the RS0 output of the PAD.

Control Signals

The PSD3XX control signals are \overline{WR} or R/\overline{W} , $\overline{RD}/E/\overline{DS}$, ALE, \overline{PSEN} , RESET, and A19/ \overline{CSI} . Each of these signals can be configured to meet the output control signal requirements of the NEURON 3150 CHIP.

\overline{WR} or R/\overline{W} – The \overline{WR} or R/\overline{W} pin is configured as R/\overline{W} . This pin works with the \overline{DS} strobe of the $\overline{RD}/E/\overline{DS}$ pin. When R/\overline{W} is high, an active low signal on the \overline{DS} pin performs a read operation. When R/\overline{W} is low, an active low signal on the \overline{DS} pin performs a write operation.

$\overline{RD}/E/\overline{DS}$ – The $\overline{RD}/E/\overline{DS}$ pin is configured as \overline{DS} . This pin works with the R/\overline{W} signal as an active low data strobe signal. As \overline{DS} , the R/\overline{W} defines the mode of operation (Read or Write). The \overline{DS} feature is not available on the PSD311 and PSD301. The E input must be used. To generate to correct polarity, an external inverter must be used. *To minimize board space and to meet critical timing requirements, it is recommended to use the PSD312 or PSD313 with the NEURON 3150 CHIP.*

ALE – To prevent a timing violation with the Address Hold time, the ALE input pin is used to latch the address into the PSD3XX. As shown in Figure 5, PC0 output signal from Port C on the PSD3XX is connected to the ALE input to the PSD3XX. The PC0 output signal is a delayed version of the \overline{E} signal from the NEURON 3150 CHIP. Further information on this special timing condition is discussed after Figure 10.

\overline{PSEN} – The \overline{PSEN} signal is not used with the NEURON 3150 CHIP and therefore must be connected to V_{CC} .

RESET – This is an asynchronous input pin that clears and initializes the PSD3XX/3XXL. On the PSD3XX, reset polarity is programmable (active low or active high). Whenever the PSD3XX reset input is driven active for at least 100 ns, the chip is reset. On the PSD3XXL, reset is a low signal only. This device is reset and operational only after the reset input is driven low for at least 500 ns followed by another 500 ns period after the reset becomes high. In either device, the part is not automatically reset internally during boot-up and an external reset procedure is recommended for best results. Tables 3 and 4 indicate the state of the part during and after reset.

A19/ \overline{CSI} – When configured as \overline{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 PSD3XX states during the power-down mode, see Tables 5, 6, and Figure 9. The contents of the SRAM is preserved during the power-down mode. There is an Application Note on the Power-Down Mode in the Programmable Peripherals Design and Applications Handbook from WSI.

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

**Table 3.
Signal States
During and After
Reset**

<i>Signal</i>	<i>Configuration Mode</i>	<i>Condition</i>
AD0/A0–AD7/A7	All	Input
A8–A15	All	Input
PA0–PA7) (Port A)	I/O Tracking AD0/A0–AD7 Address outputs A0–A7	Input Input Low
PB0–PB7 (Port B)	I/O $\overline{CS7}$ – $\overline{CS0}$ CMOS outputs $\overline{CS7}$ – $\overline{CS0}$ open drain outputs	Input High Tri-stated
PC0–PC2 (Port C)	Address inputs A16–A18 $\overline{CS8}$ – $\overline{CS10}$ CMOS outputs	Input High

**Table 4.
Internal States
During and After
Reset**

<i>Component</i>	<i>Signals</i>	<i>Contents</i>
PAD	$\overline{CS0}$ – $\overline{CS10}$	All = 1 (Note 13)
	CSADIN, CSADOUT1, CSADOUT2, CSIOPORT, RS0, ES0 – ES7	All = 0 (Note 13)
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: 13. All PAD outputs are in a non-active state.

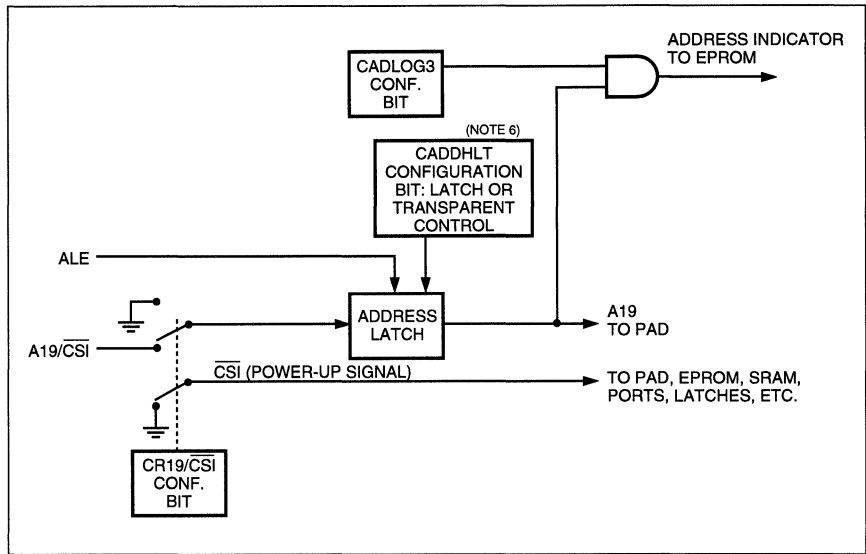
**Table 5.
Signal States
During
Power-Down
Mode**

<i>Signal</i>	<i>Configuration Mode</i>	<i>Condition</i>
AD0/A0–AD7/A7	All	Input
A8–A15	All	Input
PA0–PA7)	I/O Tracking AD0/A0–AD7 Address outputs A0–A7	Unchanged Input All 1's
PB0–PB7	I/O $\overline{CS7}$ – $\overline{CS0}$ CMOS outputs $\overline{CS7}$ – $\overline{CS0}$ open drain outputs	Unchanged All 1's Tri-stated
PC0–PC2	Address inputs A16–A18 $\overline{CS8}$ – $\overline{CS10}$ CMOS outputs	Input All 1's

**Table 6.
Internal States
During Power
Down**

<i>Component</i>	<i>Signals</i>	<i>Contents</i>
PAD	$\overline{CS0}$ – $\overline{CS10}$	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	

**Figure 9.
A19/CSI
Cell
Structure**



NOTE: 6. The CADDHLT configuration bit determines if A19 – A16 are transparent via the latch, or if they must be latched by the trailing edge of the ALE strobe.

Page Register

The page register consists of four flip-flops, which can be read from, or written to, through the I/O address space (CSIOPORT). The page register is connected to the D3-D0 lines. The Page Register address is CSIOPORT + 18H. The page register outputs are P3-P0, which are fed into the PAD. This enables the host microcontroller to enlarge its address space by a factor of 16 (there can be a maximum of 16 pages). See Figure 10. There is an Application Note from WSI that discusses how to use the Paging

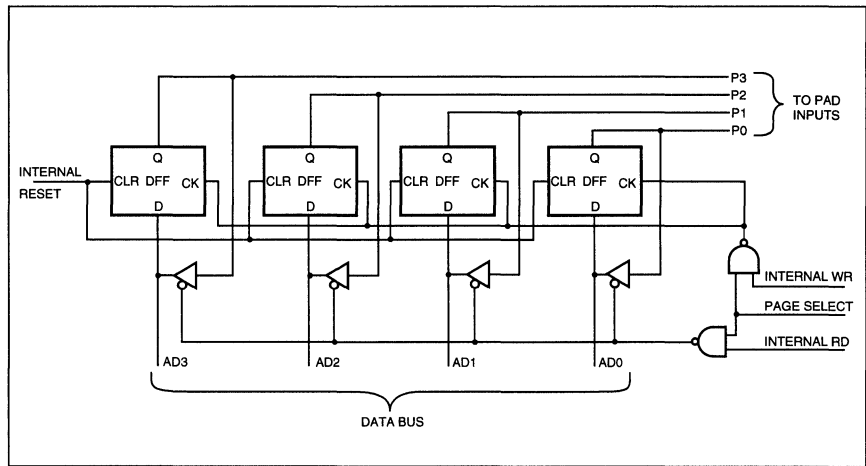
Register (see References). Because of the flexibility of the programmable logic in the PSD3XX, some blocks of EPROM can be common to each page while other blocks of EPROM can be unique to each page. The SRAM and I/O ports can be programmed to be either common to all pages or unique to a specific page. Since the paging logic is transparent to the NEURON 3150 CHIP, the NEURON C application program running on the NEURON 3150 CHIP must be designed to use this feature.

Security Mode

The Security Mode in the PSD3XX locks the contents of the PAD A, PAD B and all the configuration bits. The EPROM, SRAM, and I/O contents can be accessed only through the PAD. The Security Mode can be set by the MAPLE or Programming software. In the window packages, the mode is erasable through UV full part erasure. In the security mode, the PSD3XX contents cannot be copied on a programmer. Because of the high integration of the address decoding, eight blocks of EPROM, and SRAM, it is difficult to copy the contents of the EPROM in-circuit. The SRAM can be mapped dynamically over

the EPROM, protecting the contents of the EPROM. The internal page register can be used to map different EPROM blocks onto different pages. This would make it difficult for someone to externally sequence through the address space and capture the code on the MCU bus with a logic analyzer. Because of the flexibility of the PSD3XX, other protection schemes are possible to protect the contents of the EPROM along with the configuration of the PSD3XX from being copied.

Figure 10.
Page
Register

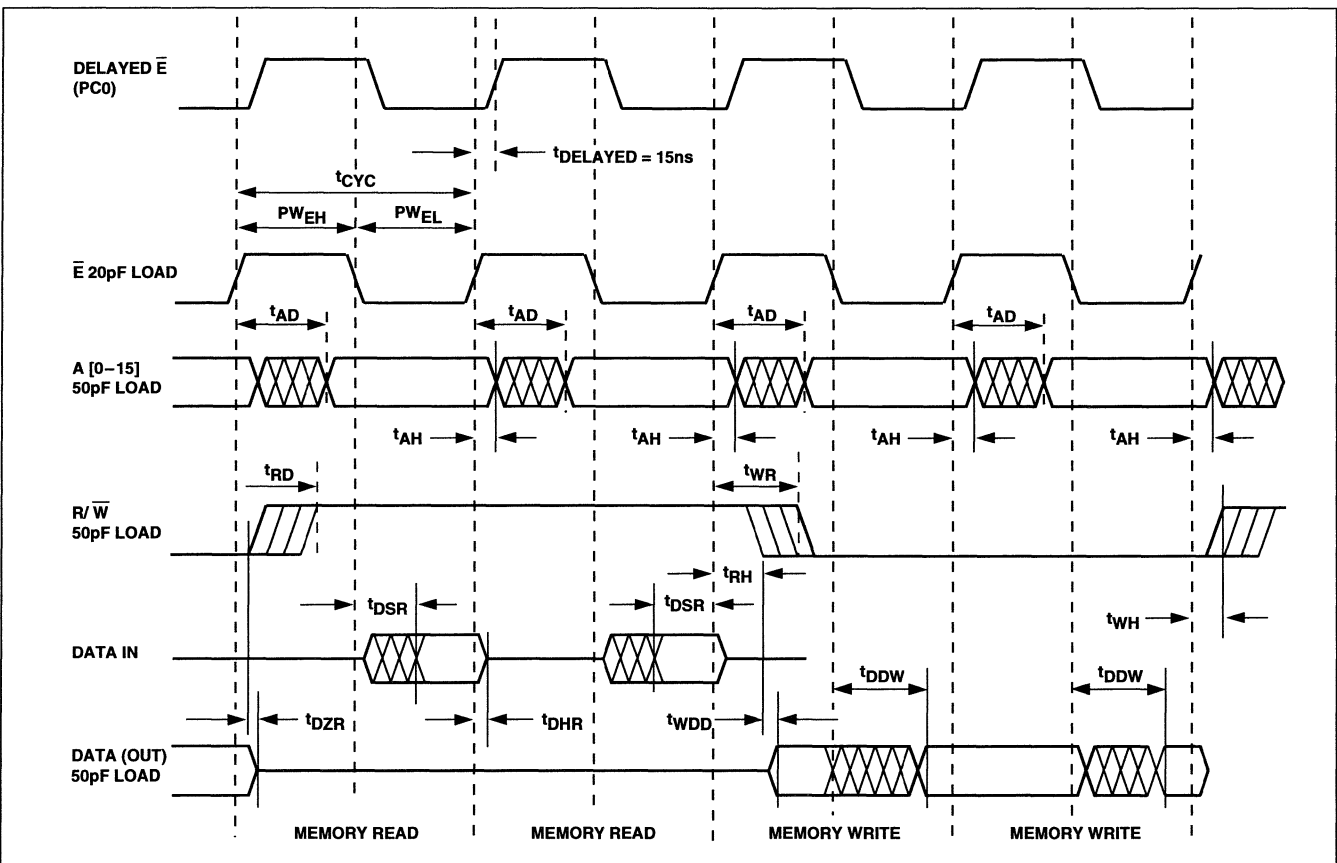


Special
Timing
Considerations

When interfacing the PSD3XX to the NEURON 3150 CHIP, a potential Address Hold time violation may occur (t_{AH} in Figure 11). The minimum Address Hold Time requirement of the PSD3XX is 15 ns. The maximum Address Hold Time of the NEURON 3150 CHIP is 7 ns. To prevent this timing violation from occurring under worst case conditions, the \bar{E} signal from the NEURON 3150 CHIP is delayed through the PSD3XX and connected to the

ALE input as shown in Figure 5. The \bar{E} signal is connected to the \bar{DS} input on the PSD3XX. This input is also used as a logic input to the PAD. The \bar{E} signal is delayed for 15 ns by feeding it through the internal PAD, and out PC0. PC0 is connected to the ALE input in-order to latch and hold the address input and meet the internal Address Hold time requirement in the PSD3XX.

Figure 11.
NEURON 3150
CHIP Memory
Interface
Timing
Diagram



Development Process

The PSD3XX features a complete set of System Development Tools. These tools provide an integrated, easy-to-use software and hardware environment to support PSD3XX 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.

The configuration of the PSD3XX device is entered using MAPLE software. The MAPLE output listing of a PSD312 configured to interface to the NEURON 3150 CHIP is shown on the next few pages. Once the PSD3XX is configured, the configuration information along with the EPROM code is compiled into one file with an “.obj” extension. This file is used to program a PSD3XX device on WSI’s MagicPro Programmer or on a third party programmer that supports the PSD3XX.

As shown on the MAPLE output listing “echelon.sv1”:

PSD Selected:
PSD312

Bus Interface:
Non-multiplexed bus, 8-bit, with $\overline{R\overline{W}}$ and \overline{DS} , signals.

Port A:
PA7-PA0 are used as the data bus interface (D7-D0) on the NEURON 3150 CHIP.

Port B:
PB7-PB0 can be used as Data I/O or Chip Selects. Each pin can be individually configured.

Port C:

PC2-PC1 can be configured as Logic inputs, or Chip Select outputs. PC0 is used as a Chip Select output and is connected to the ALE input on the PSD3XX. The Chip Select equations is $CS8 = \overline{DS}$. The E signal is only delayed through the PAD. The logic of this signal is not changed.

The PSD312 contains 64K x 8 of EPROM but only 54K bytes are used. The SRAM (RSO) and I/O Ports (CSP) can mapped over the EPROM. The portion of EPROM that overlaps the SRAM and I/O Ports cannot be used. Table 7 shows the defined Memory Map in this example.

Note that the upper 2K bytes of EPROM Block (ES6) is mapped in the same address space as the I/O Ports (in the range of D800-DFFF). Because of the overlap, the portion of EPROM from D800-DFFF cannot be accessed.

The NEURON 3150 CHIP’S memory map is defined through the Memory Properties screen of the LONBUILDER Software. The amount of each type of memory used, ROM, EEPROM, RAM, and memory mapped I/O is entered in this screen so that they match the actual external memory connected to the NEURON 3150 CHIP. The values for this example entered into the Memory Properties screen are shown in Table 8. Refer to the LONBUILDER User’s Guide for more information.

**Table 7.
Memory Map
Example**

Address Range	Size (Bytes)	Memory Type	Physical Location
0 – D7FF	54 K	EPROM	PSD312
D800 – DFFF	2 K	Memory-Mapped I/O	PSD312
E000 – E7FF	2 K	SRAM	PSD312
E800 – EFFF	2 K	RAM	NEURON 3150 CHIP
F000 – F1FF	0.5 K	EEPROM	NEURON 3150 CHIP
F200 – FBFF	2.5 K	Reserved	NEURON 3150 CHIP
FC00 – FFFF	1 K	Memory-Mapped I/O and Reserved	NEURON 3150 CHIP

**Table 8.
Memory
Properties
Screen of the
LONBUILDER**

Memory Type	Number of Pages	Start Address	End Address
ROM	215	0000	D7FF
EEPROM	0	–	–
RAM	8	E000	E7FF
I/O	8	D800	DFFF

**MAPLE
Output
Listing**

```

***** MAPLE 5.10 *****
PSD PART USED: PSD312
*****PROJECT INFORMATION*****

Project Name   : = Echelon WSi Integration
Your Name     : = Dan Friedman
Date          : = 10/8/92
Host Processor: = 3150

*****
*****ALIASES*****
*****
*****GLOBAL CONFIGURATION*****

Address/Data Mode :   NM
Data Bus Size     :   8
Reset Polarity    :   LO
Security          :   OFF
AS Polarity       :   HI
A15-A0 AS depend (Y) or Transparent (N) : Y
Are you using PSEN ? (Y/N) : N

*****
*****READ WRITE CONTROL*****
R/(/W) and /DS

*****Port A CONFIGURATION*****
Port A is Data Bus D0-D7

*****PORT B CONFIGURATION*****

Pin    CS/IO    CMOS/OD
PB0    IO      CMOS
PB1    IO      CMOS
PB2    IO      CMOS
PB3    IO      CMOS
PB4    IO      CMOS
PB5    IO      CMOS
PB6    IO      CMOS
PB7    IO      CMOS

*****
*****PORT B CHIP SELECT EQUATIONS*****
*****PORT C CONFIGURATION*****

Pin    CS/Ai    LOGIC/ADDR
PC0    CS8
PC1    CS9
PC2    CS10
A19    CSI

*****
*****PORT C CHIP SELECT EQUATIONS*****
/CS8 = /(/DS)

```

1

**MAPLE
Output
Listing
(Cont.)**

*****ADDRESS MAP*****

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	FILE	FILE	File Name
	19	18	17	16	15	14	13	12	11	STRT	STOP	STRT	STOP	
ES0	N	N	N	N	0	0	0	N	N	0	1fff	0	1fff	ECH_TEST.HEX
ES1	N	N	N	N	0	0	1	N	N	2000	3fff	2000	3fff	ECH_TEST.HEX
ES2	N	N	N	N	0	1	0	N	N	4000	5fff	4000	5fff	ECH_TEST.HEX
ES3	N	N	N	N	0	1	1	N	N	6000	7fff	6000	7fff	ECH_TEST.HEX
ES4	N	N	N	N	1	0	0	N	N	8000	9fff	8000	9fff	ECH_TEST.HEX
ES5	N	N	N	N	1	0	1	N	N	a000	bfff	a000	bfff	ECH_TEST.HEX
ES6	N	N	N	N	1	1	0	N	N	c000	dfff	c000	dfff	ECH_TEST.HEX
ES7	N	N	N	N				N	N					
RS0	N	N	N	N	1	1	1	0	0	e000	e7ff	N/A	N/A	N/A
CSP	N	N	N	N	1	1	0	1	1	d800	dfff	N/A	N/A	N/A

*****END*****

*****ADDRESS MAP (EQUATIONS)*****

ES0 = /A15 * /A14 * /A13
 ES1 = /A15 * /A14 * A13
 ES2 = /A15 * A14 * /A13
 ES3 = /A15 * A14 * A13
 ES4 = A15 * /A14 * /A13
 ES5 = A15 * /A14 * A13
 ES6 = A15 * A14 * /A13
 RS0 = A15 * A14 * A13 * /A12 * /A11
 CSP = A15 * A14 * /A13 * A12 * A11

*****ADDRESSES OF I/O PORTS*****

Direction Register of Port A : D804
 Data Register of Port A : D806
 Pin Register of Port B : D803
 Direction Register of Port B : D805
 Data Register of Port B : D807
 Page Register : D818



MAPLE Output Listing (Cont.)

*****CONFIGURATION BITS*****

CDATA =	0	CADDRDAT =	0
CA19 (/CSI) =	0	CALE =	0
CRESET =	0	(/COMB)/SEP) =	0
CPAF2 =	0	CADDHLT =	0
CSECURITY =	0	CLOT =	1
CRRWR =	1	CEDS =	1
CADLOG19 =	0		
CPAF1[0] =	1	CPACOD[0] =	0
CPAF1[1] =	1	CPACOD[1] =	0
CPAF1[2] =	1	CPACOD[2] =	0
CPAF1[3] =	1	CPACOD[3] =	0
CPAF1[4] =	1	CPACOD[4] =	0
CPAF1[5] =	1	CPACOD[5] =	0
CPAF1[6] =	1	CPACOD[6] =	0
CPAF1[7] =	1	CPACOD[7] =	0
CPBF[0] =	1	CPBCOD[0] =	0
CPBF[1] =	1	CPBCOD[1] =	0
CPBF[2] =	1	CPBCOD[2] =	0
CPBF[3] =	1	CPBCOD[3] =	0
CPBF[4] =	1	CPBCOD[4] =	0
CPBF[5] =	1	CPBCOD[5] =	0
CPBF[6] =	1	CPBCOD[6] =	0
CPBF[7] =	1	CPBCOD[7] =	0
CPCF[0] =	1	CPCF[1] =	1
CPCF[2] =	1		
CADLOG[0] =	0	CADLOG[1] =	0
CADLOG[2] =	0		

1

References

- WSI, *Programmable Peripherals Design and Applications Handbook*, 1992.
- Jeff Miller, *Using Memory Paging with the PSD3xx*, Programmable Peripheral Application Note 015.
- Echelon, *NEURON CHIP Data Book*.
- Echelon, *NEURON 3150 CHIP External Memory Interface*, LONWORKS Engineering Bulletin, August 1991.
- Echelon, *NEURON C Programmer's Guide*, 29300.
- Echelon, *LONBUILDER User's Guide*, 29200.
- Echelon, *LONWORKS Custom Node Development and Engineering Bulletin*, 005-0024-01.
- Motorola, *NEURON CHIP Distributed Communications and Control Processor*, MC14315, MC143120.
- Toshiba, *NEURON CHIP TMPN3150/3120 Data Book*.

Echelon Corporation
4015 Miranda Avenue
Palo Alto, California 94304
Tel: (415) 855-7400
(800) 258-4LON
Fax: (415) 856-6153

DMS Systems
1570 East Edinger Avenue, Suite 5
Santa Ana, California 92680
Tel: (714) 541-7362
Fax: (714) 541-7366

ECHELON and NEURON are U.S. registered trademarks of Echelon Corporation.
LONWORKS, LONBUILDER and 3150 are trademarks of Echelon Corporation.



Programmable Peripheral Application Note 026

PSD3XX Device Fit for PC Notebook Applications: Keyboard, Power Management and Auxiliary Peripherals Control

By Karen Spesard

Introduction

Typical laptop/notebook computers operating DOS or Windows have at least two micros on their boards. One is obviously the 80X86 microprocessor and the other is usually an 8042 or 80C51/31. The 8042 (the original AT standard keyboard controller) or 80C51/31 may also be called the SCP or System Control Processor because it usually handles much of the system user interface including keyboard, mouse-trackball, external keyboard, and pen input controls. In addition it provides the interface to both the PS2 and AT buses, and because of power consumption concerns, the SCP also usually handles battery/power management while controlling the backlight and contrast of the display.

Many designers of these control systems have chosen to turn away from the 8042 in favor of the 80C51 for a few reasons. For example, it is more powerful, provides additional memory capacity, has more I/O, and is much lower in power than the 8042. Using the 80C51, however, isn't always straightforward because most versions of the 80C51 don't have the complete SCP 8042-type 80X86 interface necessary for these applications. Therefore, there have usually been two options considered by hardware engineers.

The first option is to use a standard 80C51 and design an ASIC or discrete multi-chip implementation of components which incorporate the 80X86 interface for the SCP along with keyboard control, address decoding, etc.. The 80X86 interface requires a bidirectional status latch and two directional data latches. In most cases, external memory consisting of at least 8K bytes of EPROM and 1K byte of SRAM is also needed. In some cases, FLASH EPROM is used, although until recently, it was associated mainly with the core 80X86 processor.

The second option is to use the 80C51SL-12 controller from Intel in conjunction with a peripheral device from WSI's PSD3XX family. The 80C51SL-12 is a special version of the 80C51 which has been available for about 2 years, but used extensively in notebook applications for only about 6 months. It includes the 8042-type 80X86 interface along with keyboard control and other functions. The 80C51SL has 256 bytes of internal SRAM, has many I/Os and is housed in a 100-pin PQFP package. For time-to-market reasons, most versions of the 80C51SL are shipped "ROMless" and used with external EPROM.

Although the 80C51SL has many I/Os, many hardware designers still consider it lacking in enough I/O resources. It only has 8 user pins and all the rest are dedicated to specific functions. For instance, there is the standard bus interface, 2 auxiliary serial ports for external keyboards and mouse, 24-pins dedicated to scanning the keyboard matrix, parallel mailbox port on the AT bus, 4 analog inputs for A/D conversions, etc.

1

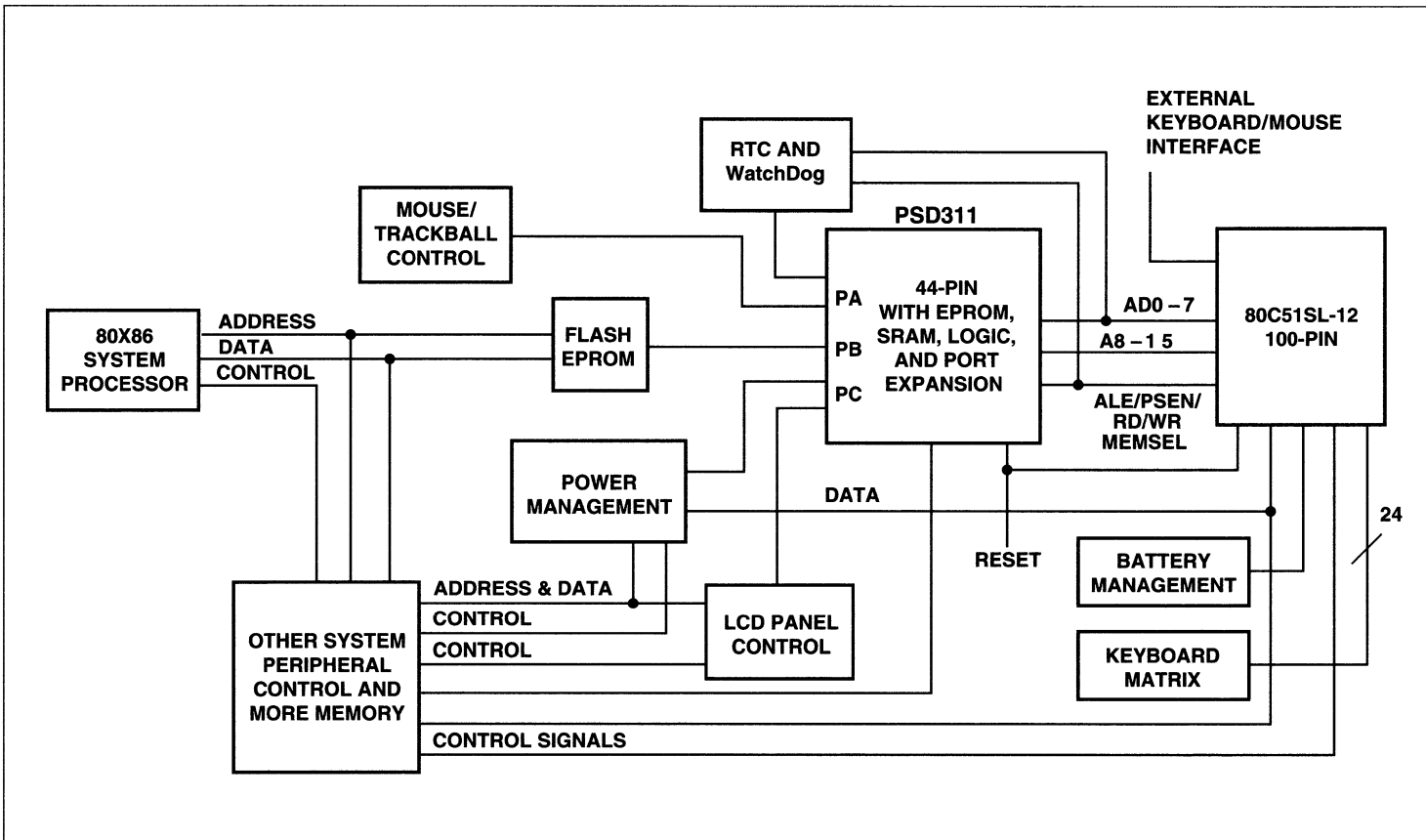
Overview

A PSD3XX family device perfectly complements the 80C51SL because it incorporates additional I/Os with port expansion and EPROM and SRAM memory. As shown in Figure 1, the PSD311 in particular provides an additional 2K bytes of SRAM in addition to the 256 bytes of SRAM already on-board the 80C51SL. It also integrates 32K bytes of EPROM. (In many cases, less than 32K bytes of EPROM is actually used.) It also provides the address decoding, 16 general-purpose I/Os, up to 11 programmable logic outputs and 4 general-purpose logic inputs. The PSD311 interfaces to the battery and power management units, a real time clock with WatchDog timer, and E² potentiometers to control LCD panel contrast/brightness levels. It monitors the suspend/resume pushbutton option through its ports, the battery power on/off control, and power sequencing for the display. The SRAM in the system is used, among other things, to store table information on the current state of the capacity remaining in the battery.

This second approach seems to be the preferred approach for notebook applications for a number of reasons. Board space is always so expensive and the board densities so high. The PSD311/80C51SL chip-set is an attractive cost-effective solution that provides as much integration as possible to keep the board size to a minimum. It also is configurable and can be used in many core designs, speeding time-to-market and eliminating the costly need to design a different ASIC with a standard 80C51 for each application. Finally, it is very low in power and helps to keep power consumption down. PSD3XX devices are available in 44-pin CLDCC/PLDCC as well as 52-pin PQFP packages and will soon be available in lower height PCMCIA TQFP packages as well. The PSD3XX devices are also available at 3.3 V and can be used with the 80C51SL when it becomes available at lower power supply voltages.

Attached, you will find an example PSD311 configuration file that can be used in an application similar to the one shown in the general block diagram of Figure 1.

Figure 1.
Block Diagram
of the WSI
PSD311/80C51SL
in a Notebook
Computer -
Keyboard/Power
Management
Application



**Example of
PSD311
Configuration
for PC Notebook
Applications**

***** MAPLE 5.10 *****

ALIASES

/CS10/A18 = CE1283
/CSI/A19 = 51MCS
/IO0 = 315RESET
/IO1 = 322RESET
/IO2 = LB
/IO3 = SENSE+5
/IO4 = MSEL
/IO5 = SILLB
/IO6 = PDNOFF
/IO7 = ROMRGNL

GLOBAL CONFIGURATION

Address/Data Mode: MX
Data Bus Size: 8
CSI/A19: CSI
Reset Polarity: HI
ALE Polarity: HI
WRD/RWE: WRD
A16-A19 Transparent or Latched by ALE: T
Are you using PSEN? Y
Separate Data and Program address spaces: Y

PORT A CONFIGURATION (Address/IO)

Bit	No.	Ai/IO.	CMOS/OD.
	0	A0	CMOS
	1	A1	CMOS
	2	A2	CMOS
	3	A3	CMOS
	4	A4	CMOS
	5	A5	CMOS
	6	IO	CMOS
	7	IO	CMOS

PORT B CONFIGURATION

Bit	No.	CS/IO.	CMOS/OD.
	0	IO	CMOS
	1	IO	CMOS
	2	IO	CMOS
	3	IO	CMOS
	4	IO	CMOS
	5	IO	CMOS
	6	IO	CMOS
	7	IO	CMOS

CHIP SELECT EQUATIONS



**Example of
PSD311
Configuration
for PC Notebook
Applications (Cont.)**

PORT C CONFIGURATION

Bit	No.	CS/Ai.
	0	CS8
	1	CS9
	2	CS10

CHIP SELECT EQUATIONS

CE1283 = /(A15 * /A14 * /A13 * /A12 * /A11)

ADDRESS MAP

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	EPROM	EPROM	File Name
	19	18	17	16	15	14	13	12	11	STRT	STOP	START	STOP	
ES0	N	N	N	N	0	0	0	0	N	0	FFF	0	FFF	80C51SL.HEX
ES1	N	N	N	N	0	0	0	1	N	1000	1FFF	1000	1FFF	80C51SL.HEX
ES2	N	N	N	N	0	0	1	0	N	2000	2FFF	2000	2FFF	80C51SL.HEX
ES3	N	N	N	N	0	0	1	1	N	3000	3FFF	3000	3FFF	80C51SL.HEX
ES4	N	N	N	N					N					
ES5	N	N	N	N					N					
ES6	N	N	N	N					N					
ES7	N	N	N	N					N					
RS0	N	N	N	N	1	0	1	0	0	A000	A7FF			
CSP	N	N	N	N	1	0	1	0	1	A800	AFFF			

***** ADDRESS MAP EQUATIONS *****

ES0 = /A15 * /A14 * /A13 * /A12
 ES1 = /A15 * /A14 * /A13 * A12
 ES2 = /A15 * /A14 * A13 * /A12
 ES3 = /A15 * /A14 * A13 * A12
 ES4 =
 ES5 =
 ES6 =
 ES7 =
 RS0 = A15 * /A14 * A13 * /A12 * /A11
 CSP = A15 * /A14 * A13 * /A12 * A11

***** END *****



Programmable Peripheral Application Note 027

Simplification of Logic Networks in the PSD3XX PAD Using DeMorgan's Theorem

By Don Buccini

1

Introduction

One purpose of WSI's PSD family is to elevate the design task of interfacing external peripherals and memory to a microcontroller, from a hardware level to a software or system level, similar to what the single-chip microcontroller did for its multi-chip predecessor. Architecturally, the PSD3XX family of Programmable System Devices contains configurable logic to interface with virtually any 8- or 16-bit microcontroller with no "glue logic" required. Also included in the PSD are EPROM, SRAM and I/O Ports. These functions are interconnected by the internal Programmable Address Decoder (PAD) whose primary purpose is to generate all of the internal and external system Chip Selects.

The PAD is more versatile than just generating chip selects. By using DeMorgan's Theorem, relatively complex combinatorial logic networks can be minimized in the PAD, eliminating several logic chips. The intent of this Application Note is to demonstrate how to simplify logic in the PAD without using design entry software tools. A working knowledge of the PSD3XX by the reader is assumed.

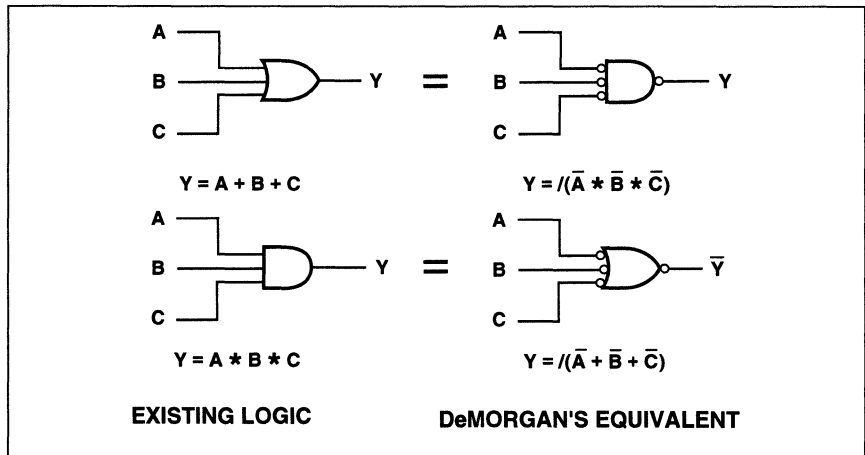
DeMorgan's Theorem

DeMorgan's Theorem describes the transformation of conjunctions (AND) to disjunctions (OR) by use of negation, or vice-versa. Simply put, it allows the inversion of an entire logic equation by using the following theorem:

$$\overline{(A * B)} = \overline{A} + \overline{B} \text{ and } \overline{(A + B)} = \overline{A} * \overline{B}$$

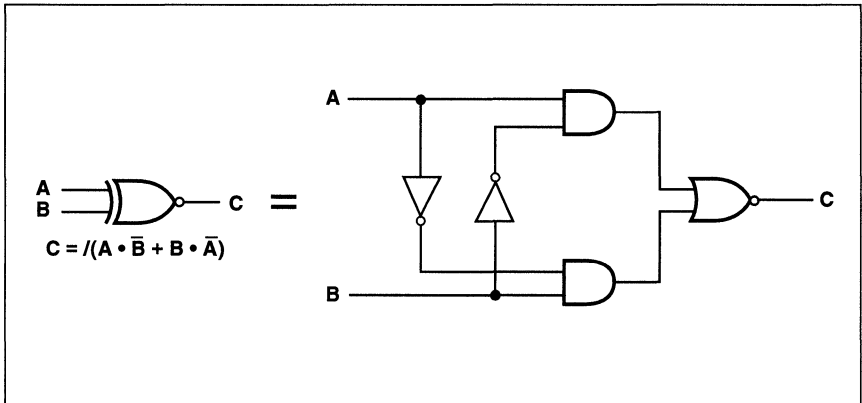
Figure 1 is an example of how negation is used to transform an OR function to a NAND function, and an AND function to a NOR function. Figure 2 shows how a more complex logic function is converted to its simpler elements. The logic functions of a complex network can be converted to functions which are recognized by the selected logic array.

Figure 1.



DeMorgan's Theorem

Figure 2. Exclusive NOR Equivalent



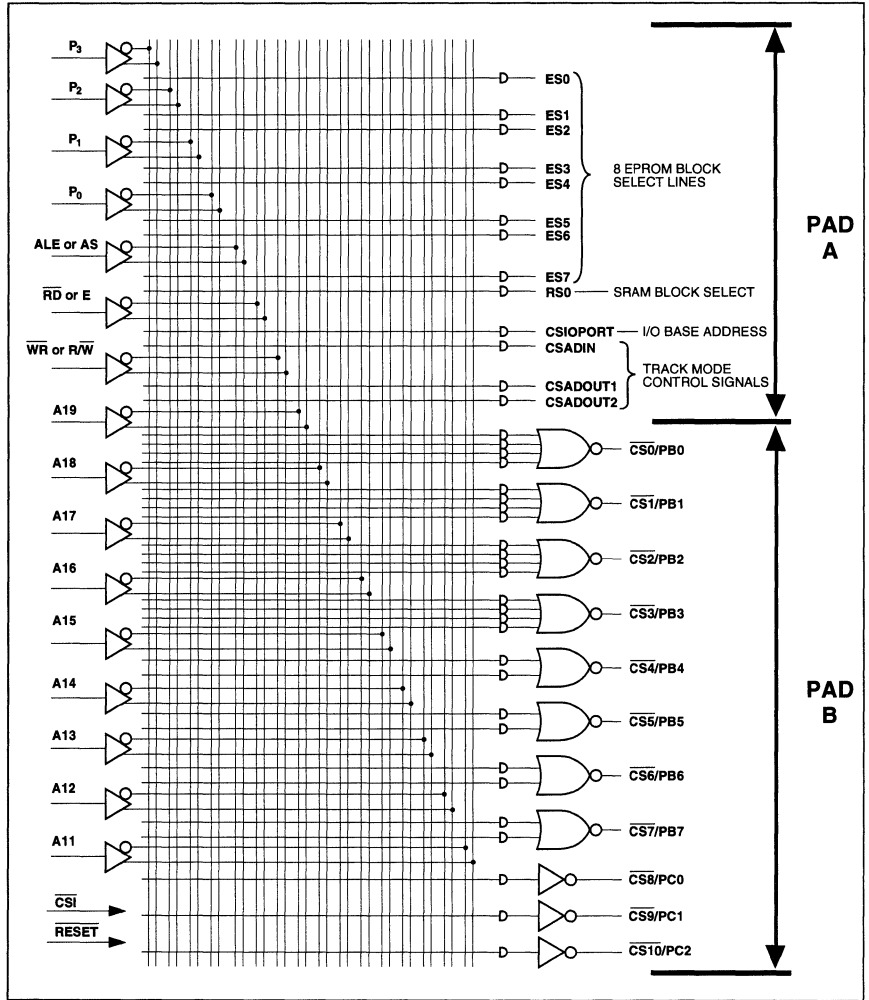
PAD Architecture

The PAD Logic Diagram is shown in Figure 3. There are 12 inputs (16 including the 4-bit Page Register of the PSD3X2 and PSD3X3) and 24 outputs. Port C (A16 – A18), ALE/AS and A19/CSI are independent of the microcontroller interface and can be configured as general purpose logic inputs to the PAD, dependent on how the PSD3XX is configured and which microcontroller is used. Of the 40 product terms available to the user, PAD A uses 13 of the terms for internal selection of the EPROM, SRAM, Track Mode and I/O Ports. The remaining 27 product terms are used by PAD B for outputs (24, if Port C is configured as inputs).

The logic functions of the PAD consist of NOR, AND (each product term input is a multiple input AND gate) and Inverter elements. DeMorgan's Theorem converts dissimilar logic functions to these three elements for inclusion into the PAD. Figure 4 illustrates how a typical AND/OR network is converted into the PSD3XX PAD AND/NOR/Inverter equivalent.

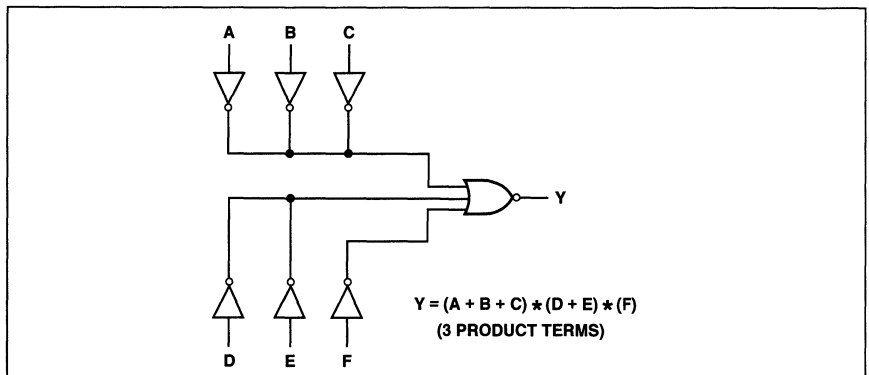
PAD Architecture
(Cont.)

Figure 3. PAD Architecture



1

Figure 4. PSD3XX PAD NOR Equivalent



Logic Network Minimization

To show how effectively a logic network can be minimized in the PAD, DeMorgan's Theorem will be used to reduce the multi-level logic of Figure 5 to a series of equations for the Port B output pins. To obtain the maximum number of logic inputs to the PAD, an 8-bit microcontroller with separate address and data buses was selected to free up the ALE/AS function for use as a logic input. Four pins of Port B are configured as logic outputs; the other four pins can be used as chip selects and/or I/O pins.

The logic network of Figure 5 is converted into its PAD equivalent as illustrated in Figure 6. A sequence of "x" represents one of the many multi-input AND functions of each product term. When the conversion is completed, the logic network is reduced to the following four equations:

$$CS0 = /A16 * (A18 + A19 + AS) * (A17 + /A19)$$

$$CS1 = A16 * (/A18 + A19 + AS) * (/A17 + /AS)$$

$$/CS4 = (/A18 * A19) + (A18 * /A19)$$

$$/CS5 = (A18 * /AS) + (/A18 * AS)$$

Since the PAD consists of the sum of product terms into an inverting logic gate (NOR), equations CS0 and CS1 must be converted from their present format of product of sum terms into a non-inverting logic gate (AND). Using DeMorgan's Theorem, the converted equations in the PAD are entered as follows:

$$/CS0 = A16 + (/A18 * /A19 * /AS) + (/A17 * A19)$$

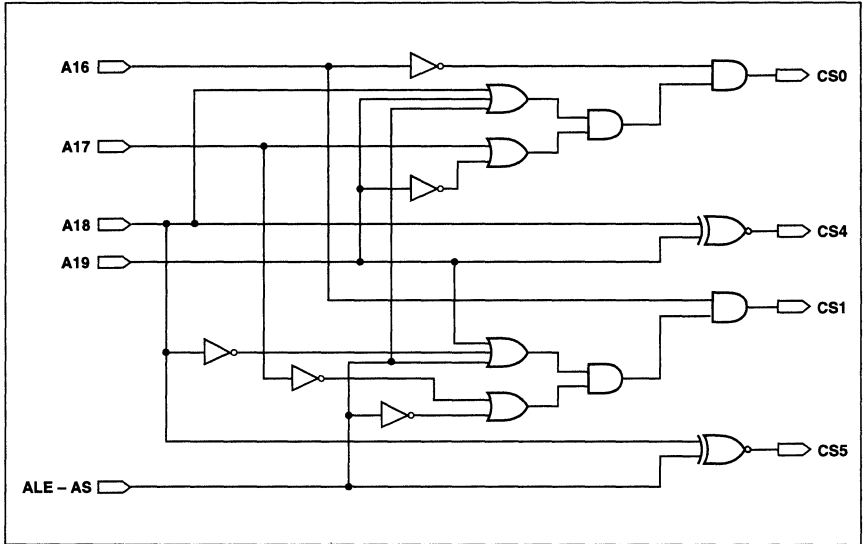
$$/CS1 = /A16 + (A18 * /A19 * /AS) + (A17 * AS)$$

Equations /CS4 and /CS5 are already in a format acceptable to the PAD and do not have to be converted.

Although the Port B outputs appear to be active low at first glance, these outputs can be programmed to be active high, as shown above, by assigning a logic input to each of the NOR product terms. If more than four signals are needed to control an output, two or more outputs can be configured as Open Drains and OR-tied together. The PSD312 Maple ASCII software Configuration file (.SV1) for this example is found in Appendix A.

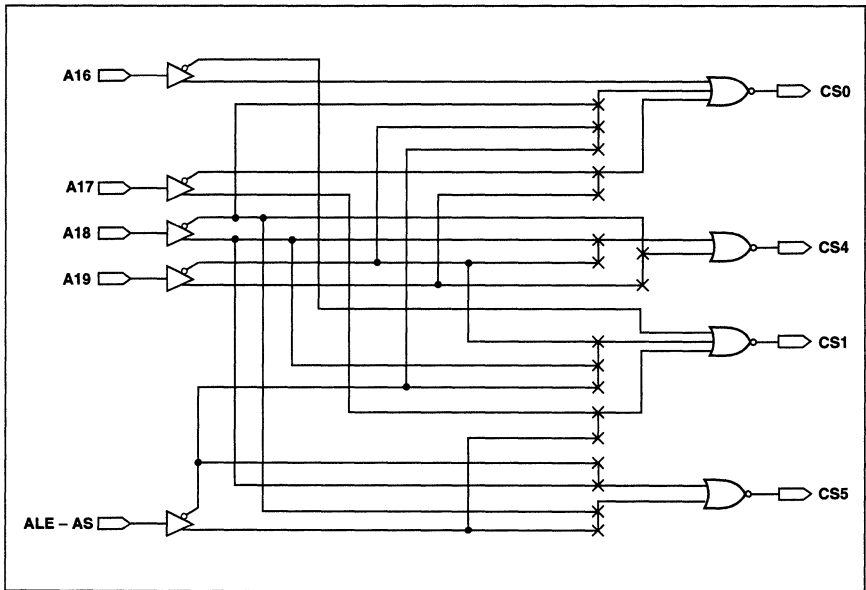
**Logic
Network
Minimization
(Cont.)**

Figure 5. Multi-Level Logic Network



1

Figure 6. PAD Conversion Multi-Level Logic Network



Appendix A.
PSD312
Configuration

```
***** MAPLE 5.00 *****
PSD PART USED: PSD312
*****PROJECT INFORMATION*****
Project Name   : = PAD Logic Reduction
Your Name     : = Don Buccini
Date          : = March 31, 1993
Host Processor: = 8-Bit Non-Mux'd Bus
*****
*****ALIASES*****
*****
*****GLOBAL CONFIGURATION*****
Address/Data Mode:  NM
Data Bus Size   :    8
Reset Polarity :    LO
Security        :    OFF
AS Polarity     :    LO
A15-A0 AS dependent (Y) or Transparent (N):  N
Are you using PSEN ? (Y/N)           :  N
*****

*****READ WRITE CONTROL*****
R/(/W) and E

*****

*****Port A Configuration *****
Port A is Data Bus D0-D7.
```



Appendix A.
PSD312
Configuration
(Cont.)

*****PORT B CONFIGURATION*****

Pin	CS/IO	CMOS/OD
PB0	CS0	CMOS
PB1	CS1	CMOS
PB2	IO	CMOS
PB3	IO	CMOS
PB4	CS4	CMOS
PB5	CS5	CMOS
PB6	IO	CMOS
PB7	IO	CMOS

*****PORT B CHIP SELECT EQUATIONS*****

/CS0 =
 / (A16
 + /A19 * /A18 * / AS
 + A19 * /A17
)

/CS1 =
 / (/A16
 + /A19 * A18 * / AS
 + A17 * AS
)

/CS4 =
 / (A19 * /A18
 + /A19 * A18
)

/CS5 =
 / (A18 * / AS
 + /A18 * AS
)

*****PORT C CONFIGURATION*****

Pin	CS/Ai	LOGIC/ADDR
PC0	A16	LOGIC
PC1	A17	LOGIC
PC2	A18	LOGIC
A19	A19	LOGIC

*****PORT C CHIP SELECT EQUATIONS*****

Appendix A.
PSD312
Configuration
(Cont.)

*****ADDRESS MAP*****

	A	A	A	A	A	A	A	A	A	SEGMENT	SEGMENT	FILE	FILE	File Name	Page	Reg	Q.F
	19	18	17	16	15	14	13	12	11	STRT	STOP	STRT	STOP		3210	AS	
ES0	X	X	X	X	0	0	0	N	N			0000	1FFF	NON_MUX.TXT			
ES1	X	X	X	X	0	0	1	N	N			2000	3FFF	NON_MUX.TXT			
ES2	X	X	X	X	0	1	0	N	N			4000	5FFF	NON_MUX.TXT			
ES3	X	X	X	X	0	1	1	N	N								
ES4	X	X	X	X	1	0	0	N	N								
ES5								N	N								
ES6								N	N								
ES7								N	N								
RS0	X	X	X	X	1	1	0	0	0			N/A	N/A	N/A			
CSP	X	X	X	X	1	1	0	1	0			N/A	N/A	N/A			

*****END*****

*****ADDRESS MAP (EQUATIONS)*****

ES0 = /A15 * /A14 * /A13
 ES1 = /A15 * /A14 * A13
 ES2 = /A15 * A14 * /A13
 ES3 = /A15 * A14 * A13
 ES4 = A15 * /A14 * /A13
 RS0 = A15 * A14 * /A13 * /A12 * /A11
 CSP = A15 * A14 * /A13 * A12 * /A11



**Appendix A.
PSD312
Configuration
(Cont.)**

```

*****CONFIGURATIONBITS*****
CDATA=          0          CADDRDAT =    0
CA19/(/CSI)=   1          CALE =          1
CRESET=         0          (/COMB)/SEP)=  0
CPAF2=          0          CADDHLT=       0
CSECURITY=      0          CLOT=          0
CRRWR=          1          CEDS=          0
CADLOG19=       0

CPAF1[0]=       1          CPACOD[0]=    0
CPAF1[1]=       1          CPACOD[1]=    0
CPAF1[2]=       1          CPACOD[2]=    0
CPAF1[3]=       1          CPACOD[3]=    0
CPAF1[4]=       1          CPACOD[4]=    0
CPAF1[5]=       1          CPACOD[5]=    0
CPAF1[6]=       1          CPACOD[6]=    0
CPAF1[7]=       1          CPACOD[7]=    0

CPBF[0]=        0          CPBCOD[0]=  0
CPBF[1]=        0          CPBCOD[1]=  0
CPBF[2]=        1          CPBCOD[2]=  0
CPBF[3]=        1          CPBCOD[3]=  0
CPBF[4]=        0          CPBCOD[4]=  0
CPBF[5]=        0          CPBCOD[5]=  0
CPBF[6]=        1          CPBCOD[6]=  0
CPBF[7]=        1          CPBCOD[7]=  0

CPCF[0]=        0          CPCF[1] =    0
CPCF[2]=        0

CADLOG[0]=      0          CADLOG[1] =    0
CADLOG[2]=      0
    
```

1



Conclusion

The versatility of the PSD3XX PAD in replacing several ICs in a multi-level logic network becomes more apparent as the designer applies it to his/her specific application. Its simplicity negates the need for design entry tools. It is also possible to use one or more of PAD inputs A11 – A15 as logic inputs if they are not used for address lines in systems requiring 32K bytes or less of memory. The functionality of the PAD will be greatly expanded as the PSD family of future devices grows.



Programmable Peripheral Application Note 032

Use A ROM Emulator For Rapid Software Debug Of A PSD3XX-Based Design

By Don Buccini

Introduction

EPROM Emulation has become a relatively easy and inexpensive method to software debug a small to medium size microcontroller system which uses external EPROMs. The EPROM is removed from its socket and the RAM-based Emulator is plugged into the socket, allowing rapid on-line changes and verification of program code. The PSD3XX family incorporates EPROM, SRAM, I/O Ports and PLD logic in a single package whose pinouts differ from conventional EPROMs, preventing direct access to the program code.

This Application Note describes an effective way to gain direct access to the on-chip EPROM without the need for special circuitry on the system circuit board, using any commercially available ROM Emulator. This allows quick software debug of the system under test by running the program in RAM and enabling rapid code changes via the keyboard of a PC. The remaining functions configured in the PSD3XX – SRAM, I/O, Chip Selects, etc. – are transparent to the emulator and unaffected by its use. Familiarity with the PSD3XX Architecture and the MAPLE software is assumed.

Design Philosophy

The design philosophy is straightforward - gain access to the required interface signals (A0 – A15, D0 – D7, \bar{R} , \bar{W} , E, ALE/AS, etc.) between the selected microcontroller and the PSD3XX, bypass the EPROM chip selects on the system PSD3XX and program a second PSD3XX (Emulator PSD3XX) to access the ROM Emulator memory being used to run the program code. The recent availability of specialized test socket adapters such as Emulation Technology's Bug Katcher (P/N BC 4-44-PCC3-0000), shown in Figure 1, allows easy access to these interface signals without modifying the design of the system circuit board.

The system PSD3XX is removed from its socket and the test adapter is inserted between the socket and the System PSD3XX. A short cable is connected between the interface pins of the adapter and a small printed circuit adapter board which contains the second PSD3XX and an EPROM socket(s). This board can be designed for both an 8-bit and 16-bit data bus, although more complex 16-bit designs would most probably use an In-Circuit Emulator for debugging.

The following design procedure uses some of the more popular microcontrollers for illustration:

1. Design The ROM Emulator Adapter Board.

1) 8-Bit Multiplexed Address/Data Bus...Figure 2 shows an Interface Diagram for the two components necessary for this bus interface – the Simulator PSD3XX and a 32-pin JEDEC EPROM socket. Port A is used to latch the low order address byte from the microcontroller. PB7 is used as address line A16 to the EPROM Socket, utilizing the Page Register logic of the PSD3X2 and PSD3X3 for memory page switching. PB6 is used to generate the \bar{RD} signal which is converted to the \bar{OE} pin of the EPROM. This same schematic can be used for the 80C31 family by making the following changes to the PSD3XX interface signals:

- Change E to \bar{RD}
- Change AS to ALE
- Disconnect BHE/PSEN from V_{CC} and connect to PSEN

**Design
Philosophy
(Cont.)**

- 2) 16-Bit Multiplexed Address/Data Bus... (see Figure 3) for this application, an additional 74HC373/573 latch is required to latch the high order address byte from the microcontroller. Two EPROM sockets are shown, although one socket could be used if the Emulator can plug into a 64K x 16 socket.
 - 3) 8-Bit Separate Address and Data Bus... Figure 4 is an Interface Diagram similar to Figure 2, except that Port A is not needed to separate the Address and Data Buses.
2. Configure all the parameters on the system PSD3XX except the 8 internal EPROM Chip Selects (ES0 – ES7). These remain inactive and will be emulated by the Emulator PSD3XX. The level of integration and ease of programming of the PSD3XX allow these temporary configuraton changes that are necessary to use ROM Emulation without impacting the existing system hardware design.
 3. In the Emulator PSD3XX, configure the eight product terms of Port B pins 0 and 1 (OD configuration) to correspond to the internal Chip Select (ES0 – ES7) addresses of the EPROM memory map of the System PSD3XX selected by the system software. This is the combined Chip Select signal to the external memory socket and divides the external emulation memory into eight independent blocks identical to the PSD3XX. The three Port C pins are configured for the Memory and I/O signals required by the specific Microcontroller. All other configuration parameters remain inactive.

In summary, the main function of the ROM Simulator Adapter Board is to allow the use of a ROM Simulator without any board modification, divide the external memory into eight blocks identical to the PSD3XX internal EPROM, and enable all other functions of the System PSD3XX to operate in a normal mode.

Figure 1. Bug Katcher™ – PLCC

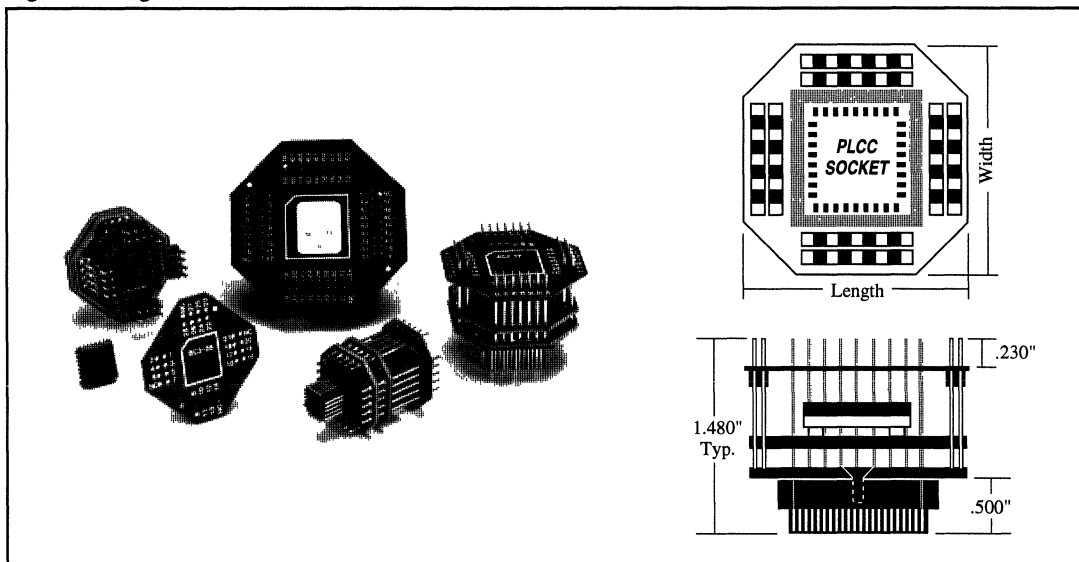


Figure 2. Interface Diagram – 8-Bit Multiplexed Address/Data Bus

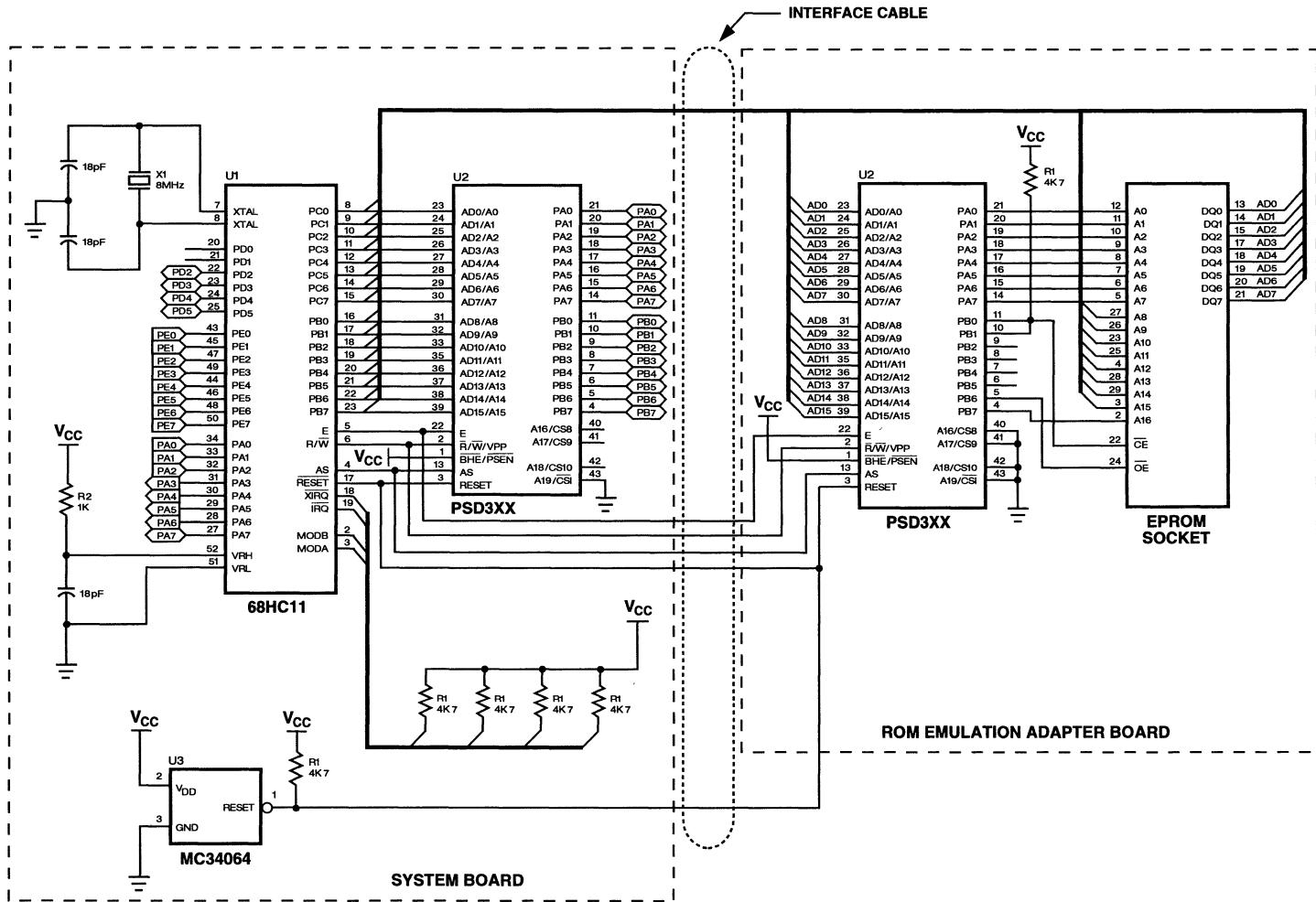


Figure 3. Interface Diagram - 16-Bit Multiplexed Address/Data Bus

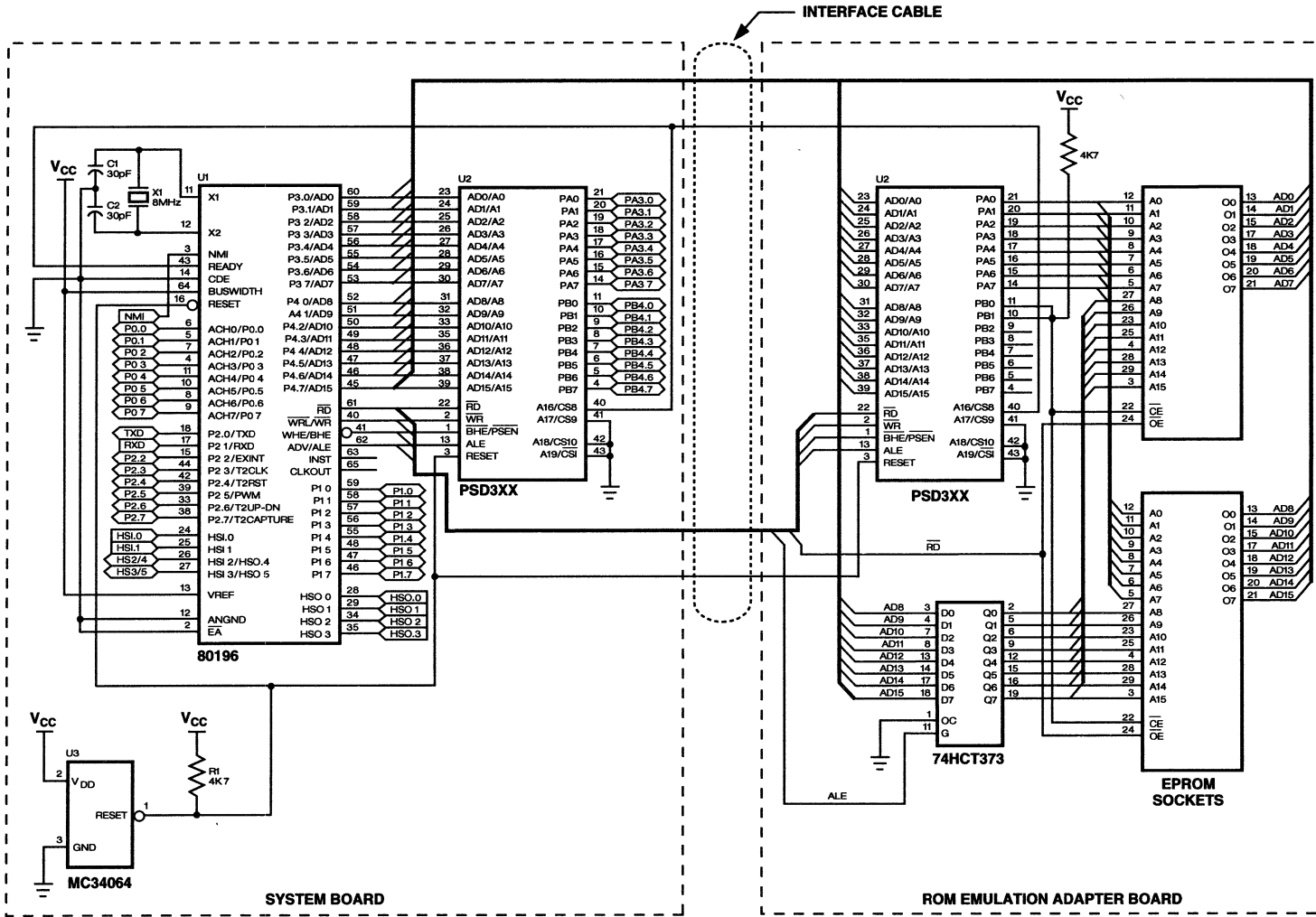
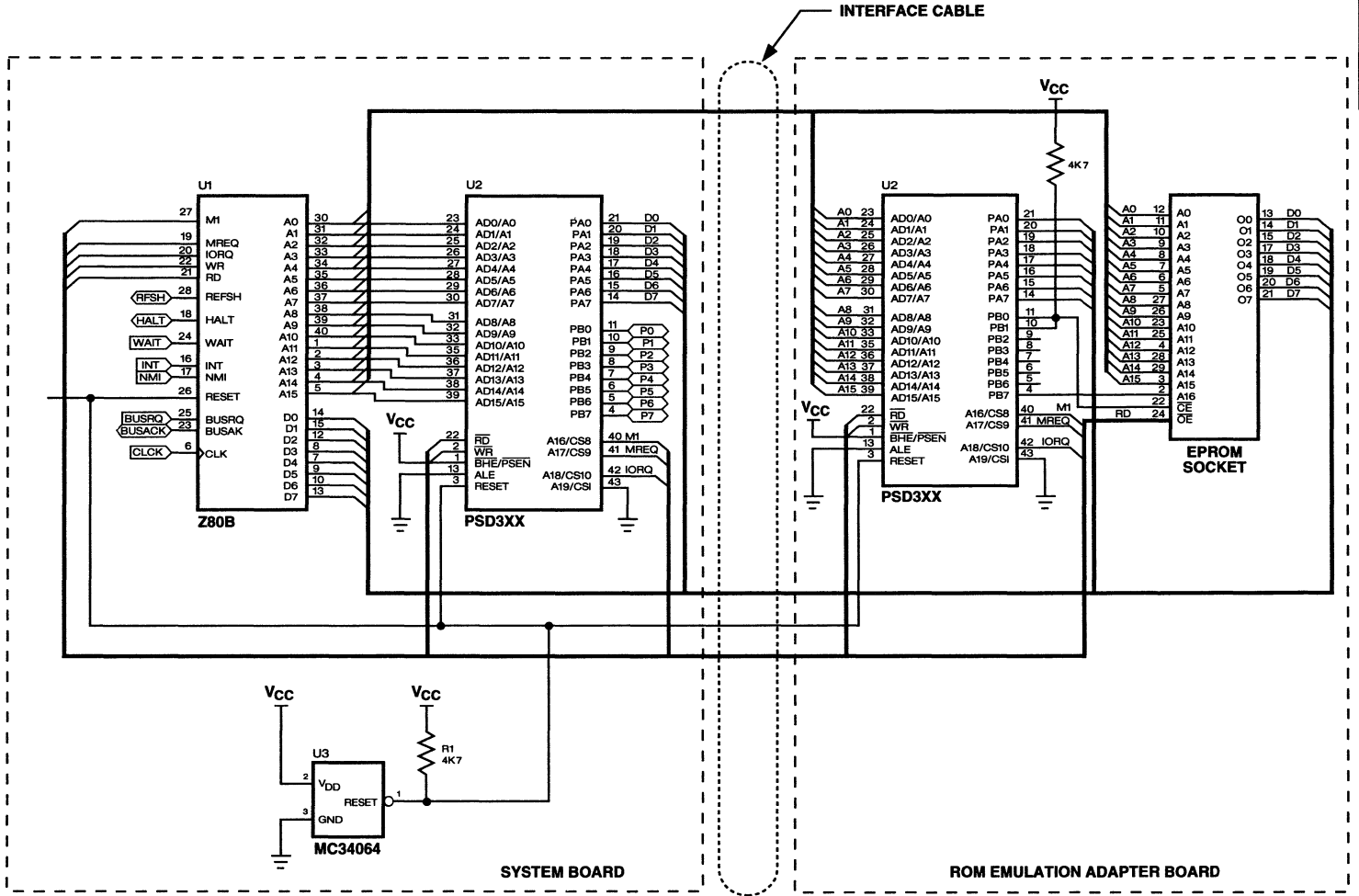


Figure 4. Interface Diagram – 8-Bit Separate Address/Data Bus



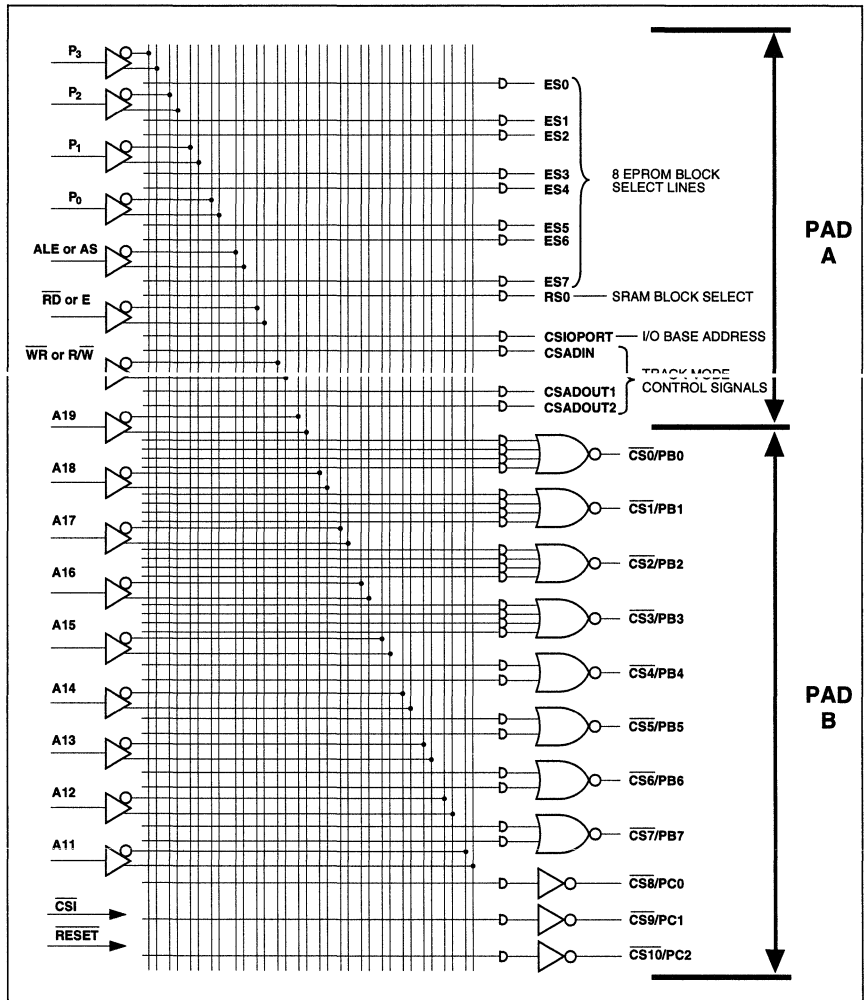
WEST

System PSD3XX Configuration

The Emulator functions by replacing the system's EPROM with SRAM. To accomplish this, when the system PSD3XX is configured by the MAPLE software, ES0 – ES7 are disabled to prevent their eight respective internal EPROM blocks from attempting to access the external Data Bus (Segment Start and Stop Columns in the Address Map are left blank). Reference Figure 5 – PSD3XX PAD Description. Appendix A contains the generalized System PSD3XX Configuration Printout corresponding to the Interface Diagram in Figure 2 and the Memory Map in Figure 6 configured as an example. The Address Map is configured such that all routines are accessible from either Page 1 or Page 2 of main memory.

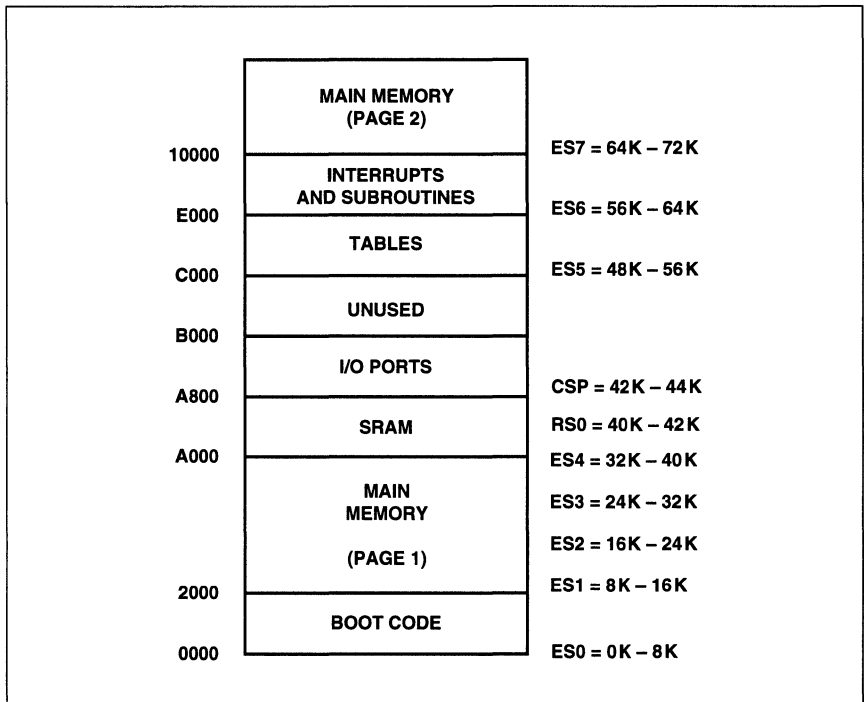
Note: The Address Map in Appendix A shows ES0 – ES7 active to illustrate the block addresses which will be controlled by PB0 – PB1 in the Emulator PSD3XX. ES0 – ES7 are left blank in the actual chip configuration.

**Figure 5.
PSD3XX
PAD
Description**



- NOTES:**
1. \overline{CSi} is a power-down signal. When high, the PAD is in stand-by mode and all its outputs become non-active.
 2. RESET deselects all PAD output signals
 3. A18, A17, and A16 are internally multiplexed with $\overline{CS10}$, $\overline{CS9}$, and $\overline{CS8}$, respectively. Either A18 or $\overline{CS10}$, A17 or $\overline{CS9}$, and A16 or $\overline{CS8}$ can be routed to the external pins of Port C. Port C can be configured as either input or output.

**Figure 6.
Memory Map
System PSD302**



1

Emulator PSD3XX Configuration

The sole function of the Emulator PSD3XX is to access the external program memory whenever the Microcontroller attempts to access the EPROM memory. The addresses of the memory blocks corresponding to ES0 – ES7 of the System PSD3XX internal EPROM Memory Map are programmed into the Emulator PSD3XX, not as ES0 – ES7, but as the eight PAD product terms for PB0 and PB1. These two outputs are configured as Open Drain outputs and wire-or'd together into a common Chip Select to the external EPROM socket. This feature ensures that the Emulator memory is not addressed when the internal SRAM and I/O of the system PSD3XX are accessed. Also, this allows the Emulator SRAM to truly emulate the EPROM of the system PSD3XX by eliminating the need for the program code in the SRAM to be contiguous, by dividing it into eight blocks. Port A is configured to latch the low order address byte for Microcontrollers with Multiplexed Address/Data Buses. PC0 – PC2 can be configured for specific Memory and I/O signals which may be required by the selected Microcontroller. Appendix B contains the generalized Emulator PSD3XX Configuration Printout.

Appendix A.
PSD_SYS.SV1

```
***** MAPLE 5.00 *****
PSD PART USED: PSD302
*****PROJECT INFORMATION*****
Project Name   : = ROM Emulation-System PSD
Your Name     : = Don Buccini
Date          : = June 30, 1993
Host Processor: = 68HC11
*****ALIASES*****
*****GLOBAL CONFIGURATION*****
Address/Data Mode:  MX
Data Bus Size   :    8
Reset Polarity  :    LO
Security        :    OFF
AS Polarity     :    HI
Are you using PSEN ? (Y/N)           :  N
*****

*****READ WRITE CONTROL*****
R/(/W) and E

*****

*****Port A CONFIGURATION*****
ADDRESS/IO

*****

*****PORT A (ADDRESS/IO)*****

Pin    Ai/IO    CMOS/OD
PA0    IO      CMOS
PA1    IO      CMOS
PA2    IO      CMOS
PA3    IO      CMOS
PA4    IO      CMOS
PA5    IO      CMOS
PA6    IO      CMOS
PA7    IO      CMOS

*****
```



Appendix A.
PSD_SYS.SV1
 (Cont.)

*****PORT B CONFIGURATION*****

Pin	CS/IO	CMOS/OD
PB0	IO	CMOS
PB1	IO	CMOS
PB2	IO	CMOS
PB3	IO	CMOS
PB4	IO	CMOS
PB5	IO	CMOS
PB6	IO	CMOS
PB7	IO	CMOS

 *****PORT B CHIP SELECT EQUATIONS*****

*****PORT C CONFIGURATION*****

Pin	CS/Ai	LOGIC/ADDR
PC0	A16	ADDR
PC1	A17	ADDR
PC2	A18	ADDR
A19	CSI	ADDR

 *****PORT C CHIP SELECT EQUATIONS*****

Appendix A.
PSD_SYS.SV1
(Cont.)

*****ADDRESS MAP*****

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	FILE	FILE	File Name	Page	Reg	Q.F
	19	18	17	16	15	14	13	12	11	STRT	STOP	STRT	STOP		3210		AS
ES0	N	0	0	0	0	0	0	N	N	0	1fff	0	1FFF	SYS_PROG.OBJ	0000		N
ES1	N	0	0	0	0	0	1	N	N	2000	3fff	2000	3FFF	SYS_PROG.OBJ	0000		N
ES2	N	0	0	0	0	1	0	N	N	4000	5fff	4000	5FFF	SYS_PROG.OBJ	0000		N
ES3	N	0	0	0	0	1	1	N	N	6000	7fff	6000	7FFF	SYS_PROG.OBJ	0000		N
ES4	N	0	0	0	1	0	0	N	N	8000	9fff	8000	9FFF	SYS_PROG.OBJ	0000		N
ES5	N	0	0	0	1	1	0	N	N	c000	dfff	0	1FFF	SYS_TABL.OBJ	XXXX		N
ES6	N	0	0	0	1	1	1	N	N	e000	ffff	A000	BFFF	SYS_PROG.OBJ	0000		N
ES7	N	0	0	0	0	0	0	N	N	0	1fff	C000	DFFF	SYS_PROG.OBJ	0001		N
RS0	N	0	0	0	1	0	1	0	0	a000	a7ff	N/A	N/A	N/A	XXXX		N
CSP	N	0	0	0	1	0	1	0	1	A800	afff	N/A	N/A	N/A	XXXX		N

*****END*****

*****ADDRESS MAP (EQUATIONS)*****

ES0 = /A18 * /A17 * /A16 * /A15 * /A14 * /A13
 * /P3 * /P2 * /P1 * /P0
 ES1 = /A18 * /A17 * /A16 * /A15 * /A14 * A13
 * /P3 * /P2 * /P1 * /P0
 ES2 = /A18 * /A17 * /A16 * /A15 * A14 * /A13
 * /P3 * /P2 * /P1 * /P0
 ES3 = /A18 * /A17 * /A16 * /A15 * A14 * A13
 * /P3 * /P2 * /P1 * /P0
 ES4 = /A18 * /A17 * /A16 * A15 * /A14 * /A13
 * /P3 * /P2 * /P1 * /P0
 ES5 = /A18 * /A17 * /A16 * A15 * A14 * /A13
 ES6 = /A18 * /A17 * /A16 * A15 * A14 * A13
 * /P3 * /P2 * /P1 * /P0
 ES7 = /A18 * /A17 * /A16 * /A15 * /A14 * /A13
 * /P3 * /P2 * /P1 * P0
 RS0 = /A18 * /A17 * /A16 * A15 * /A14 * A13 * /A12 * /A11
 CSP = /A18 * /A17 * /A16 * A15 * /A14 * A13 * /A12 * A11

*****ADDRESSES OF I/O PORTS*****

Pin Register of Port A: A802 Page (Binary): XXXX
 Direction Register of Port A: A804
 Data Register of Port A: A806
 Pin Register of Port B: A803
 Direction Register of Port B: A805
 Data Register of Port B: A807
 Page Register: A818



**Appendix B.
PSD_SIM.SV1**

```

***** MAPLE 5.00 *****
PSD PART USED: PSD302
*****PROJECT INFORMATION*****
Project Name   : = ROM Emulation-Emulate PSD
Your Name     : = Don Buccini
Date          : = June 30, 1993
Host Processor: = 68HC11
*****ALIASES*****
/CS0  =  ES0 - ES3
/CS1  =  ES4 - ES7
/CS6  =  RD
/CS7  =  A16
*****GLOBAL CONFIGURATION*****
Address/Data Mode:  MX
Data Bus Size   :    8
Reset Polarity  :    LO
Security        :    OFF
AS Polarity     :    HI
Are you using PSEN ? (Y/N)           : N
*****READ WRITE CONTROL*****
R/(/W) and E
*****
*****PORT A CONFIGURATION*****
ADDRESS/IO
*****
*****PORT A (ADDRESS/IO)*****

Pin      Ai/IO      CMOS/OD
PA0      A0        CMOS
PA1      A1        CMOS
PA2      A2        CMOS
PA3      A3        CMOS
PA4      A4        CMOS
PA5      A5        CMOS
PA6      A6        CMOS
PA7      A7        CMOS
*****

```

1



Appendix B.
PSD_SIM.SV1
 (Cont.)

*****PORT B CONFIGURATION*****

Pin	CS/IO	CMOS/OD
PB0	CS0	OD
PB1	CS1	OD
PB2	IO	CMOS
PB3	IO	CMOS
PB4	IO	CMOS
PB5	IO	CMOS
PB6	CS6	CMOS
PB7	CS7	CMOS

 *****PORT B CHIP SELECT EQUATIONS*****

$$\begin{aligned}
 \text{ES0} - \text{ES3} = & / (/A15 * /A14 * /A13 * /P3 * /P2 * /P1 * /P0 \\
 & + /A15 * /A14 * A13 * /P3 * /P2 * /P1 * /P0 \\
 & + /A15 * A14 * /A13 * /P3 * /P2 * /P1 * /P0 \\
 & + /A15 * A14 * A13 * /P3 * /P2 * /P1 * /P0 \\
 &)
 \end{aligned}$$

$$\begin{aligned}
 \text{ES4} - \text{ES7} = & / (A15 * /A14 * /A13 * /P3 * /P2 * /P1 * /P0 \\
 & + A15 * A14 * /A13 \\
 & + A15 * A14 * A13 * /P3 * /P2 * /P1 * /P0 \\
 & + /A15 * /A14 * /A13 * /P3 * /P2 * /P1 * P0 \\
 &)
 \end{aligned}$$

$$\text{RD} = /(\text{E} * \text{R}/\text{W})$$

$$\text{A16} = /(/P3 * /P2 * /P1 * P0)$$

 *****PORT C CONFIGURATION*****

Pin	CS/Ai	LOGIC/ADDR
PC0	A16	LOGIC
PC1	A17	LOGIC
PC2	A18	LOGIC
A19	CSI	

 *****PORT C CHIP SELECT EQUATIONS*****



Appendix B.
PSD_SIM.SV1
(Cont.)

```
*****ADDRESS MAP*****
      A  A  A  A  A  A  A  A  A  A  SEGMENT SEGMENT  FILE  FILE  File Name  Page Reg  Q.F
      19 18 17 16 15 14 13 12 11  STRT  STOP  STRT  STOP
ES0   N                               N  N                               N
ES1   N                               N  N                               N
ES2   N                               N  N                               N
ES3   N                               N  N                               N
ES4   N                               N  N                               N
ES5   N                               N  N                               N
ES6   N                               N  N                               N
ES7   N                               N  N                               N
RS0   N                               N/A  N/A  N/A                               N
CSP   N                               N/A  N/A  N/A                               N
*****END*****
*****ADDRESS MAP (EQUATIONS)*****
*****
```

1

Conclusion

The PSD3XX family was conceived as a cost effective and attractive alternative to discrete logic and memory used in designing an embedded controller system. Although the PSD3XX offers an integrated solution, its flexibility allows the use of basic and inexpensive development tools such as a ROM Emulator. This versatility becomes more important as the need to rapidly develop and debug a product is realized.

Emulation Technology, Inc.

Worldwide Headquarters:
 2344 Walsh Avenue, Bldg. F
 Santa Clara, CA 95051-1301 U.S.A
 Tel: (408) 982-0660
 Fax: (408) 982-0664

Bug Katcher is a trademark of Emulation Technology, Inc.





Programmable Peripheral Application Note 040

Three-Chip Feature Phone

By Steve Torp – Motorola Semiconductor and Karen Spesard – WSI

Introduction

This application note describes how to build a programmable telephone with three off-the-shelf integrated circuits. The feature phone includes a 16 x 2 LCD display and can be programmed to offer many popular functions such as last-number redial, autodial by code number, and hold.

1

Using The Motorola MC34010P, M68HC11D0 and WSI PSD311

The three chips used in the feature phone are the Motorola MC34010 Dialer (DTMF tone generator with speech network and DC line voltage regulator), the Motorola M68HC11D0 microcontroller, and the WSI PSD311 programmable MCU peripheral. Each device is carefully chosen for a specific purpose.

The Motorola MC34010

This device is a single-chip integrated telephone circuit that also supplies a microcontroller interface. It is this interface that enables a simplified connection to the M68HC11D0 and the PSD311.

The Motorola MC68HC11D0

The MC68HC11D0 is an economical microcontroller with low power consumption. The instruction set and memory map are very simple to use making the M68HC11D0 an excellent choice for a low-cost design. The M68HC11D0 also facilitates expansion of the feature phone in order to take advantage of new telephone services as they become available. In this application, the MC68HC11D0 operates in expanded mode. This is important because expanded mode supports interfacing to external SRAM and EPROM devices.

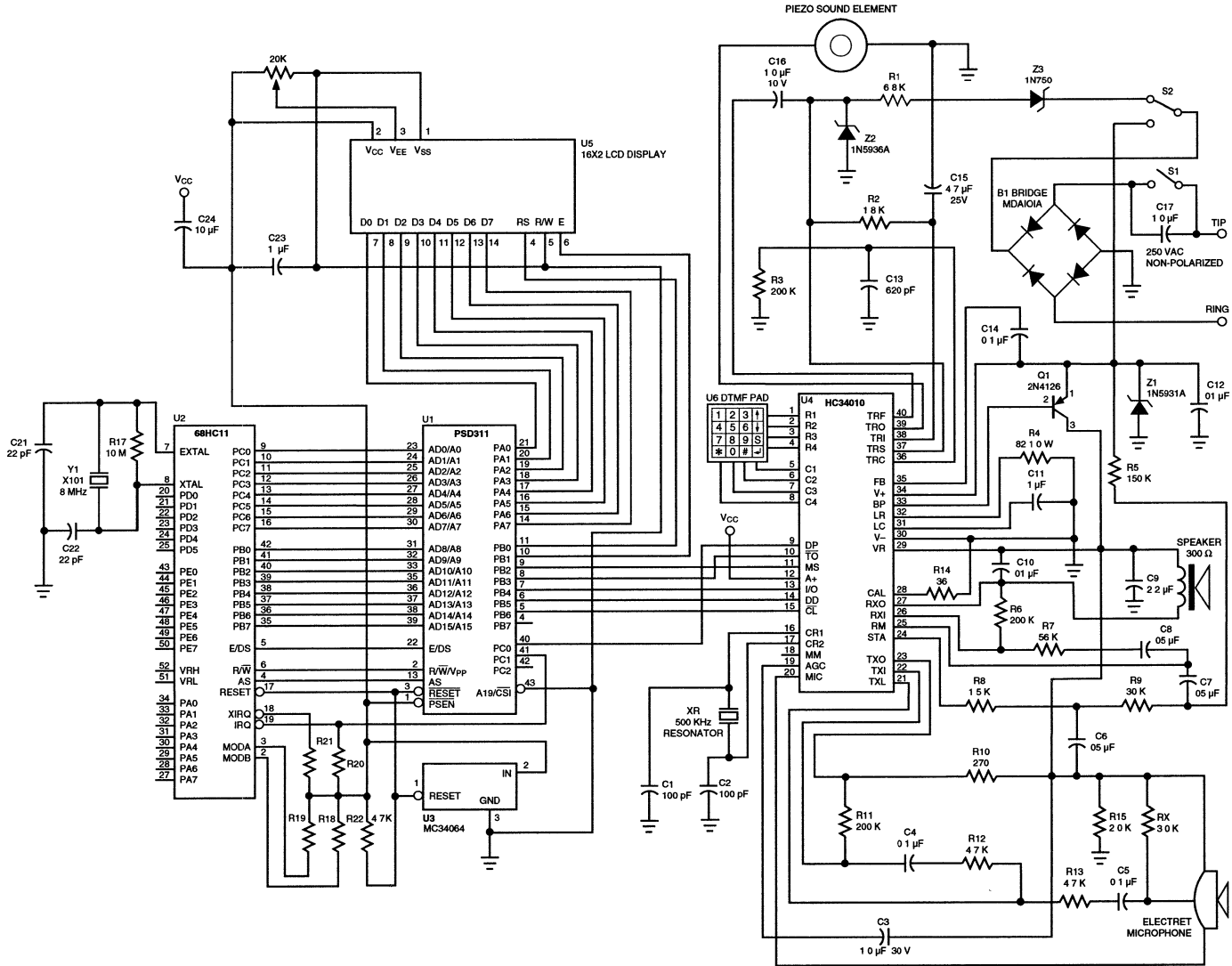
The WSI PSD311

The user-configurable WSI PSD311 Programmable Microcontroller Peripheral is an integral part of this design. The PSD311 eliminates all glue logic and reconstructs the two ports lost by the M68HC11D0 when it is in expanded mode. The PSD311 also incorporates 32K bytes of EPROM, 2K bytes of SRAM, and preserves low power operation. With this programmable device, the feature phone is capable of retaining the last 256 phone numbers that were dialed, making the automatic recall of numbers effortless.

Since the M68HC11D0 does not contain a ROM, the PSD311's EPROM will contain the main program that will service the entire feature phone. The PSD311 will also use its port pins to send data and control information to the Philips 16 x 2 LCD module. The PSD311 is an excellent comprehensive choice that enables this feature phone to be comprised of only three chips.

The design schematic is shown in Figure 1. Note how simple the design becomes with these readily available components.

Motorola/WSI Feature Phone Schematic



Interconnecting The Parts

The MC34010 has six pins that are used as the digital interface. They are DP, DD, \overline{TO} , MS, \overline{CL} , and I/O. Pin 12 (A+) allows the digital interface to be active and must have a voltage, typically connected to pin 34 (V+). The DP pin, which is inverted from active HIGH to active LOW (by routing through some PSD311 programmable logic), is connected to the microcontroller \overline{IRQ} pin. Thus, when a key is depressed on the keypad, the \overline{IRQ} pin is asserted on the M68HC11D0 microcontroller. Upon assertion of \overline{IRQ} , the M68HC11D0 fetches the interrupt vector and begins to execute interrupt exception code. This code looks for the DD pin to be low. With the help of the \overline{CL} input (the clock input), a 4-bit serial data stream is transmitted into the PSD311 from the I/O pin and is then translated into one of the 16 keys on the keypad. A DTMF tone is generated (\overline{TO} is driven low) and the number is saved in the PSD311's SRAM and subsequently sent to the LCD display. The last output on the MC34010 is the MS output that serves to enable or disable the tone generator.

1

Configuring The PSD

The user-configurability feature of the WSI PSD311 makes implementing the interface between the M68HC11D0 and the MC34010 very simple. The PSD311 can be programmed to connect to the M68HC11D0 on one side and interfaced to virtually any peripheral like the MC34010 on the other. To achieve this flexibility, the PSD contains non-volatile configuration bits that are chosen by the user when using WSI's PSD MAPLE software and are set in the device during programming.

To achieve a direct hardware interface to the M68HC11H0, the PSD311 options programmed for this application include: active high AS, active low reset, and R/ \overline{W} and E mode for the control signals.

The LCD interface incorporates eight bidirectional I/O lines that are connected to Port A on the PSD311. Port A is configured as a general-purpose I/O to provide the eight bits of data to the LCD display. In addition, for other applications, Port A is capable of transferring up to eight low order address bits. The functionality of Port B on the PSD311 is split. While all the pins are configured as general-purpose I/Os and not chip select or logic outputs, two pins (PB0 and PB1) enable the LCD display and RS line. Another five I/O pins on Port B map directly to the MC34010's microprocessor interface lines: DD, \overline{TO} , MS, \overline{CL} , and I/O. Finally, two pins on Port C, one used as an input to the internal programmable logic array, and the other used as a chip select/logic output from the array, are used to invert the active HIGH DP line from the MC34010 to active LOW so the microcontroller \overline{IRQ} will recognize a depressed key.

The PSD311 also specifies the address map for the system. For example, both the LCD and the MC34010 peripherals start at location h'4000, as shown in the software listing. The rest of the address map encoded in the PSD311 includes 32K bytes of EPROM beginning at location h'2000, and the SRAM at location h'5000. A summary of the PSD configurations is shown in the listing file in Appendix A.

The System Software

The M68HC11 software is written in assembly code for this application and appears in Appendix B. It is, for the most part, self-explanatory and is well commented.

Appendix A.
PSD311
Listing File For
Feature Phone
Application

```
***** MAPLE 6.21 *****
                                ALIASES
/CS8/A16 = INTR
/CS9/A17 = BINTR
*****

                                GLOBAL CONFIGURATION
Address/Data Mode:                MX
Data Bus Size:                    8
CSI/A19:                          CSI
Reset Polarity:                   LO
ALE Polarity:                     HI
WRD/RWE:                          RWE
A16-A19 Transparent or Latched by ALE: T
Are you using PSEN? N
*****

                                PORT A CONFIGURATION (Address/IO)
Bit No.  Ai/IO.  CMOS/OD.
  0       IO     CMOS
  1       IO     CMOS
  2       IO     CMOS
  3       IO     CMOS
  4       IO     CMOS
  5       IO     CMOS
  6       IO     CMOS
  7       IO     CMOS
*****

                                PORT B CONFIGURATION
Bit No.  CS/IO.  CMOS/OD.
  0       IO     CMOS
  1       IO     CMOS
  2       IO     CMOS
  3       IO     CMOS
  4       IO     CMOS
  5       IO     CMOS
  6       IO     CMOS
  7       IO     CMOS
*****

                                CHIP SELECT EQUATIONS
*****

                                PORT C CONFIGURATION
Bit No.  CS/Ai.
  0       A16
  1       CS9
  2       CS10
*****

                                CHIP SELECT EQUATIONS
BINTR = /(INTR)
*****
```



Appendix A.
PSD311
Listing File For
Feature Phone
Application
(Cont.)

ADDRESS MAP

	A	A	A	A	A	A	A	A	SEGMENT	SEGMENT	EPROM	EPROM	File Name	
	19	18	17	16	15	14	13	12	STRT	STOP	START	STOP		
ES0	N	N	N	X	0	0	1	0	N	2000	2FFF	2000	2fff	FPHONE.HEX
ES1	N	N	N	X	0	0	1	1	N	3000	3FFF	3000	3fff	FPHONE.HEX
ES2	N	N	N	X	1	1	1	1	N	F000	FFFF	f000	ffff	FPHONE.HEX
ES3	N	N	N						N					
ES4	N	N	N						N					
ES5	N	N	N						N					
ES6	N	N	N						N					
ES7	N	N	N						N					
RS0	N	N	N	X	0	1	0	1	0	5000	57FF			
CSP	N	N	N	X	0	1	0	0	0	4000	47FF			

1

***** ADDRESS MAP (EQUATIONS) *****

ES0 = /A15 * /A14 * A13 * /A12
 ES1 = /A15 * /A14 * A13 * A12
 ES2 = A15 * A14 * A13 * A12
 ES3 =
 ES4 =
 ES5 =
 ES6 =
 ES7 =
 RS0 = /A15 * A14 * /A13 * A12 * /A11
 CSP = /A15 * A14 * /A13 * /A12 * /A11

***** END *****

**Appendix B.
Feature Phone
Software Listing**

```

0001 *****
0002 * Feature Phone Software for use with 68HC11 and PSD *
0003 * By Karen Spesard and Steve Torp - 1/11/95 *
0004 *****
0005
0006 2000 ORG $2000 PROGRAM MEMORY
0007
0008 003d INIT: EQU $003D RAM AND I/O MAPPING REGISTER
0009 4000 PORTAB: EQU $4000 I/O BASE ADDRESS OF THE PSD311
0010 000f KEY1: EQU $0F KEYPAD 1
0011 0007 KEY2: EQU $07 KEYPAD 2 (ABC)
0012 000b KEY3: EQU $0B KEYPAD 3 (DEF)
0013 0003 KEY4: EQU $03 KEYPAD MODE (NORMAL/STORE/RECALL)
0014 000d KEY4: EQU $0D KEYPAD 4 (GHI)
0015 0005 KEY5: EQU $05 KEYPAD 5 (JKL)
0016 0009 KEY6: EQU $09 KEYPAD 6 (MNO)
0017 0001 KEYB: EQU $01 KEYPAD SEND
0018 000e KEY7: EQU $0E KEYPAD 7 (PQRS)
0019 0006 KEY8: EQU $06 KEYPAD 8 (TUV)
0020 000a KEY9: EQU $0A KEYPAD 9 (WXYZ)
0021 0002 KEYC: EQU $02 KEYPAD UP
0022 000c KEYS: EQU $0C KEYPAD * (STOP/ERASE)
0023 0004 KEY0: EQU $04 KEYPAD 0
0024 0008 KEYN: EQU $08 KEYPAD # (ENTER)
0025 0000 KEYD: EQU $00 KEYPAD DOWN
0026
0027 2000 of START: SEI SET INTERRUPT MASK IN CCR REG FOR INIT
0028
0029 *64 Bytes of Register Area
0030
0031 0000 PORTA: EQU $0000 PORT A DATA REGISTER
0032 0002 PIOC: EQU $0002 PARALLEL I/O CTL REGISTER
0033 0003 PORTC: EQU $0003 PORT C DATA REGISTER (AD0-AD7)
0034 0004 PORTB: EQU $0004 PORT B DATA REGISTER (A8-A15)
0035 0007 DDRC: EQU $0007 PORT C DATA DIRECTION REGISTER
0036 0008 PORTD: EQU $0008 PORT D DATA REGISTER (RxD, TxD, AND I/O)
0037 0009 DDRD: EQU $0009 PORT D DATA DIRECTION REGISTER
0038 000b CFORC: EQU $000B TIMER COMPARE FORCE REGISTER
0039 000c OC1M: EQU $000C OUTPUT COMPARE 1 MASK REGISTER
0040 000d OC1D: EQU $000D OUTPUT COMPARE 1 DATA REGISTER
0041 000e TCNT: EQU $000E TIMER COUNTER REGISTER (16-BIT) $000F LSB
0042 0010 TIC1: EQU $0010 TIMER INPUT CAPTURE REGISTER 1 (16-BIT)
0043 0012 TIC2: EQU $0012 TIMER INPUT CAPTURE REGISTER 2 (16-BIT)
0044 0014 TIC3: EQU $0014 TIMER INPUT CAPTURE REGISTER 3 (16-BIT)
0045 0016 TOC1: EQU $0016 TIMER OUTPUT COMPARE REGISTER 1 (16-BIT)
0046 0018 TOC2: EQU $0018 TIMER OUTPUT COMPARE REGISTER 2 (16-BIT)
0047 001a TOC3: EQU $001A TIMER OUTPUT COMPARE REGISTER 3 (16-BIT)
0048 001c TOC4: EQU $001C TIMER OUTPUT COMPARE REGISTER 4 (16-BIT)
0049 001e TOC5: EQU $001E TIMER OUTPUT COMPARE REGISTER 5/INPUT
0050 0020 TCTL1: EQU $0020 TIMER CONTROL REGISTER 1
0051 0021 TCTL2: EQU $0021 TIMER CONTROL REGISTER 2
0052 0022 TMASK1: EQU $0022 MAIN TIMER INTERRUPT MASK REGISTER 1
0053 0023 TFLG1: EQU $0023 MAIN TIMER INTERRUPT FLAG REGISTER 1
0054 0024 TMASK2: EQU $0024 MAIN TIMER INTERRUPT MASK REGISTER 2
0055 0025 TFLG2: EQU $0025 MAIN TIMER INTERRUPT FLAG REGISTER 2
0056 0026 PACTL: EQU $0026 PULSE ACCUMULATOR CONTROL REGISTER
0057 0027 PACNT: EQU $0027 PULSE ACCUMULATOR COUNT REGISTER
0058 0028 SPCR: EQU $0028 SPI CONTROL REGISTER
0059 0029 SPSR: EQU $0029 SPI STATUS REGISTER
0060 002a SPDR: EQU $002A SPI DATA REGISTER
0061 002b BAUD: EQU $002B SCI BAUD RATE CONTROL REGISTER
0062 002c SCCR1: EQU $002C SCI CONTROL REGISTER 1
0063 002d SCCR2: EQU $002D SCI CONTROL REGISTER 2

```



Appendix B. Feature Phone Software Listing (Cont.)

0064 002e	SCSR:	EQU	\$002E	SCI STATUS REGISTER
0065 002f	SCDR:	EQU	\$002F	SCI DATA REGISTER
0066 0039	OPTION:	EQU	\$0039	SYSTEM CONFIGURATION OPTIONS
0067 003a	COPRST:	EQU	\$003A	ARM/RESET COP TIMER CIRCUITRY
0068 003b	PPROG:	EQU	\$003B	EEPROM PROGRAMMING REGISTER
0069 003c	HPRIO:	EQU	\$003C	HIGHEST PRIORITY INTERRUPT
0070 003e	TEST1:	EQU	\$003E	FACTORY TEST REGISTER
0071 003f	CONFIG:	EQU	\$003F	CONFIGURATION CONTROL REGISTER
0072				
0073	*192 Bytes of Internal RAM			
0074				
0075 0040	FLAGS:	EQU	\$0040	FLAG REGISTER
0076 0041	N:	EQU	\$0041	DIGIT POINTER FOR PHONE NUMBER
0077 0042	M:	EQU	\$0042	NUMBER OF DIGITS IN PHONE NUMBER
0078 0043	P:	EQU	\$0043	STORED DIGIT WHEN READ
0079 0044	P12:	EQU	\$0044	1ST TWO STORED DIGITS
0080 0045	P34:	EQU	\$0045	2ND TWO STORED DIGITS
0081 0046	P56:	EQU	\$0046	3RD TWO STORED DIGITS
0082 0047	P78:	EQU	\$0047	4TH TWO STORED DIGITS
0083 0048	P910:	EQU	\$0048	5TH TWO STORED DIGITS
0084 0049	P1112:	EQU	\$0049	6TH TWO STORED DIGITS
0085 00ff	STPTR:	EQU	\$00FF	STACK POINTER AREA
0086 004a	SWRPTR:	EQU	\$004A	MASS STORAGE RAM WRITE POINTER
0087 004b	SRDPTR:	EQU	\$004B	MASS STORAGE RAM READ POINTER
0088				
0089	*2K x 8 RAM in PSD311			
0090				
0091 5000	MASSTOR:	EQU	\$5000	START OF MASS STORAGE BUFFER RAM IN
PSD311				
0092				
0093	*Other register initialization			
0094				
0095 2001 86 20	LDAA	#\$20		SET UP OPTION REGISTER/IRQE=0
0096 2003 97 39	STAA	OPTION		
0097 2005 86 04	LDAA	#\$04		
0098 2007 97 3f	STAA	CONFIG		
0099 2009 8e 00 ff	LDS	#STPTR		SET UP STACK
0100 200c b6 50 00	LDAA	MASSTOR		SET UP SRAM WRITE POINTERS
0101 200f 97 4a	STAA	SWRPTR		AND SRAM READ POINTERS
0102 2011 97 4b	STAA	SRDPTR		
0103				
0104	*PSD Port Direction Set Up			
0105				
0106 2013 ce 40 00	LDX	#PORTAB		PORTS A&B OF PSD
0107 2016 86 ff	LDAA	#\$FF		SET ALL PORT A PINS AS OUTPUTS
0108 2018 a7 04	STAA	4,X		AND STORE
0109 201a 86 eb	LDAA	#\$EB		SET ALL PORT B PINS AS OUTPUTS BUT
0110 201c a7 05	STAA	5,X		PINS PB4 (I/O) AND PB2 (MS) FOR NOW
0111				
0112	*Set Up Values for PSD Port Data Outputs Interfacing to MC34010			
0113				
0114 201e ce 40 00	LDX	#PORTAB		PORTS A&B OF PSD
0115 2021 86 00	LDAA	#\$00		SET PORT B PINS 3,5,6 AS LOW OUTPUTS
0116 2023 a7 07	STAA	7,X		
0117				

**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0118		*Display set up		
0119				
0120 2025 ce 27 10	DISINIT:	LDX	#\$2710	100ms DELAY (PWR UP DELAY FOR DISPLAY)
				TIME DELAY
0121 2028 bd 30 a4		JSR	TDELAY	TIME DELAY
0122 202b 86 38		LDAA	#\$038	SET UP DISPLAY
0123 202d bd 30 b7		JSR	SENDI	SEND INSTRUCTION (30 1ST TIME)
0124 2030 ce 03 00		LDX	#\$300	6.1ms DELAY
0125 2033 bd 30 a4		JSR	TDELAY	TIME DELAY
0126 2036 bd 30 b7		JSR	SENDI	SEND INSTRUCTION (30 2ND TIME)
0127 2039 bd 30 a1		JSR	TD150	TIME DELAY
0128 203c bd 30 b7		JSR	SENDI	SEND INSTRUCTION (30 3RD TIME)
0129 203f bd 30 a1		JSR	TD150	TIME DELAY
0130 2042 86 38		LDAA	#\$038	FUNCTION SET (8-BIT:2-LINE)
0131 2044 bd 30 b7		JSR	SEND	SEND INSTRUCTION
0132 2047 ce 02 80		LDX	#\$280	5ms DELAY
0133 204a bd 30 a4		JSR	TDELAY	TIME DELAY
0134 204d 86 0e		LDAA	#\$0E	DISPLAY ON - CURSOR ON
0135 204f bd 30 b7		JSR	SENDI	SEND INSTRUCTION
0136 2052 ce 02 80		LDX	#\$280	5ms DELAY
0137 2055 bd 30 a4		JSR	TDELAY	TIME DELAY
0138 2058 86 06		LDAA	#\$06	ENTRY MODE SET
0139 205a bd 30 b7		JSR	SENDI	SEND INSTRUCTION
0140 205d ce 02 80		LDX	#\$280	5ms DELAY
0141 2060 bd 30 a4		JSR	TDELAY	TIME DELAY
0142 2063 bd 30 ab		JSR	CSCREEN	CLEAR SCREEN
0143 2066 ce 02 80		LDX	#\$280	5ms DELAY
0144 2069 bd 30 a4		JSR	TDELAY	TIME DELAY
0145 206c bd 30 af		JSR	HOME	DISPLAY CURSOR HOME!
0146 206f ce 01 90		LDX	#\$190	4ms DELAY
0147 2072 bd 30 a4		JSR	TDELAY	TIME DELAY
0148				
0149		*Final Initialization		
0150				
0151 2075 86 07	REGINIT:	LDAA	#\$07	INITIALIZE REGISTER "M"
0152 2077 97 42		STAA	M	STORE# OF DIGITS IN TEL NO.
0153 2079 86 00		LDAA	#\$00	CLEAR FOLLOWING "REGISTERS"
0154 207b 97 41		STAA	N	
0155 207d 97 43		STAA	P	
0156 207f 97 44		STAA	P12	STORAGE FOR EACH DIGIT OF
0157 2081 97 45		STAA	P34	PHONE NUMBER . . .
0158 2083 97 46		STAA	P56	
0159 2085 97 47		STAA	P78	
0160 2087 97 48		STAA	P910	
0161 2089 97 49		STAA	P1112	
0162				
0163 208b 18 ce 30 1e	DISPLAY:	LDY	#\$301E	NORMAL MODE
0164 208f bd 30 a1		JSR	TD150	TIME DELAY
0165 2092 bd 30 b3		JSR	LINE2	USE LINE 2
0166 2095 bd 30 8a		JSR	PDOD	DISPLAY MODE
0167 2098 bd 30 a1		JSR	TD150	TIME DELAY
0168 209b bd 30 af		JSR	HOME	USE LINE 1 NEXT
0169 209e 0e		CLI		CLEAR IRQ MASK
0170 209f 86 00		LDAA	#\$00	LOAD A WITH STOP ENABLE FOR CCR
0171 20a1 06		TAP		TRANSFER A ACCUM TO CCR
0172 20a2 01		NOP		
0173 20a3 01		NOP		
0174 20a4 cf	STOP1:	STOP		

Appendix B. Feature Phone Software Listing (Cont.)

```

0175
0176          *IRQ Service Routine
0177
0178          *Read Which Key Depressed, Store in Accumulator A
0179
0180 20a5 ce 40 00      GDATA:  LDX  #PORTAB  SET UP PSD PORTB TO READ KEY
0181 20a8 86 eb          LDAA  #$EB    SET ALL PORT B PINS AS OUTPUTS EXCEPT
0182 20aa a7 05          STAA  5,X    PINS PB4 (I/O) AND PB2 (DP) FOR NOW
0183 20ac 86 00          LDAA  #$00    INITIALIZE PORT B OUTPUT PINS 3,5,6 LOW
0184 20ae a7 07          STAA  7,X
0185
0186 20b0 bd 30 97      RDATA:  JSR  T20    TDELAY 20uS
0187 20b3 ce 40 00      LDX  #PORTAB  SET UP PSD PORTB TO READ KEY
0188 20b6 1d 07 40      BCLR  7,X $40  ETC /CL LOW
0189 20b9 bd 30 97      JSR  T20    TDELAY 20uS
0190 20bc 1c 07 40      BSET  7,X $40  ETC /CL HIGH
0191 20bf bd 30 97      JSR  T20    TDELAY 20uS
0192 20c2 1d 07 40      BCLR  7,X $40  ETC /CL LOW
0193 20c5 bd 30 97      JSR  T20    TDELAY 20uS
0194 20c8 1c 07 40      BSET  7,X $40  ETC /CL HIGH
0195 20cb bd 30 97      JSR  T20    TDELAY 20uS
0196
0197 20ce 4f            CLRA          CLEAR ACCUMULATOR A
0198 20cf 18 ce 00 04    LDY  #$04    READ 4 I/O-DATA BITS
0199 20d3 0c            READD:  CLC          CLEAR CARRY BIT
0200 20d4 1d 07 40      BCLR  7,X $40  ETC /CL LOW
0201 20d7 1f 07 10 01  BRCLR  7,X $10 N1  BRANCH IF I/O LOW TO CLEAR CARRY,
0202 20db 0d            SEC          OTHERWISE, SET CARRY
0203 20dc 49            N1:    ROLA          RD EACH BIT TO DETERMINE KEY PRESSED
0204 20dd bd 30 97      JSR  T20    TDELAY 20uS
0205 20e0 1c 07 40      BSET  7,X $40  ETC /CL HIGH
0206 20e3 bd 30 97      JSR  T20    TDELAY 20uS
0207 20e6 18 09          DEY          DECREMENT COUNT
0208 20e8 18 8c 00 00    CPY  #$00    COUNT = 0?
0209 20ec 26 e5          BNE  READD   IF NOT DONE, GO TO READD
0210 20ee 97 43          STAA  P      STORE DIALED NUMBER IN P REGISTER
0211
0212          *Start Routine
0213
0214 20f0 81 0f            START1:  CMPA  #KEY1    1 KEY?
0215 20f2 26 0d            BNE  DOIT10  IF NOT GOTO DOIT10
0216
0217 20f4 18 ce 30 00      LDY  #$3000  DISPLAY "1"
0218 20f8 bd 30 8a          JSR  PDOD    SEND MESSAGE TO DISPLAY
0219 20fb bd 30 d8          JSR  CHD     CHK IF 1ST DIGIT & SAVE
0220
0221 20fe 7e 22 1b        RETRY:  JMP  ENDKEYDP  RTI AFTER KEY IS NO LONGER DEPRESSED
0222

```

**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0223 2101 81 07	DOIT10:	CMPA	#KEY2	2 KEY?
0224 2103 26 0d		BNE	DOIT20	IF NOT GOTO DOIT20
0225 2105 18 ce 30 03		LDY	#\$3003	DISPLAY "2"
0226 2109 bd 30 8a		JSR	PDOD	
0227 210c bd 30 d8		JSR	CHD	SAVE DIGIT
0228 210f 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0229				
0230 2112 81 0b	DOIT20:	CMPA	#KEY3	3 KEY?
0231 2114 26 09		BNE	DOIT30	IF NOT GOTO DOIT30
0232 2116 bd 30 8a		JSR	PDOD	
0233 2119 bd 30 d8		JSR	CHD	SAVE DIGIT
0234 211c 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0235				
0236 211f 81 03	DOIT30:	CMPA	#KEYA	A KEY? (MODE)
0237 2121 26 13		BNE	DOIT40	IF NOT GOTO DOIT40
0238 2123 18 ce 30 30		LDY	#\$3030	NORMAL/RECALL/STORE
0239 2127 bd 30 a1		JSR	TD150	
0240 212a bd 30 b3		JSR	LINE2	(THIS ROUTINE NOT COMPLETE)
0241 212d bd 30 a1		JSR	TD150	
0242 2130 bd 30 8a		JSR	PDOD	
0243 2133 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0244				
0245 2136 81 0d	DOIT40:	CMPA	#KEY4	4 KEY?
0246 2138 26 0d		BNE	DOIT50	IF NOT GOTO DOIT50
0247 213a 18 ce 30 09		LDY	#\$3009	DISPLAY "4"
0248 213e bd 30 8a		JSR	PDOD	
0249 2141 bd 30 d8		JSR	CHD	SAVE DIGIT
0250 2144 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0251				
0252 2147 81 05	DOIT50:	CMPA	#KEY5	5 KEY?
0253 2149 26 10		BNE	DOIT60	IF NOT GOTO DOIT60
0254 214b 18 ce 30 0c		LDY	#\$300C	
0255 214f bd 30 8a		JSR	PDOD	DISPLAY "5"
0256 2152 bd 30 d8		JSR	CHD	SAVE DIGIT
0257 2155 bd 31 9d		JSR	CHAD	IF ALL DIGITS PRESSED IN NO., STORE
0258 2158 7e 22 1b		JMP	ENDKEYDP	
0259				
0260 215b 81 09	DOIT60:	CMPA	#KEY6	6 KEY?
0261 215d 26 0d		BNE	DOIT70	IF NOT GOTO DOIT70
0262 215f 18 ce 30 0f		LDY	#\$300F	
0263 2163 bd 30 8a		JSR	PDOD	DISPLAY "6"
0264 2166 bd 30 d8		JSR	CHD	SAVE DIGIT
0265 2169 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0266				
0267 216c 81 01	DOIT70:	CMPA	#KEYB	B KEY? (SEND)
0268 216e 26 10		BNE	DOIT80	IF NOT GOTO DOIT80
0269 2170 18 ce 30 54		LDY	#\$3054	
0270 2174 bd 30 b3		JSR	LINE2	
0271 2177 bd 30 8a		JSR	PDOD	DISPLAY "SEND"
0272 217a bd 31 ea		JSR	DIALNO	SEND NO TO DTMF AND DIAL
0273 217d 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0274				
0275				
0276 2180 81 0e	DOIT80:	CMPA	#KEY7	7 KEY?
0277 2182 26 0d		BNE	DOIT90	IF NOT GOTO DOIT90
0278 2184 18 ce 30 12		LDY	#\$3012	
0279 2188 bd 30 8a		JSR	PDOD	DISPLAY "7"
0280 218b bd 30 d8		JSR	CHD	SAVE DIGIT
0281 218e 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0282				

Appendix B. Feature Phone Software Listing (Cont.)

0283 2191 81 06	DOIT90:	CMPA	#KEY8	8 KEY?
0284 2193 26 0d		BNE	DOIT100	IF NOT GOTO DOIT100
0285 2195 18 ce 30 15		LDY	#\$3015	
0286 2199 bd 30 8a		JSR	PDOD	DISPLAY "8"
0287 219c bd 30 d8		JSR	CHD	SAVE DIGIT
0288 219f 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0289				
0290 21a2 81 0a	DOIT100:	CMPA	#KEY9	9 KEY?
0291 21a4 26 0d		BNE	DOIT110	IF NOT GOTO DOIT110
0292 21a6 18 ce 30 18		LDY	#\$3018	
0293 21aa bd 30 8a		JSR	PDOD	DISPLAY "9"
0294 21ad bd 30 d8		JSR	CHD	SAVE DIGIT
0295 21b0 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0296				
0297 21b3 81 02	DOIT110:	CMPA	#KEYC	C KEY? (UP)
0298 21b5 26 22		BNE	DOIT120	IF NOT GOTO DOIT120
0299 21b7 bd 30 ab		JSR	CSCREEN	
0300 21ba 18 ce 30 66		LDY	#\$3066	
0301 21be bd 30 a1		JSR	TD150	
0302 21c1 bd 30 b3		JSR	LINE2	
0303 21c4 bd 30 8a		JSR	PDOD	DISPLAY "SCROLL UP"
0304 21c7 bd 30 a1		JSR	TD150	
0305 21ca bd 30 af		JSR	HOME	
0306 21cd 86 08		LDAA	#\$08	
0307 21cf 90 4b		SUBA	SRDPTR	
0308 21d1 97 4b		STAA	SRDPTR	
0309 21d3 bd 31 c7		JSR	SCROLL	SEND NUMBER TO DISPLAY
0310 21d6 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0311				
0312 21d9 81 0c	DOIT120:	CMPA	#KEYS	* KEY?
0313 21db 26 03		BNE	DOIT130	IF NOT GOTO DOIT130
0314 21dd 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0315				
0316 21e0 81 04	DOIT130:	CMPA	#KEY0	0 KEY?
0317 21e2 26 0d		BNE	DOIT140	IF NOT GOTO DOIT140
0318 21e4 18 ce 30 1b		LDY	#\$301B	
0319 21e8 bd 30 8a		JSR	PDOD	DISPLAY "0"
0320 21eb bd 30 d8		JSR	CHD	SAVE DIGIT
0321 21ee 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0322				
0323 21f1 81 08	DOIT140:	CMPA	#KEYN	# KEY?
0324 21f3 26 03		BNE		DOIT150
IF NOT GOTO DOIT150				
0325 21f5 7e 22 1b		JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0326				
0327 21f8 81 00	DOIT150:	CMPA	#KEYD	D KEY? (DOWN)
0328 21fa 26 1c		BNE		DOIT160
IF NOT GOTO DOIT160				
0329 21fc 18 ce 30 78		LDY	#\$3078	
0330 2200 bd 30 a1		JSR	TD150	
0331 2203 bd 30 b3		JSR	LINE2	
0332 2206 bd 30 8a		JSR	PDOD	DISPLAY "SCROLL DOWN"
0333 2209 bd 30 a1		JSR	TD150	
0334 220c bd 30 af		JSR	HOME	
0335 220f 86 08		LDAA	#\$08	
0336 2211 9b 4b		ADDA	SRDPTR	
0337 2213 97 4b		STAA	SRDPTR	
0338 2215 bd 31 c7		JSR	SCROLL	
0339				
0340 2218 7e 22 1b	DOIT160:	JMP	ENDKEYDP	RTI AFTER KEY IS NO LONGER DP
0341				
0342				

**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0343				*Check if key no longer depressed
0344				
0345 221b ce 40 00	ENDKEYDP:	LDX	#PORTAB	CHECK FOR MS - PORTB PIN 2 (DP) LOW
0346 221e a6 03	REPEAT:	LDA	3,X	WHICH MEANS KEY IS NO LONGER DEPRESSED
0347 2220 84 04		ANDA	#\$04	
0348 2222 26 fa			BNE	REPEAT
0349 2224 0e			CLI	CLEAR INTERRUPT
0350 2225 7e 20 a4		JMP	STOP1	WAIT FOR NEXT KEY TO BE DEPRESSED
0351				
0352				*Screens
0353				
0354 3000		ORG	\$3000	
0355				
0356 3000 31	SCR1:	FCC	"1"	
0357 3001 00 00		FDB	\$00	
0358 3003 32	SCR2:	FCC	"2"	
0359 3004 00 00		FDB	\$00	
0360 3006 33	SCR3:	FCC	"3"	
0361 3007 00 00		FDB	\$00	
0362 3009 34	SCR4:	FCC	"4"	
0363 300a 00 00		FDB	\$00	
0364 300c 35	SCR5:	FCC	"5"	
0365 300d 00 00		FDB	\$00	
0366 300f 36	SCR6:	FCC	"6"	
0367 3010 00 00		FDB	\$00	
0368 3012 37	SCR7:	FCC	"7"	
0369 3013 00 00		FDB	\$00	
0370 3015 38	SCR8:	FCC	"8"	
0371 3016 00 00		FDB	\$00	
0372 3018 39	SCR9:	FCC	"9"	
0373 3019 00 00		FDB	\$00	
0374 301b 30	SCR0:	FCC	"0"	
0375 301c 00 00		FDB	\$00	
0376 301e 20 20 4e 4f 52 4d 41 4c 20 4d 4f 44 45 20 20 20	SCRA1:	FCC	" NORMAL MODE "	
0377 302e 00 00		FDB	\$00	
0378 3030 20 20 52 45 43 41 4c 4c 20 4d 4f 44 45 20 20 20	SCRA2:	FCC	" RECALL MODE "	
0379 3040 00 00		FDB	\$00	
0380 3042 20 20 20 53 54 4f 52 45 20 4d 4f 44 45 20 20 20	SCRA3:	FCC	" STORE MO "	
0381 3052 00 00		FDB	\$00	
0382 3054 20 20 20 20 53 45 4e 44 20 20 20 20 20 20 20	SCRB:	FCC	" SEND "	
0383 3064 00 00		FDB	\$00	
0384 3066 20 20 20 53 43 52 4f 4c 4c 20 55 50 20 20 20 20	SCRc:	FCC	" SCROLL UP "	
0385 3076 00 00		FDB	\$00	
0386 3078 20 20 53 43 52 4f 4c 4c 20 44 4f 57 4e 20 20 20	SCRd:	FCC	" SCROLL DOWN "	
0387 3088 00 00		FDB	\$00	
0388				
0389				

**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0390	*Subroutines		
0391			
0392	*Put Data on Display		
0393			
0394 308a 18 a6 00	PDOD:	LDA 0,Y	GET BYTE
0395 308d 27 07		BEQ PDOD1	IF END (00), GOTO NEXT 1
0396 308f bd 30 c3		JSR SENDD	
0397 3092 18 08		INY	NEXT BYTE
0398 3094 20 f4		BRA PDOD	RETURN TO NEXT
0399 3096 39	PDOD1:	RTS	
0400			
040			
0402	*Time Delay Routine		
0403 3097 c6 0f	T20:	LDB #\$0F	>20 uS DELAY
0404 3099 20 00		BRA TDLY	
0405 309b 5a	TDLY:	DECB	
0406 309c c1 00		CMPB #\$00	
0407 309e 26 fb		BNE TDLY	
0408 30a0 39		RTS	
0409			
0410 30a1 ce 00 0f	TD150:	LDX #\$000F	150us DELAY
0411 30a4 09	TDELAY:	DEX	DECREMENT COUNT
0412 30a5 8c 00 00		CPX #\$0000	COUNT = 0?
0413 30a8 26 fa		BNE TDELAY	IF NOT DONE, GOTO TDELAY
0414 30aa 39		RTS	RETURN FROM SUBROUTINE
0415			
0416	*Clear Screen, Cursor Home, and Send Control Instruction		
0417			
0418 30ab 86 01	CSCREEN:	LDA #\$0001	CLEAR DISPLAY
0419 30ad 20 08		BRA SENDI	SEND INSTRUCTION
0420 30af 86 02	HOME:	LDA #\$0002	CURSOR HOME
0421 30b1 20 04		BRA SENDI	SEND INSTRUCTION
0422 30b3 86 c0	LINE2:	LDA #\$00C0	SET CURSOR TO LINE 2
0423 30b5 20 00		BRA SENDI	SEND INSTRUCTION
0424 30b7 ce 40 00	SENDI:	LDX #PORTAB	SET UP DATA TRANSFER
0425 30ba a7 06		STAA 6,X	STORE AT PIA PORT A
0426 30bc 1c 07 02		BSET 7,X \$02	DISPLAY E HIGH (PSD PORT B PIN 1)
0427 30bf 1d 07 02		BCLR 7,X \$02	DISPLAY E LOW (PSD PORT B PIN 1)
0428 30c2 39		RTS	
0429			
0430	*Send Data to Display		
0431			
0432 30c3 bd 30 a1	SENDI:	JSR TD150	150 uS TIME DELAY
0433 30c6 ce 40 00		LDX #PORTAB	SET UP DATA TRANSFER
0434 30c9 a7 06		STAA 6,X	SEND DATA
0435 30cb 1c 07 01		BSET 7,X \$01	DISPLAY RS HIGH
0436 30ce 1c 07 02		BSET 7,X \$02	DISPLAY E HIGH
0437 30d1 1d 07 02		BCLR 7,X \$02	DISPLAY E LOW
0438 30d4 1d 07 01		BCLR 7,X \$01	DISPLAY RS LOW
0439 30d7 39		RTS	
0440			
0441	*Check if first number dialed is a 1 or 0 (number stored in A		
0442	*accum) and if it is, expand expected digits in number to 11.		
0443			
0444 30d8 7c 00 41	CHD:	INC N	INCREMENT DIGIT IN NUMBER
0445 30db c6 01		LDAB #\$01	
0446 30dd d1 41		CMPB N	COMPARE N TO 1 TO SEE IF 1ST DIGIT DIALED
0447 30df 26 1a		BNE CH2D	IF N=1, CHECK IF NO. DIALED IS 1 OR 0
0448			

1



**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0449
0450
0451 30e1 c6 00
0452 30e3 d1 43
0453 30e5 26 14
0454 30e7 c6 01
0455 30e9 d1 43
0456 30eb 26 0e
0457 30ed c6 0b
0458 30ef d7 42
0459
0460
0461
0462 30f1 d6 43
0463 30f3 0c
0464 30f4 59
0465 30f5 59
0466 30f6 59
0467 30f7 59
0468 30f8 d7 44
0469 30fa 39
0470
0471
0472
0473 30fb c6 02
0474 30fd d1 41
0475 30ff 26 07
0476 3101 d6 44
0477 3103 da 43
0478 3105 d7 44
0479 3107 39
0480
0481
0482
0483 3108 c6 03
0484 310a d1 41
0485 310c 26 0a
0486 310e d6 43
0487 3110 0c
0488 3111 59
0489 3112 59
0490 3113 59
0491 3114 59
0492 3115 d7 45
0493 3117 39
0494
0495
0496
0497 3118 c6 04
0498 311a d1 41
0499 311c 26 07
0500 311e d6 45
0501 3120 da 43
0502 3122 d7 45
0503 3124 39
0504
0505
0506
0507

*Check if first digit 0 or 1 and change M to 11 digits if it is.

```
LDAB    #00
CMPB    P          COMPARE 00 TO NUMBER DIALED
BNE     CH2D      IF 1ST NO. ISN'T 0, THEN CONTINUE
LDAB    #01
CMPB    P          COMPARE 01 TO NUMBER DIALED
BNE     CH2D      F 1ST NO. ISN'T 1, THEN CONTINUE
LDAB    #0B
STAB    M          SET M=$0B IF 1ST DIGIT IS 1 OR 0
                     OTHERWISE M=$07
```

*If first digit dialed, save in upper 4 bits of register P12.

```
LDAB    P          LOAD DIALED DIGIT IN REGISTER
CLC
ROLB
ROLB     ROTATE NUMBER LEFT TO MOVE IT TO UPPER 4—
          BITS IN REGISTER
ROLB
ROLB
STAB    P12       STORE IT TEMPORARILY
RTS
```

*Save second digit in lower 4 bits of P12

```
CH2D: LDAB    #02   DETERMINE IF DIGIT DIALED WAS
        CMPB    N   SECOND DIGIT AND
        BNE     CH3D IF IT IS
        LDAB    P12  STORE 2ND DIGIT IN LOWER 4 BITS OF P12
        ORAB    P    BY "OR"ING IT W/P12 WHICH ALREADY HAS 1ST
        STAB    P12  DIGIT SAVED IN UPPER 4 BITS
        RTS
```

*Save third digit in upper 4 bits of P34

```
CH3D: LDAB    #03   DETERMINE IF DIGIT DIALED WAS
        CMPB    N   THIRD DIGIT AND
        BNE     CH4D IF IT IS
        LDAB    P    LOAD DIALED DIGIT IN REGISTER
        CLC
        ROLB
        ROLB     ROTATE NUMBER LEFT TO MOVE IT TO UPPER 4—
        ROLB     BITS IN REGISTER
        STAB    P34  STORE IT TEMPORARILY
        RTS
```

*Save fourth digit in lower 4 bits of P34

```
CH4D: LDAB    #04   DETERMINE IF DIGIT DIALED WAS
        CMPB    N   FOURTH DIGIT AND
        BNE     CH5D IF IT IS
        LDAB    P34  STORE 4TH DIGIT IN LOWER 4 BITS OF P34
        ORAB    P    BY "OR"ING IT W/P34 WHICH ALREADY HAS 3RD
        STAB    P34  DIGIT SAVED IN UPPER 4 BITS
        RTS
```

*Save fifth digit in upper 4 bits of P56



Appendix B. Feature Phone Software Listing (Cont.)

0508 3125 c6 05	CH5D:	LDAB	#\$05	DETERMINE IF DIGIT DIALED WAS
0509 3127 d1 41		CMPB	N	FIFTH DIGIT AND
0510 3129 26 0a		BNE	CH6D	IF IT IS
0511 312b d6 43		LDAB	P	LOAD DIALED DIGIT IN REGISTER
0512 312d 0c		CLC		CLEAR CARRY BIT
0513 312e 59		ROLB		ROTATE NUMBER LEFT TO MOVE IT TO UPPER 4—
0514 312f 59		ROLB		BITS IN REGISTER
0515 3130 59		ROLB		
0516 3131 59		ROLB		
0517 3132 d7 46		STAB P56		STORE IT TEMPORARILY
0518 3134 39		RTS		
0519				
0520				
0521				
0522 3135 c6 06	CH6D:	LDAB	#\$06	DETERMINE IF DIGIT DIALED WAS
0523 3137 d1 41		CMPB	N	SIXTH DIGIT AND
0524 3139 26 07		BNE	CH7D	IF IT IS
0525 313b d6 46		LDAB	P56	STORE 6TH DIGIT IN LOWER 4 BITS OF P56
0526 313d da 43		ORAB	P	BY "OR"ING IT W/P56 WHICH ALREADY HAS 5TH
0527 313f d7 46		STAB	P56	DIGIT SAVED IN UPPER 4 BITS
0528 3141 39		RTS		
0529				
0530				
0531				
0532 3142 c6 07	CH7D:	LDAB	#\$07	DETERMINE IF DIGIT DIALED WAS
0533 3144 d1 41		CMPB	N	SEVENTH DIGIT AND
0534 3146 26 15		BNE	CH8D	IF IT IS
0535 3148 d6 43		LDAB	P	LOAD DIALED DIGIT IN REGISTER
0536 314a 0c		CLC		CLEAR CARRY BIT
0537 314b 59		ROLB		ROTATE NUMBER LEFT TO MOVE IT TO UPPER 4—
0538 314c 59		ROLB		BITS IN REGISTER
0539 314d 59		ROLB		
0540 314e 59		ROLB		
0541 314f d6 42		LDAB	M	
0542 3151 d1 41		CMPB	N	CHECK IF M=N? - ALL NUMBERS DIALED
0543 3153 27 02		BEQ	NEXT	
0544 3155 ca 0a		ORB	#\$0A	
0545 3157 d7 47	NEXT:	STAB	P78	STORE IT TEMPORARILY
0546 3159 bd 31 9d		JSR	CHAD	
0547 315c 39		RTS		
0548				
0549				
0550				
0551 315d c6 08	CH8D:	LDAB	#\$08	DETERMINE IF DIGIT DIALED WAS
0552 315f d1 41		CMPB	N	EIGHTH DIGIT AND
0553 3161 26 07		BNE	CH9D	IF IT IS
0554 3163 d6 47		LDAB	P78	STORE 8TH DIGIT IN LOWER 4 BITS OF P78
0555 3165 da 43		ORAB	P	BY "OR"ING IT W/P78 WHICH ALREADY HAS 7TH
0556 3167 d7 47		STAB	P78	DIGIT SAVED IN UPPER 4 BITS
0557 3169 39		RTS		
0558				
0559				
0560				

*Save sixth digit in lower 4 bits of P56

*Save seventh digit in upper 4 bits of P78

*Save eighth digit in lower 4 bits of P78

*Save ninth digit in upper 4 bits of P910

1

**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0561 316a c6 09	CH9D:	LDAB	#\$09	DETERMINE IF DIGIT DIALED WAS
0562 316c d1 41		CMPB	N	NINTH DIGIT AND
0563 316e 26 0a		BNE	CH10D	IF IT IS
0564 3170 d6 43		LDAB	P	LOAD DIALED DIGIT IN REGISTER
0565 3172 0c		CLC		CLEAR CARRY BIT
0566 3173 59		ROLB		ROTATE NUMBER LEFT TO MOVE IT TO UPPER 4—
0567 3174 59		ROLB		BITS IN REGISTER
0568 3175 59		ROLB		
0569 3176 59		ROLB		
0570 3177 d7 48		STAB	P910	STORE IT TEMPORARILY
0571 3179 39		RTS		
0572				
0573				
0574				
				*Save tenth digit in lower 4 bits of P910
0575 317a c6 0a	CH10D:	LDAB	#\$0A	DETERMINE IF DIGIT DIALED WAS
0576 317c d1 41		CMPB	N	TENTH DIGIT AND
0577 317e 26 07		BNE	CH11D	F IT IS
0578 3180 d6 48		LDAB	P910	STORE 10TH DIGIT IN LOWER 4 BITS OF P910
0579 3182 da 43		ORAB	P	BY "OR"ING IT W/P910 WHICH ALREADY HAS 9TH
0580 3184 d7 48		STAB	P910	DIGIT SAVED IN UPPER 4 BITS
0581 3186 39		RTS		
0582				
0583				
0584				
				*Save eleventh digit in upper 4 bits of P910
0585 3187 c6 0b	CH11D:	LDAB	#\$0B	DETERMINE IF DIGIT DIALED WAS
0586 3189 d1 41		CMPB	N	ELEVENTH DIGIT AND
0587 318b 26 0f		BNE	CDONE	IF IT IS
0588 318d d6 43		LDAB	P	LOAD DIALED DIGIT IN REGISTER
0589 318f 0c		CLC		CLEAR CARRY BIT
0590 3190 59		ROLB		ROTATE NUMBER LEFT TO MOVE IT TO UPPER 4—
0591 3191 59		ROLB		BITS IN REGISTER
0592 3192 59		ROLB		
0593 3193 59		ROLB		
0594 3194 ca 0a		ORB	#\$0A	LAST 4 BITS WILL REPRESENT END OF NO.
0595 3196 d7 49		STAB	P1112	STORE IT TEMPORARILY
0596 3198 bd 31 9d		JSR	CHAD	
0597 319b 39		RTS		
0598				
0599 319c 39	CDONE:	RTS		
0600				
0601				
0602				
				*Check if all digits in number dialed
0603 319d d6 42	CHAD:	LDAB	M	
0604 319f d1 41		CMPB	N	CHECK IF M=N? - ALL NUMBERS DIALED
0605 31a1 27 01		BEQ	STBUF	STORE ENTERED NUMBER IN BUFFER RAM
0606 31a3 39		RTS		
0607				
0608				
0609				
				*Enter Dialed Number in Buffer RAM

Appendix B. Feature Phone Software Listing (Cont.)

0610 31a4 de 4a	STBUF:	LDX	SWRPTR	STORE SRAM WRITE POINTER IN X REG
0611 31a6 86 08		LDA	#\$08	
0612 31a8 9b 4a		ADDA	SWRPTR	
0613 31aa 97 4a		STAA	SWRPTR	STORED PHONE NO. (INCREMENT WRITE PTR)
0614 31ac 97 4b		STAA	SRDPTR	SET READ POINTER AT LAST WRITTEN LOCATION
0615 31ae 96 44		LDA	P12	
0616 31b0 a7 00		STAA	0,X	STORE FIRST TWO DIGITS IN RAM MEMORY
0617 31b2 96 45		LDA	P34	
0618 31b4 a7 01		STAA	1,X	STORE NEXT TWO DIGITS IN RAM MEMORY
0619 31b6 96 46		LDA	P56	
0620 31b8 a7 02		STAA	2,X	STORE NEXT TWO DIGITS IN RAM MEMORY
0621 31ba 96 47		LDA	P78	
0622 31bc a7 03		STAA	3,X	STORE NEXT TWO DIGITS IN RAM MEMORY
0623 31be 96 48		LDA	P910	
0624 31c0 a7 04		STAA	4,X	STORE NEXT TWO DIGITS IN RAM MEMORY
0625 31c2 96 49		LDA	P1112	
0626 31c4 a7 05		STAA	5,X	STORE LAST DIGIT IN RAM MEMORY
0627 31c6 39		RTS		
0628				
0629				
0630				
0631 31c7 18 de 4b	SCROLL:	LDY	SRDPTR	LOAD ADDRESS OF READ POINTER IN Y REGISTER
0632				
0633 31ca 18 a6 00	NXTNUM:	LDA	0,Y	LOAD 1ST DIGIT, PX?, FROM RAM IN A ACCUM
0634 31cd 44		LSRA	BY	GETTING PXY AND SHIFTING RIGHT
0635 31ce 44		LSRA		4 TIMES
0636 31cf 44		LSRA		TO ACHIEVE 0000/XXXX
0637 31d0 44		LSRA		THEN MAKE IT
0638 31d1 8a 30		ORA	#\$30	ASCII EQUIVALENT - 0011/XXXX
0639 31d3 bd 30 c3		JSR	SEND	SEND DATA TO DISPLAY
0640 31d6 18 a6 00		LDA	0,Y	READ 2ND DIGIT, P?Y, FROM RAM MEMORY
0641 31d9 84 0f		ANDA	#\$0F	CLEAR UPPER 4 BITS
0642 31db 8a 30		ORA	#\$30	SET UPPER 4 BITS TO 3 → 0011/YYYY
0643 31dd 81 3a		CMPA	#\$3A	CHECK IF LAST NUMBER DIALED
0644 31df 27 08		BEQ	NUMRET	AND IF IT IS, RETURN, OTHERWISE,
0645 31e1 bd 30 c3		JSR	SEND	SEND DATA TO DISPLAY,
0646 31e4 18 08		INY		INCREMENT ADDRESS, AND
0647 31e6 bd 31 ca		JSR	NXTNUM	RETRIEVE NEXT NUMBER
0648 31e9 39	NUMRET:	RTS		
0649				
0650				
0651				
0652 31ea 18 de 4b	DIALNO:	LDY	SRDPTR	TRANSFER CONTENTS AT ADDRESS CONTAINED IN
0653 31ed 18 e6 00		LDA	0,Y	READ POINTER TO ACCUM B
0654 31f0 ce 40 00	SDDATA:	LDX	#PORTAB	CONFIGURE PSD PORTB TO SEND NUMBER
0655 31f3 86 fb		LDA	#\$FB	SET ALL PORT B PINS AS OUTPUTS EXCEPT
0656 31f5 a7 05		STAA	5,X	PIN PB2 (DP)
0657 31f7 86 00		LDA	#\$00	INITIALIZE PORT B PINS 3,4,5,6 AS LOW
0658 31f9 a7 07		STAA	7,X	OUTPUTS
0659 31fb 1c 07 08	SCTL:	BSET	7,X \$08	/TO (TONE OUTPUT) HIGH
0660 31fe bd 30 97		JSR	T20	TDELAY 20uS
0661 3201 1c 07 20		BSET	7,X \$20	/DD (DATA DIRECTION) HIGH FOR OUTPUT
0662 3204 bd 30 97		JSR	T20	TDELAY 20uS
0663				
0664 3207 86 02	NXT2NUM:	LDA	#\$02	LOOP TWICE TO SEND 2 DIGITS/8-BITS
0665 3209 18 ce 00 04	NXTDIG:	LDY	#\$04	LOOP 4 TIMES TO SEND 4 BITS/DIGIT IN NO.
0666 320d 1c 07 40	S4BIT:	BSET	7,X \$40	ETC /CL (CLOCK INPUT) HIGH
0667 3210 bd 30 97		JSR	T20	TDELAY 20uS
0668 3213 59		ROLB		SHIFT IN CARRY BIT
0669 3214 24 06		BCC	SNDLO1	BRANCH IF CARRY CLEAR
0670 3216 1c 07 10		BSET	7,X \$10	SET CARRY BIT AND SEND DATA ON I/O
0671 3219 bd 32 1f		JSR	DELAY	

**Appendix B.
Feature Phone
Software Listing
(Cont.)**

0672 321c 1d 07 10	SNDLO1:	BCLR	7,X \$10	CLEAR CARRY BIT AND SEND DATA ON I/O
0673 321f bd 30 97	DELAY:	JSR	T20	TDELAY 20uS - COULD CHANGE TO 10uS
0674 3222 1d 07 40		BCLR	7,X \$40	ETC /CL (CLOCK INPUT) LOW
0675 3225 bd 30 97		JSR	T20	TDELAY 20uS
0676 3228 18 09		DEY		DECREMENT COUNT
0677 322a 18 8c 00 00		CPY	#\$00	COUNT = 0?
0678 322e 26 dd		BNE	S4BIT	IF NOT DONE, GO TO S4BIT
0679				
0680 3230 1d 07 08		BCLR	7,X \$08	/TO LOW FOR TONE GENERATION INTERVAL
0681 3233 bd 30 97		JSR	T20	TDELAY 20uS
0682 3236 1c 07 08		BSET	7,X \$08	/TO HIGH FOR TONE GENERATION INTERVAL
0683 3239 bd 30 97		JSR	T20	TDELAY 20uS
0684				
0685 323c 4a		DECA		DECREMENT COUNT
0686 323d 81 00		CMPA	#\$00	COUNT = 0?
0687 323f 26 0f		BNE	CHKDIG	CHECK FOR LAST DIGIT
0688 3241 bd 32 09	GETNXT:	JSR	NXTDIG	
0689 3244 18 de 4b		LDY	SRDPTR	
0690 3247 18 08		INY		INCREMENT ADDRESS FOR NEXT 2 NUMBERS
0691 3249 18 e6 00		LDAB	0,Y	AND LOAD IN B ACCUM
0692 324c bd 32 07		JSR	NXT2NUM	
0693 324f 39		RTS		
0694				
0695 3250 c4 f0	CHKDIG:	ANDB	#\$F0	CLEAR LOWER 4-BITS
0696 3252 c1 a0		CMPB	#\$A0	COMPARE DIGIT THAT WILL BE SENT W/\$A0
0697 3254 27 02		BEQ	STORENO	IF LAST NO. A0, STORE SINCE ALL NOS SENT
0698 3256 20 e9		BRA	GETNXT	
0699				
0700 3258 18 de 4b	STORENO:	LDY	SRDPTR	COPY DIALED NUMBER INTO RAM MEMORY
0701 325b de 4a		LDX	SWRPTR	
0702 325d 18 a6 00		LDAA	0,Y	
0703 3260 a7 00		STAA	0,X	
0704 3262 18 a6 01		LDAA	1,Y	
0705 3265 a7 01		STAA	1,X	
0706 3267 18 a6 02		LDAA	2,Y	
0707 326a a7 02		STAA	2,X	
0708 326c 18 a6 03		LDAA	3,Y	
0709 326f a7 03		STAA	3,X	
0710 3271 18 a6 04		LDAA	4,Y	
0711 3274 a7 04		STAA	4,X	
0712 3276 18 a6 05		LDAA	5,Y	
0713 3279 a7 05		STAA	5,X	
0714 327b 18 a6 06		LDAA	6,Y	
0715 327e a7 06		STAA	6,X	
0716 3280 18 a6 07		LDAA	7,Y	
0717 3283 a7 07		STAA	7,X	
0718 3285 39		RTS		
0719				
0720				
				*Reset and Interrupt Vectors
072				
0722 fff2		ORG	\$\$\$FF2	
0723				
0724 fff2 20 a5	IRQ:	FDB	GDATA	/IRQ - EXTERNAL PIN
0725 fff4 20 00	XIRQ:	FDB	START	/XIRQ PIN (PSEUDO-NONMASKABLE)
0726 fff6 20 00	SWI:	FDB	START	SOFTWARE INTERRUPT
0727 fff8 20 00	IOT:	FDB	START	ILLEGAL OPCODE TRAP (START OVER)
0728 fffa 20 00	COPS:	FDB	START	COP FAILURE (RESET)
0729 fffc 20 00	COPS1:	FDB	START	COP CLOCK MONITOR FAIL (RESET)
0730 fffe 20 00	RESET:	FDB	START	RESET
0731				
0732	END			THE END



Programmable Peripheral Application Note 041

Detailed Step-By-Step Design Implementation of an M68HC11 and PSD311 or PSD311R

By Steve Torp – Motorola Semiconductor and Karen Spesard – WSI, Inc.

Introduction

The purpose of this application note is to show the steps involved in moving from an OTP M68HC711 or expanded mode M68HC11 multi-chip solution to a two-chip solution using an expanded mode M68HC11 and a WSI PSD311 or PSD311R (SRAMless) Programmable MCU Peripheral. This two-chip approach provides many advantages such as increased system flexibility and several options for more EPROM and SRAM memory while maintaining low power system needs.

The main areas to consider when implementing the two-chip M68HC11 and WSI PSD311 solution are discussed below. They are mapping the PSD in the M68HC11 address space, configuring the PSD using the PSD-SILVER software package, modifying the microcontroller code as necessary, and programming the PSD.

M68HC11 Expanded Mode Considerations

An important advantage of the M68HC11 is that it has many subsystems. They are A/D, E²PROM, synchronous peripheral interface (SPI), and serial communications interface (SCI). As a result, these subsystems will have to be defined by the system designer in the address map. In addition, the system design must also incorporate memory and I/O mapping definitions.

A typical single-chip OTP design could incorporate an M68HC711E9 and specific system I/O. The M68HC711E9 has 8-bit A/D, 512 bytes E²PROM, one SPI, and one SCI as well as 12K bytes of EPROM and 512 bytes of SRAM. A total of 38 I/O lines are available for the user to define in the system.

The OTP M68HC711E9 mapping is specified in Motorola's M68HC11 Reference Manual. The E²PROM and EPROM are directly mapped at \$B600-\$B7FF and \$D000-\$FFFF, respectively. The 64 byte register block and the SRAM areas, however, do require some consideration. The 64 byte register block specifies the SPI, SCI and I/O Ports. After reset, it is located at \$1000-\$103F. However, the register block can be relocated, if necessary, on any 4K block boundary anywhere within the 64K address space. The INIT register is written to at location \$103D but to do this keep in mind there is a time protection limitation of 64 clock cycles out of RESET.

The other area to be considered for mapping is the 512 byte SRAM. After reset, the default mapping of the SRAM is at \$0000-\$01FF but it can also be relocated anywhere in the 64K address space on a 4K byte boundary by modifying the INIT register.

A typical expanded OTP or ROMless M68HC11 design places address and data signals on the M68HC11 Port B and Port C pins so it can address 64K bytes of external memory. Higher-order address bits are output on the Port B pins and lower-order address bits and the 8-bit data bus are multiplexed on the Port C pins. The AS pin provides the control output used in demultiplexing the low-order address at Port C. The R/W pin is used to control the direction of data transfer on the Port C bus. To convert from single-chip to expanded mode on the M68HC11, simply pull the MODA pin HIGH to V_{DD}.

M68HC11
Expanded Mode
Considerations
 (Cont.)

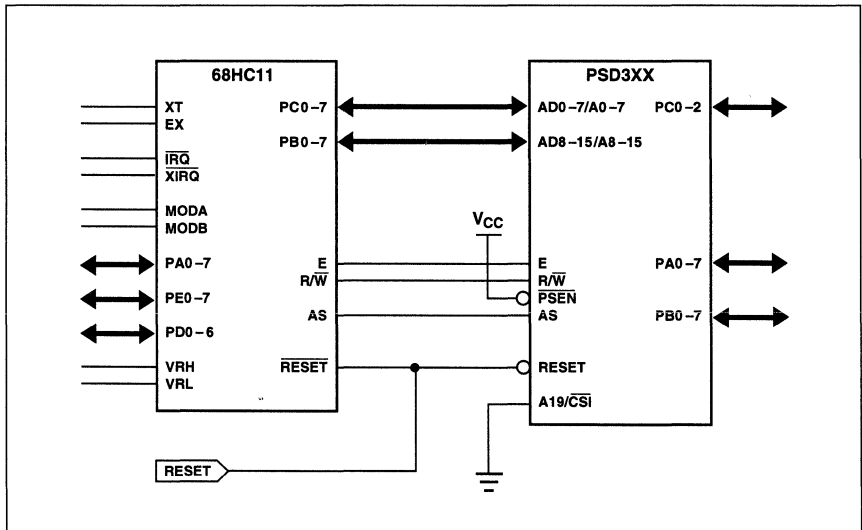
In multi-chip expanded mode designs, an external latch such as a 74HC373 is normally required with the microcontroller to demultiplex the address from the data. In addition, some address decoding would have to be defined for memory and peripheral mapping which can be done in discrete logic, such as by using a 74HC138, or a simple PLD such as a 16V8 or 22V10. And finally, if additional direct mapped I/O is required in expanded mode, additional logic components will be necessary such as transparent latches, i.e., 74HC374, 74HC341, and 68HC24.

Figure 1 illustrates the two-chip configuration with the M68HC11 in expanded mode. The PSD311 or PSD311R integrates 32K bytes of EPROM, an optional 2K bytes of SRAM, a demultiplexing latch, programmable address decoding, other programmable logic, and 19 user-configurable I/Os. As a result, the functionality in the PSD incorporates the components that would be necessary in a cumbersome multi-chip configuration by integrating on-chip the demultiplexing latches and address decoding. The PSD also integrates the EPROM and reconstructs the two ports of I/O that are lost when placing the M68HC11 in expanded mode.

The interface of the M68HC11 with the PSD is quite simple. The address/data bus (AD0–7 and A8–15) from the expanded mode M68HC11 Ports B and C map directly to the address/data bus of the PSD (AD0–7, A8–15). The R/W, E, and AS from the M68HC11 map directly to the E/DS, R/W, and ALE/AS of the PSD. The RESET pin on the PSD will be tied to the RESET of the M68HC11. Since the PSEN pin on the PSD311 is not used, it will be tied HIGH. Finally, the 19 configurable I/Os and A19/CS \bar{I} will be available to be used to reconstruct the M68HC11 Ports B and C and for additional expansion.

The address mapping for expanded mode designs is similar to the OTP design, but now the external memory and peripheral I/O need to be considered. The external memory and I/O should be mapped to avoid conflicts with internally mapped resources. If there is a conflict, the internal resources always have priority and the address and data will not be presented externally. Keep in mind the interrupt vector assignments are located at \$FFC0 – \$FFFF and must be physically mapped at these locations.

Figure 1. Two-Chip Configuration with PSD311 and M68HC11 in Expanded Mode



M68HC11 Expanded Mode Considerations (Cont.)

In the two-chip configuration of the PSD and M68HC11, the memory and peripheral I/O mapping in the PSD device is achieved using the WSI PSD software. There are two software packages available.

PSD-SILVER software supports the PSD3XX devices and includes the MAPLE and MAPPRO software modules which run under the DOS platform. MAPLE software is used to configure the PSD chip. It features simple menu driven commands for selecting different device configurations. It also provides mapping of the EPROM, SRAM, and chip select outputs into the user's address space, and locates the files to be programmed into the EPROM segments. MAPPRO enables the user to program the PSD on a WSI MagicPro III programmer.

The second software package, PSDsoft™ (WS7001 or WS7002), supports the PSD3XX, PSD4XX, and PSD5XX families and runs under MicroSoft® Windows®. It includes PSDabel, PSD configuration, PSD compiler, PSDsilos III simulator and PSD programming software. The PSDsoft environment allows design and simulation of the PSD on-chip PLD logic under Data I/O ABEL, PSD interface selections to any MCU, configuration of the I/O, and address mapping of the EPROM and SRAM memory, among other things.

For simplification, a step-by-step procedure for configuring the PSD311 or PSD311R using the PSD-SILVER software is shown below.

Configuring The PSD311 With The PSD-Silver Software

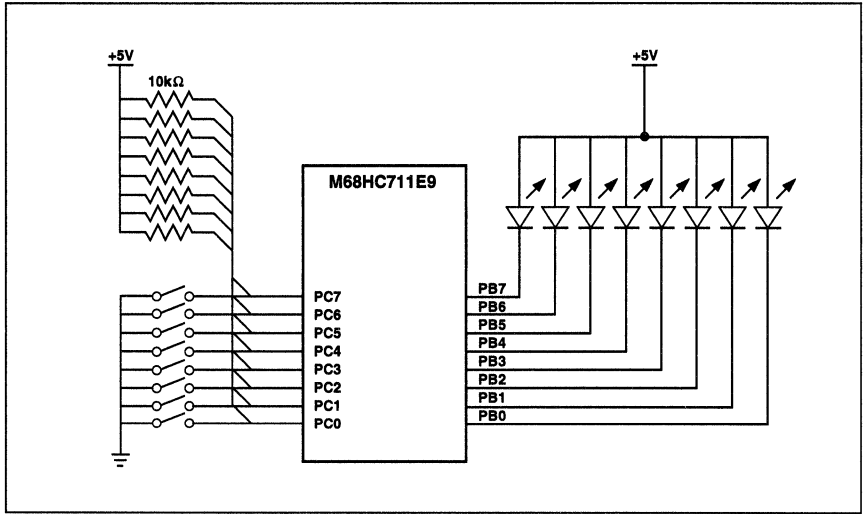
Before the PSD311 is configured to interface with the M68HC11, the rest of the system requirements need to be defined. Of course, the memory and I/O port mapping will be needed. In addition, the PSD311 can integrate some chip-select and glue logic, which can help reduce other logic components on the board. These should be specified by the user in the application.

The following example shown in Figures 2 and 3 will illustrate an application that reads eight simple DIP switches and display the values on two 7-segment LEDs. The first schematic uses a single-chip OTP M68HC711E9. The second schematic is a simple conversion to the ROMless M68HC11E1 used with the PSD311 and is functionally equivalent. In the second example, the PSD-SILVER software is used to configure the PSD311 to support this application. The PSD-SILVER's ease-of-use illustrates the flexibility of the PSD which will be demonstrated below.

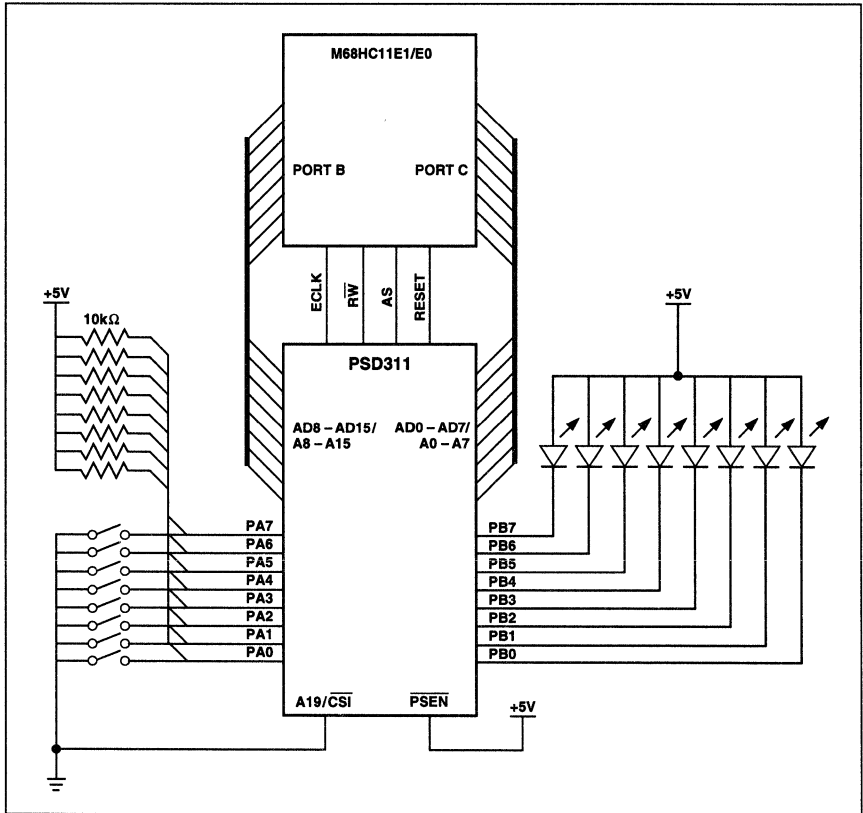
The second example with the PSD311 will include mapping for 12Kbytes of the 32Kbyte EPROM that is available for program storage, 2Kbytes of SRAM for data storage included on-chip, and 16 general-purpose MCU I/O pins. Additional system enhancements that could require chip select or additional logic can also be incorporated in the PSD311 PLD arrays.

**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 2. Single-Chip M68HC711E9 Example Application



**Figure 3. Twin-Chip M68HC11E1/E0 With The WSI PSD311
Example Application**

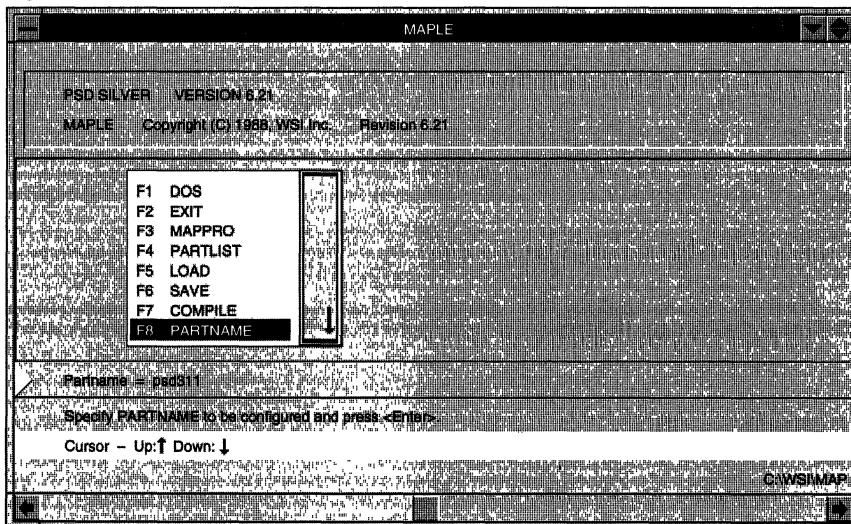


**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

The PSD-SILVER software menus for the PSD311 are illustrated and described on the following pages for this application. Figure 4 shows the PSD-SILVER MAPLE MAIN menu. It is invoked by typing MAPLE at the DOS prompt when in the WSI\MAP subdirectory. It lists the function keys and their associated operations. F1 suspends the MAPLE software to DOS for file editing or updating. F2 exits the program and returns the user to the DOS environment. F3 selects the programmer option so the user can program the compiled object file into the PSD311 device when a WSI MagicProRIII programmer is connected to the system. The LOAD selection, (F5), loads an existing PSD configuration into the MAPLE environment for editing and recompiling. F6 saves that program under a user-defined name. F7 compiles the user-generated file into an object file that can be transferred to the programmer. F8 provides part type selection – in this case, the PSD311.

1

Figure 4. PSD-Silver MAPLE Main Menu



After selecting PARTNAME, Figure 5 illustrates a second menu that appears to the right of the MAIN menu. The list shows ALIASES, CONFIGURATION, PORT C, PORT A, PORT B, and ADDRESS MAP. The designer selects each choice, starting from ALIASES, and moves down through the list configuring each option. The ALIASES menu shown in Figure 6 lets the user individually define the port pins with user-relevant names. In this example, we will not enter any alias names.

**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 5. PSD-Silver Main Menu for Configuring the PSD311

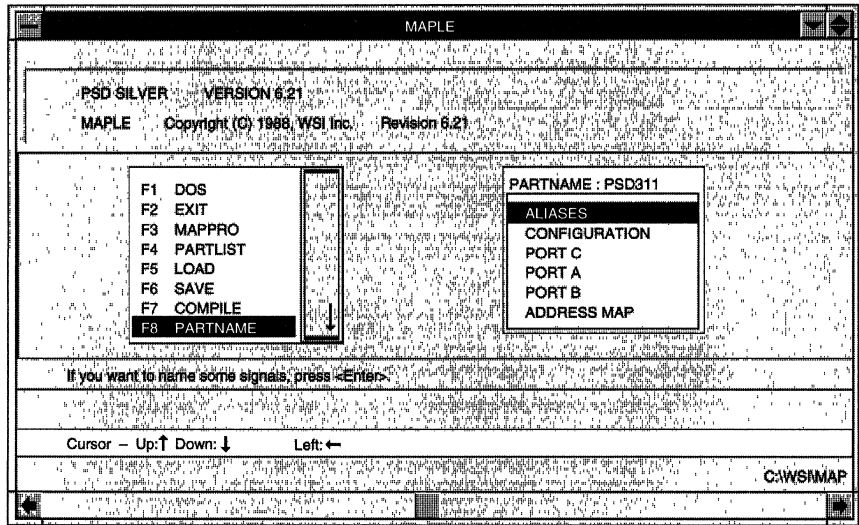
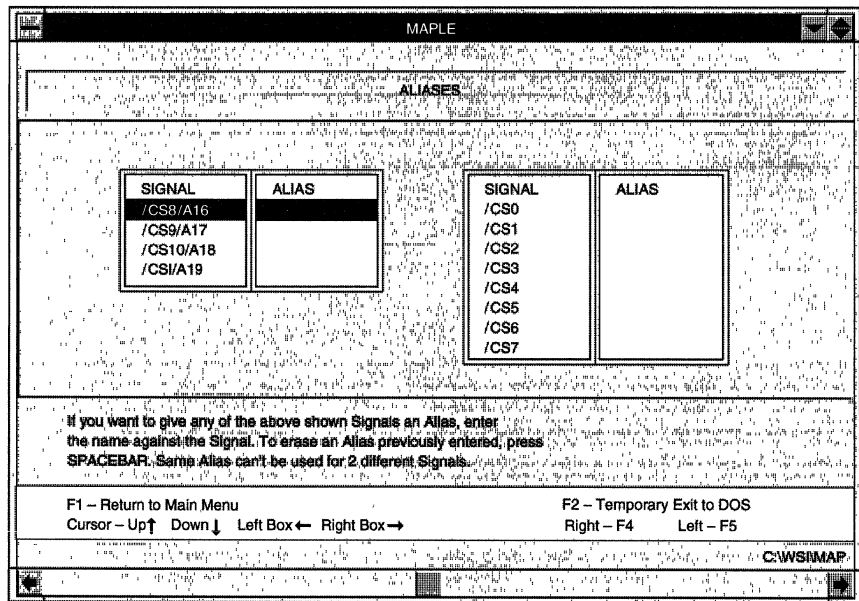


Figure 6. Aliases Menu for Ports B and C



**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 7 shows the CONFIGURATION menu which is accessed by selecting CONFIGURATION in the MAIN menu. In our example, the PSD311 has been configured for use with the M68HC11E1, i.e., the 8-bit address/data bus is multiplexed. The chip-select input is chosen rather than the A19 input. The RESET polarity is active LOW, the ALE (AS) polarity is active HIGH, and R \bar{W} and E control inputs are enabled. The inputs A16-A19 are transparent and the EPROM and SRAM share the same 64K address space (combined memory mode).

Figure 7. PSD Configuration Menu with the M68HC11 Interface

MAPLE

CONFIGURATION

Address/Data Mode (Multiplexed: MX, Non-Multiplexed: NM)	MX
Data Bus Width (8 bits for PSD311) (8/16 bits for PSD301)	8
CSI (Power-Down/Chip Enable) or A19	CSI
Reset Polarity (Active Low: LO, Active High: HI)	LO
ALE Polarity (Active Low: LO, Active High: HI)	HI
WR and RD (WRD) or R/W and E (RWE)?	RWE
A19-A16 Transparent or Latched by ALE (Trans: T, Latched: L)	T
Are you using PSEN? (Y/N)	N

If Address and Data Buses are not multiplexed, press SPACEBAR.

F1- Return to Main Menu F2-Temporary exit to DOS Cursor Up: ↑ Down: ↓

C:\WSMAP

After the device interface is configured, the PSD311 Port C can be set up. If the MAIN menu is invoked from the CONFIGURATION menu by pressing F1, PORT C can be selected as shown in Figure 5. As shown in Figure 8, the individual selection of CS/AI configures the three pins as chip-select outputs CS8, CS9, and CS10. The chip-select equations are specified by selecting F3 for the chip-select definition and entering logic HIGH, LOW, or “don't care” conditions in the column of the logic inputs that you need “AND”ed together. For example, if a chip-select is needed at location \$5800-\$5FFF, entering 01011 in the row under A15, A14, A13, A12, and A11 results in the following active low chip-select equation as shown created: CS8 = !A15 & A14 & !A13 & A12 & A11. Port C can also be used for general purpose logic inputs to create programmable logic output equations or it can be used to extend the address space of a microcontroller by bringing in A16, A17, and A18. If the Port C pins are not needed for any of these functions, leave them as chip select outputs and don't specify any equation.

**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 8. Example of Port C Configuration for Chip-Selects

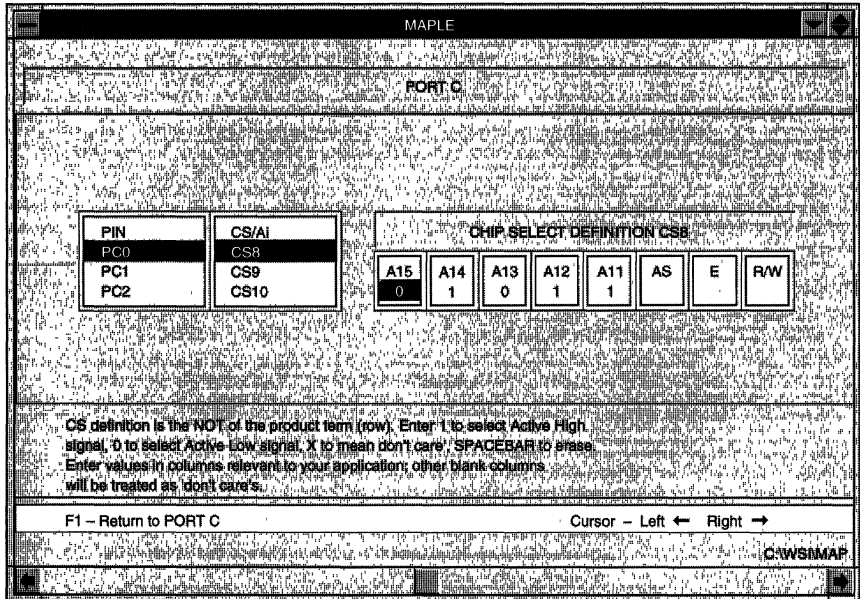
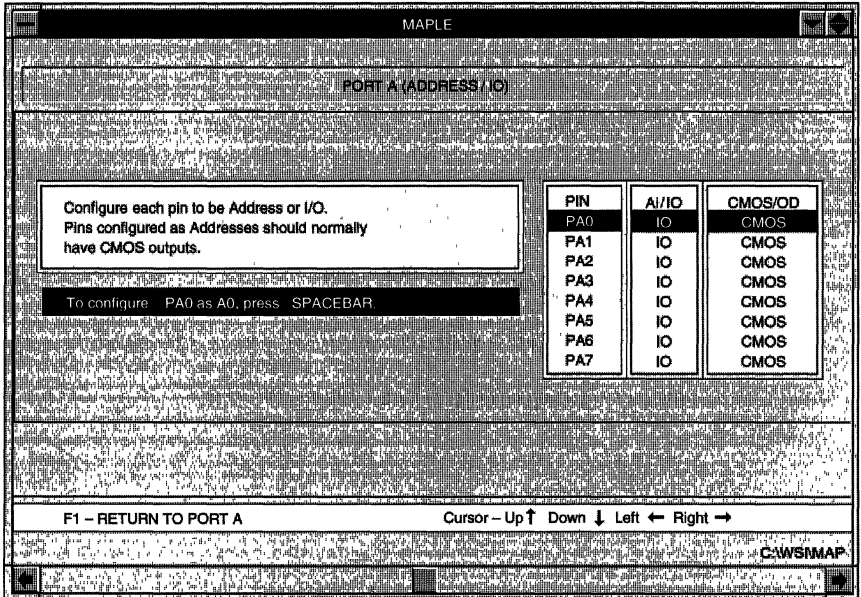


Figure 9 shows the PSD311 PORT A in the Address/I/O configuration. Since the M68HC11E1 Port B is output only and will be reconstructed on the PSD311 Port A, all eight of the PSD311 Port A pins will be configured as I/Os with CMOS outputs and the data direction register for the PSD311 Port A will be set to 'FF' to position it for outputs. If these eight outputs are not needed, one of the alternate configurations for the PSD311 Port A is Lower Order Latched Address bits which includes an internal output latch on Port A.

Figure 10 gives the PSD311 PORT B selection as eight I/Os with CMOS outputs which will reconstruct all eight bidirectional I/Os on the M68HC11E1 Port C. The direction of the individual I/O pins in the PSD311 Port B is determined by the original OTP application. The direction is set by writing to the data direction register. To make a pin an input, the appropriate bit in the register must be cleared and to make a pin an output, the appropriate bit must be set. If all eight I/Os are not needed, the alternate configurations for the PSD311 Port B pins are chip selects.

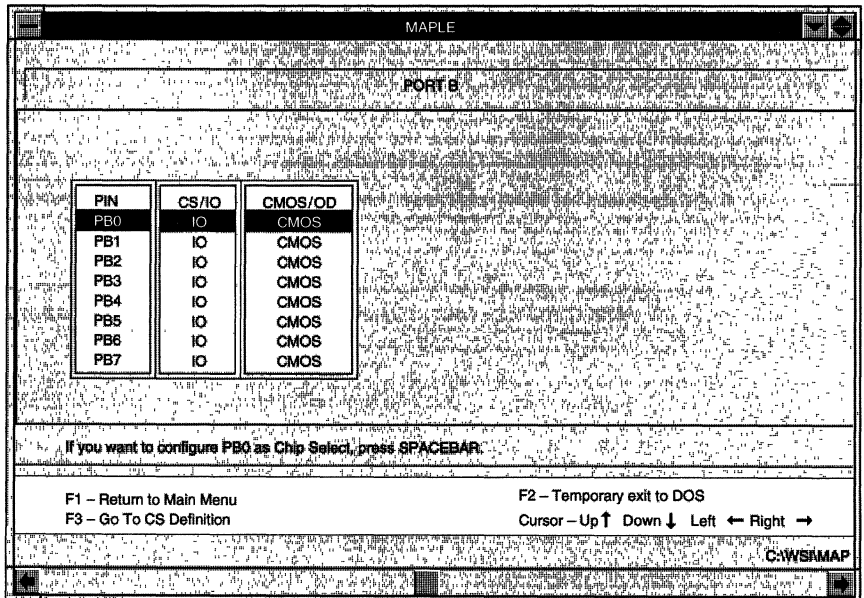
**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 9. Port A Configuration for All Eight Pins as I/Os



1

Figure 10. Port B Configuration for All Eight Pins as I/Os



**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 11 shows the ADDRESS MAP menu. The designer can enter a binary code for the address range assignments of the various select lines or a hexadecimal starting and stopping address can be entered to locate the memory and peripherals within the M68HC11E1 address space. ES0-ES7 are the chip-selects for the eight 4Kbyte EPROM blocks, RS0 is the chip-select for the 2Kbyte SRAM, and CSP is the chip-select for the CSIOPORT base address. A space for individual hexadecimal files to be programmed into the PSD311 EPROM is reserved under the FILENAME section. The M68HC11E1 code listed under the FILE NAME "68HC11.hex" must be in Intel MCS hex format. If Intel MCS hex format is not available, a conversion program to convert Motorola S records to Intel MCS hex is included with the PSD-SILVER software package.

Figure 11. Address Map Menu for Selecting Address Locations of CSIOPORT (Ports A and B), EPROM, and SRAM

	A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	SEGMENT START	SEGMENT STOP	FILE START	FILE STOP	FILE NAME
ES0	N	N	N	N	1	1	0	1	N	D000	DFFF	D000	DFFF	68HC11.HEX
ES1	N	N	N	N	1	1	1	0	N	E000	FFFF	E000	FFFF	68HC11.HEX
ES2	N	N	N	N	1	1	1	1	N	F000	FFFF	F000	FFFF	68HC11.HEX
ES3	N	N	N	N					N					
ES4	N	N	N	N					N					
ES5	N	N	N	N					N					
ES6	N	N	N	N					N					
ES7	N	N	N	N					N					
RS0	N	N	N	N	0	0	1	1	0	3000	37FF			
CSP	N	N	N	N	0	0	1	0	0	2000	27FF			

Fill in A19 – A11 (Binary) or SEGMENT START (Hex); and FILE (START, STOP) and FILE NAME.
 Use SPACEBAR to erase any field value.
 F1 – Return to Main Menu
 F2 – Temporary exit to DOS
 F3 – Go to Help
 Cursor – Up:↑ Down:↓ Left Col:← Right Col:→ Right – F4 Left – F5

C:\WSIMAP

In our application example, three 4Kbyte sections of code for a total of 12Kbytes of EPROM will be mapped from \$D000-\$FFFF. The filename with the code is called 68HC11.HEX and is located in the same directory as the MAPLE software. The additional PSD311 SRAM will be located at \$3000-\$37FF and the CSIOPORT base address will be at location \$2000.

After the configuration has been established, the user can return to the MAIN menu and SAVE (F6) the PSD311 configuration. Saving the configuration creates a filename.SV1 file which documents all of the selections made during the configuration process. The file created from our example is shown in Figure 12.

**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

Figure 12. Example Configuration Output File for PSD311

```

***** MAPLE 6.21 *****
                GLOBAL CONFIGURATION
Address/Data Mode:           MX
Data Bus Size:               8
CSI/A19:                    CSI
Reset Polarity:             LO
ALE Polarity:               HI
WRD/RWE:                    RWE
A16-A19 Transparent or Latched by ALE:  T
Are you using PSEN?         N
*****

                PORT A CONFIGURATION (Address/IO)
Bit No.  Ai/IO  CMOS/OD
0         IO    CMOS
1         IO    CMOS
2         IO    CMOS
3         IO    CMOS
4         IO    CMOS
5         IO    CMOS
6         IO    CMOS
7         IO    CMOS
*****

                PORT B CONFIGURATION
Bit No.  CS/IO  CMOS/OD
0         IO    CMOS
1         IO    CMOS
2         IO    CMOS
3         IO    CMOS
4         IO    CMOS
5         IO    CMOS
6         IO    CMOS
7         IO    CMOS
*****

                CHIP SELECT EQUATIONS
*****

                PORT C CONFIGURATION
Bit No.  CS/Ai
0         CS8
1         CS9
2         CS10

                CHIP SELECT EQUATIONS
/CS8 = /( /A15 * A14 * /A13 * A12 * A11)
*****

```

**Configuring
The PSD311
With The
PSD-Silver
Software
(Cont.)**

ADDRESS MAP

	A	A	A	A	A	A	A	A	A	SEGMT	SEGMT	EPROM	EPROM	File Name
	19	18	17	16	15	14	13	12	11	STRT	STOP	START	STOP	
ES0	N	N	N	N	1	1	0	1	N	D000	DFFF	d000	dfff	68HC11.HEX
ES1	N	N	N	N	1	1	1	0	N	E000	FFFF	e000	efff	68HC11.HEX
ES2	N	N	N	N	1	1	1	1	N	F000	FFFF	f000	ffff	68HC11.HEX
ES3	N	N	N	N					N					
ES4	N	N	N	N					N					
ES5	N	N	N	N					N					
ES6	N	N	N	N					N					
ES7	N	N	N	N					N					
RS0	N	N	N	N	0	0	1	1	0	3000	37FF			
CSP	N	N	N	N	0	0	1	0	0	2000	27FF			

***** ADDRESS MAP (EQUATIONS) *****

ES0 = /A15 * /A14 * /A13 * A12
 ES1 = /A15 * /A14 * A13 * /A12
 ES2 = /A15 * /A14 * A13 * A12
 ES3 =
 ES4 =
 ES5 =
 ES6 =
 ES7 =
 RS0 = /A15 * /A14 * A13 * A12 * /A11
 CSP = /A15 * /A14 * A13 * /A12 * /A11

***** END *****

Finally, the user will invoke the COMPILE (F7) option. The compile option merges the PSD configuration information with the code that would normally be programmed into the EPROM to create one output file with a filename.OBJ extension.



Software Considerations

The code for the M68HC11, when transitioning from an OTP single chip version to a twin-chip ROMless M68HC11 version and a PSD311, will need to be changed slightly if reconstructing the M68HC11 Ports B and C to the PSD311 Ports A and B as in this example.

The two eight-bit I/O port address locations are remapped from the M68HC711 64 byte register block area to the PSD311 chip-select I/O port base address (CSIOPORT) by using offsets from this base address. The tables below show the offset with the PSD311 base address of \$2000 appended to form the physical addresses as appropriate for this example.

M68HC11 Port B		Maps To		PSD311 Port A	
Register Name	Physical Location	Register Name	Physical Location	Register Name	Physical Location
DDRB	–	Direction	\$2004		
PORTB Data Write	\$1004	Data Write/Read	\$2006		
PORTB Pin Read	\$1004	Pin Read	\$2002		
M68HC11 Port C		Maps To		PSD311 Port B	
Register Name	Physical Location	Register Name	Physical Location	Register Name	Physical Location
DDRB	\$1007	Direction	\$2005		
PORTB Data Write	\$1003	Data Write/Read	\$2007		
PORTB Pin Read	\$1003	Pin Read	\$2003		

The code differences this translates to can be illustrated with the sample code for both the single chip M68HC711E9 and then for the twin chip M68HC11E1 and PSD311 solution as shown below.

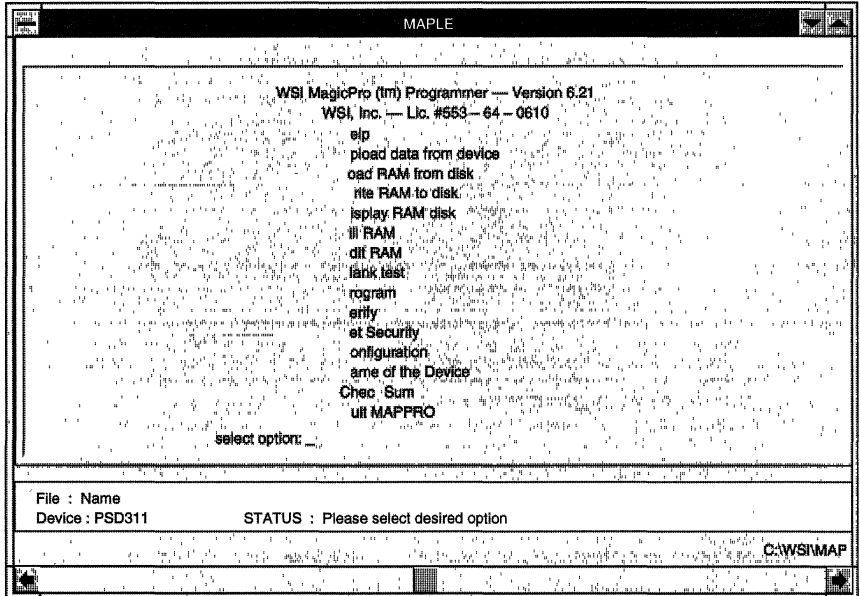
Single-Chip M68HC711E9 Code		
ORG	\$D000	
LDAA	#\$00	
STAA	\$1007	"Set M68HC11 Port C pins to inputs
STAA	\$1004	"Turn off all LED segments
LDAA	\$1003	"Read port C dip switches
STAA	\$1004	"Turn on or off the appropriate LED segments
BRA	\$D000	"Continue to read Port C and display on Port B

Twin-Chip Solution M68HC11E1 Code		
ORG	\$D000	
LDAA	#\$FF	" Note: M68HC11 Port B are outputs only
STAA	\$2004	"Set PSD311 Port A to all outputs
LDAA	#\$00	
STAA	\$2005	"Set PSD311 Port B for all inputs
STAA	\$2006	"Turn off all LED segments
LDAA	\$2003	"Read the PSD311 Port B pin register
STAA	\$2006	"Turn on/off the appropriate LEDs
BRA	\$D000	"Continue to read Port B and display on Port A

**Programming
The PSD311
Or PSD311R**

The PSD311 or PSD311R is programmed with the file that is generated during the COMPILER section of the PSD-SILVER MAPLE software. It usually has a *filename.OBJ* extension. The file is then loaded into programmer RAM on either a WSI MagicPro III PC-compatible programmer or an industry-standard programmer. Then the PSD device is ready to be programmed which is very similar to programming a standard EPROM or PLD. Figure 13 shows the menu for programming the PSD devices using the MAPPRO (F3) option from the MAIN menu.

Figure 13. MAPPRO Programming Software



Programming Manufacturers

Several of the programming manufacturers that support PSD devices are listed below. Many of their programmers have been officially qualified by WSI.

WSI (510) 656-5400 (800) 832-6974	Data I/O (800) 426-1045 (800) 247-5700	Advin Systems (408) 243-7000 (800) 627-2456
B&C Microsystems (408) 730-5511	BP Microsystems (713) 688-4600	Bytek (800) 523-1565
Link Computer (201) 808-8990	Logical Devices (303) 279-6868	Needham's Elec (916) 924-8037
SMS (206) 883-8447	Stag Microsystems (408) 988-1118	Sunrise (909) 595-7774
Systems General (408) 263-6667	Tribal Microsystems (510) 623-8860	

1

Conclusion

Implementing a two-chip solution with the M68HC11E1 and a PSD311 or PSD311R has been shown to be a simple process requiring very little code conversion or hardware modification. As demonstrated, this alternative to using a single-chip OTP M68HC711 or expanded mode multichip M68HC11 configuration offers flexible integration and can provide extra memory, logic, and I/O to further enhance your system capabilities.



PSD3XX Family

1

ZPSD3XX Family

2

PSD4XX/5XX Family

3

Motorola Application Notes

4

***Sales Representatives
and Distributors***

5

Section Index

<i>ZPSD3XX Family</i>	Application Note 034	ZPSD Power Consumption Calculations.....	2-1
------------------------------	----------------------	--	-----

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



Programmable Peripheral Application Note 034

ZPSD Power Consumption Calculations

By Yoram Cedar

Zero Power PSD Background

Portable and battery powered systems have recently become major embedded control application segments. As a result, the demand has increased dramatically for electronic components having extremely low power consumption. Recognizing this need, WSI, Inc. has developed a new ZPSD (Zero Power PSD) technology for use in low power programmable Microcontroller peripheral circuits.

ZPSD products virtually eliminate the DC component of power consumption reducing it to standby levels (μA). Eliminating the DC component is the basis for the words "Zero Power" in the ZPSD name. ZPSD products also minimize the AC power component when the logic is changing states by using address transition detection, array partitioning and DPTL (Differential Path Transistor Logic) design techniques. The result is a programmable microcontroller peripheral family that replaces memory, PLD and discrete logic functions while drawing much less power than a single EPROM.

2

Zero Power PSD Operation

Upon each address or logic input change to the ZPSD device, the internal logic powers up from low power standby for a very short time period. During this power up cycle, the ZPSD consumes only the necessary power to deliver new logic or memory data to its outputs as a response to the input change. After the new outputs are stable, the ZPSD latches them and automatically reverts to standby mode.

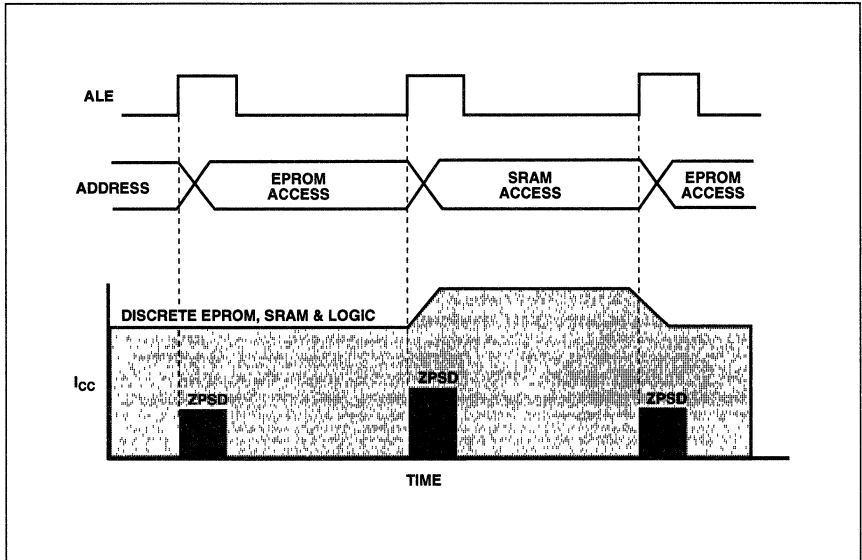
The I_{CC} current consumed during standby mode and during DC operation (chip enabled, no toggling) is identical and is only a few microamps (μA). The ZPSD automatically reduces its DC current drain to these low levels and does not require controlling by the CSI (Chip Select) input. Disabling the CSI unconditionally disables the MCU interface causing the PSD to power down independent of any transition on the microcontroller bus (address, data and control). In the ZPSD3XX family, disabling the CSI input will power down the entire device while in the ZPSD4XX/5XX products, the memory and address decoding PLD will power down and the GPLD/PPLD will consume power based on logic input changes in the system.

The ZPSD contains the first architecture to apply zero power techniques to memory circuit arrays as well as logic.

Figure 1 describes the operation of the ZPSD compared to the operation of a discrete solution. A standard microcontroller (MCU) bus cycle usually starts with the generation of an address and an ALE (or AS) pulse. The ZPSD detects the address transition and powers up internally. The ZPSD then latches the outputs of the PLD, EPROM and SRAM to the new values. After finishing this operation, the ZPSD then turns off its internal power and enters standby mode.

**Zero Power
PSD
Operation
(Cont.)**

Figure 1. ZPSD Power Operation



The ZPSD will remain in standby mode if the address does not change between bus cycles (for example, looping on a single address or during a Halt operation). The time taken for the entire operation is less than the ZPSD “access time” and much less than the MCU bus cycle (ALE to ALE). The only significant power consumption in the ZPSD occurs during AC operation and can be calculated using the ALE frequency.

An alternate system implementation using discrete EPROM, SRAM, PLD, latches, and other individual components will consume operating power during the entire bus cycle.

The ZPSD power consumption is controlled by the Turbo and Cmiser bits. Their operation is described in each ZPSD data sheet. In general, the Turbo bit controls the power consumption and speed of the ZPLD while the Cmiser bit controls the power consumption and speed of the EPROM and SRAM. Each ZPSD data sheet provides the power consumption of each mode of operation using the Turbo and Cmiser bits in the DC electrical characteristics section.

In addition, each PSD4XX and PSD5XX data sheet describes the operation of the Power Management Mode Registers (PMMR) that enable controlling the power consumption of various internal functional modules in those devices. For detailed explanation of the PSD4XX/5XX PMMR and APD operation please refer to Application Note 030.

Following are examples of the AC/DC Parameter tables and power consumption graphs of the ZPSD devices.

ZPSD3XX DC Characteristics – Commercial

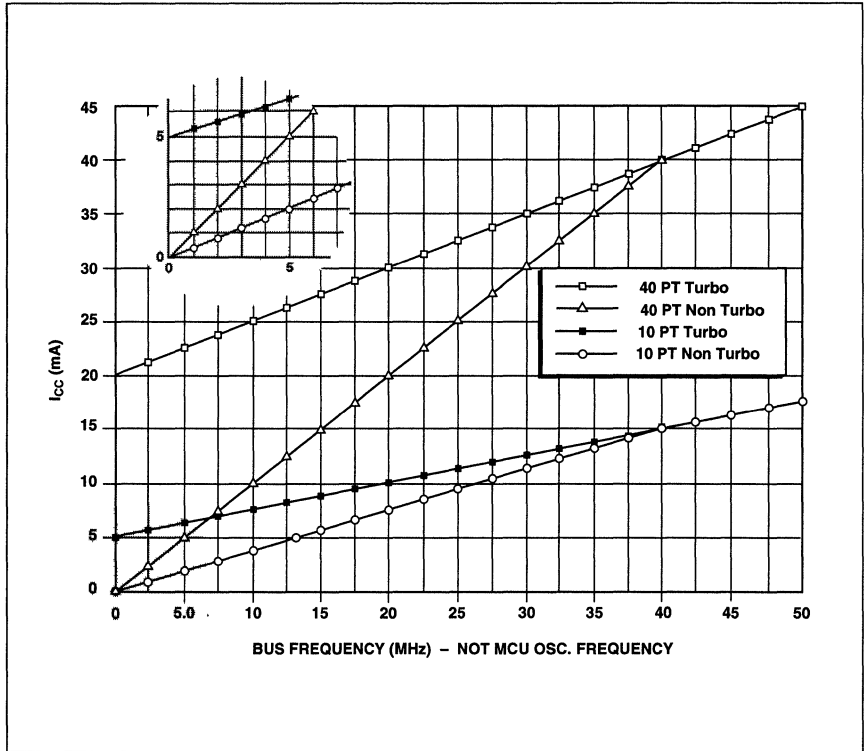
(5 V ± 10%)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V _{CC}	Supply Voltage	All Speeds	4.5	5	5.5	V
V _{IH}	High-Level Input Voltage	4.5 V < V _{CC} > 5.5 V	2		V _{CC} + .1	V
V _{IL}	Low-Level Input Voltage	4.5 V < V _{CC} > 5.5 V	-0.5		0.8	V
V _{OH}	Output High Voltage	I _{OH} = -20 μA, V _{CC} = 4.5 V	4.4	4.49		V
		I _{OH} = -2 mA, V _{CC} = 4.5 V	2.4	3.9		V
V _{OL}	Output Low Voltage	I _{OL} = 20 μA, V _{CC} = 4.5 V		0.01	0.1	V
		I _{OL} = 8 mA, V _{CC} = 4.5 V		0.15	0.45	V
I _{SB}	Standby Supply Current	$\overline{CS1} > V_{CC} - 3 V$		10	20	μA
I _{LI}	Input Leakage Current	V _{SS} < V _{IN} > V _{CC}	-1	±1	1	μA
I _{LO}	Output Leakage Current	.45 < V _{IN} > V _{CC}	-10	±5	10	μA
I _{CC} (DC)	Operating Supply Current	ZPLD_TURBO = OFF, f = 0 MHz		10	20	μA
		ZPLD_TURBO = ON, f = 0 MHz		0.5	1	mA/PT
I _{CC} (AC)	ZPLD AC Base	(See Figure 2)				mA/MHz
	EPROM Access AC Adder	CMiser = ON and 8-Bit Bus Mode		0.8	2.0	mA/MHz
		All Other Cases (Note 4)		1.8	4.0	mA/MHz
	SRAM Access AC Adder	CMiser = ON and 8-Bit Bus Mode		1.4	2.7	mA/MHz
		CMiser = ON and 16-Bit Bus Mode		2	4	mA/MHz
CMiser = OFF			3.8	7.5	mA/MHz	

- NOTES:**
1. CMOS inputs: GND ± 0.3 V or V_{CC} ± 0.3V.
 2. TTL inputs: V_{IL} ≤ 0.8 V, V_{IH} ≥ 2.0 V.
 3. $\overline{CS1}/A19$ is high and the part is in a power-down configuration mode.
 4. All other cases include CMiser = ON and 16-bit bus mode and CMiser = OFF and 8- or 16-bit bus mode.

**Zero Power
PSD
Operation
(Cont.)**

Figure 2. ZPSD3XX PAD I_{CC} vs. Frequency ($5\text{ V} \pm 10\%$)



ZPSD3XX DC Characteristics – Commercial**(ZPSD V Versions Only) (3 V ± 10%)**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V _{CC}	Supply Voltage	All Speeds	2.7	3	5.5	V
V _{IH}	High-Level Input Voltage	2.7 V < V _{CC} < 5.5 V	.7 V _{CC}		V _{CC} + .5	V
V _{IL}	Low-Level Input Voltage	2.7 V < V _{CC} < 5.5 V	-0.5		.3 V _{CC}	V
V _{OH}	Output High Voltage	I _{OH} = -20 μA, V _{CC} = 2.7 V	2.6	2.69		V
		I _{OH} = -1 mA, V _{CC} = 2.7 V	2.3	2.4		V
V _{OL}	Output Low Voltage	I _{OL} = 20 μA, V _{CC} = 2.7 V		0.01	0.1	V
		I _{OL} = 4 mA, V _{CC} = 2.7 V		0.15	0.45	V
I _{SB}	Standby Supply Current	CS1 > V _{CC} - .3 V (V _{CC} = 3.0 V)		1	5	μA
I _{LI}	Input Leakage Current	V _{IN} = V _{CC} or GND	-1	±.1	1	μA
I _{LO}	Output Leakage Current	V _{OUT} = V _{CC} or GND	-1	.1	1	μA
I _{CC} (DC)	Operating Supply Current	ZPLD_TURBO = OFF, f = 0 MHz (V _{CC} = 3.0 V)		1	5	μA
		ZPLD_TURBO = ON, f = 0 MHz (V _{CC} = 3.0 V)		.17	.35	mA/PT
I _{CC} (AC)	ZPLD AC Base	See Figure 3 (V _{CC} = 3.0 V)				mA/MHz
	EPROM Access AC Adder	CMiser = ON and 8-Bit Bus Mode (V _{CC} = 3.0 V)		0.4	1	mA/MHz
		All Other Cases (Note 8) (V _{CC} = 3.0 V)		0.9	1.7	mA/MHz
	SRAM Access AC Adder	CMiser = ON and 8-Bit Bus Mode (V _{CC} = 3.0 V)		0.7	1.4	mA/MHz
		CMiser = ON and 16-Bit Bus Mode (V _{CC} = 3.0 V)		1	2	mA/MHz
		CMiser = OFF (V _{CC} = 3.0 V)		1.9	3.8	mA/MHz

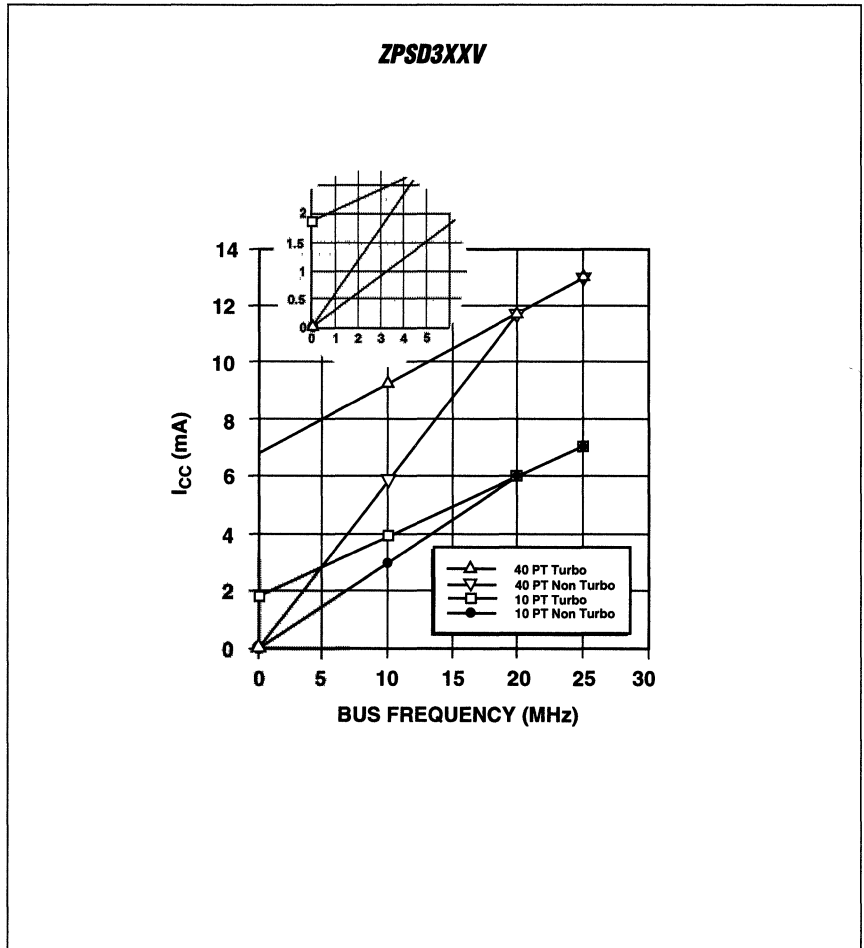
NOTES: 5. CMOS inputs: GND ± 0.3 V or V_{CC} ± 0.3V.6. TTL inputs: V_{IL} ≤ 0.8 V, V_{IH} ≥ 2.0 V.

7. CS1/A19 is high and the part is in a power-down configuration mode.

8. All other cases include CMiser = ON and 16-bit bus mode and CMiser = OFF and 8- or 16-bit bus mode.

**Zero Power
PSD
Operation
(Cont.)**

Figure 3. Typical PLD AC I_{CC} Curve ($V_{CC} = 3.0\text{ V}$)



ZPSD4XX/5XX DC Characteristics

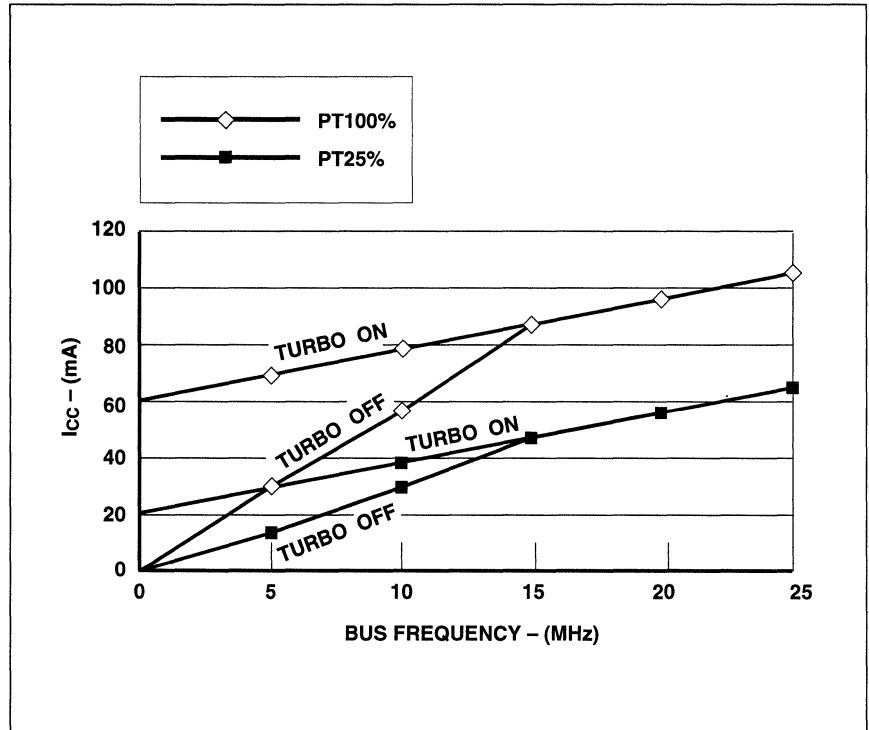
(5 V ± 10%)

Symbol	Parameter		Conditions	Min	Typ	Max	Unit
V _{CC}	Supply Voltage		All Speeds	4.5	5	5.5	V
V _{IH}	High Level Input Voltage		4.5 V < V _{CC} < 5.5 V	2		V _{CC} + 5	V
V _{IL}	Low Level Input Voltage		4.5 V < V _{CC} < 5.5 V	-0.5		0.8	V
V _{IH1}	Reset High Level Input Voltage		(Note 1)	.8 V _{CC}		V _{CC} + 5	V
V _{IL1}	Reset Low Level Input Voltage		(Note 1)	-5		.2 V _{CC} - 1	V
V _{HYS}	Reset Pin Hysteresis			0.3			V
V _{OL}	Output Low Voltage		I _{OL} = 20 μA, V _{CC} = 4.5 V		0.01	0.1	V
			I _{OL} = 8 mA, V _{CC} = 4.5 V		0.15	0.45	V
V _{OH}	Output High Voltage		I _{OH} = -20 μA, V _{CC} = 4.5 V	4.4	4.49		V
			I _{OH} = -2 mA, V _{CC} = 4.5 V	2.4	3.9		V
V _{SBY}	SRAM Standby Voltage			2.7		V _{CC}	V
I _{SBY}	SRAM Standby Current		V _{CC} = 0 V		0.5	1	μA
I _{IDLE}	Idle Current (V _{STBY} Pin)		V _{CC} > V _{SBY}	-0.1		0.1	μA
V _{DF}	SRAM Data Retention Voltage		Only on V _{STBY}	2			V
I _{SB}	Standby Supply Current	Power Down Mode	$\overline{CS1} > V_{CC} - 3 \text{ V}$ (Note 10)		25	50	μA
		Sleep Mode	$\overline{CS1} > V_{CC} - 3 \text{ V}$ (Note 11)		10	20	μA
I _{LI}	Input Leakage Current		V _{SS} < V _{IN} < V _{CC}	-1	±1	1	μA
I _{LO}	Output Leakage Current		0.45 < V _{IN} < V _{CC}	-10	±5	10	μA
I _{CC} (DC)	Operating Supply Current	ZPLD Only	ZPLD_TURBO = OFF, f = 0 MHz (Note 12)				
			ZPLD_TURBO = ON, f = 0 MHz		400	700	μA/PT
I _{CC} (AC)	ZPLD AC Base		(Note 12)				
	EPROM AC Adder	CMiser = ON (8-Bit Bus Mode)			0.8	2	mA/MHz
		All Other Cases			1.8	4	mA/MHz
	SRAM AC Adder	CMiser = ON and 8-Bit Bus Mode			1.4	2.7	mA/MHz
		CMiser = ON and 16-Bit Bus MoDe			2	4	mA/MHz
CMiser = OFF			3.8	7.5	mA/MHz		

- NOTES:** 9. Reset input has hysteresis. V_{IL1} is valid at or below .2V_{CC} - 1. V_{IH1} is valid at or above .8V_{CC}.
 10. CS1 deselected or internal PD is active.
 11. Sleep mode bit is set and internal PD is active.
 12. See Figure 4 for details.

**Zero Power
PSD
Operation
(Cont.)**

Figure 4. ZPSD4XX/5XX ZPLD I_{CC} vs. Frequency (5 V \pm 10%)



Power Consumption Calculation Definitions

$I_{SB} = I_{CC} (DC)$	= Power Consumption when system is in Standby (idle) mode or system is operating but none of the ZPSD inputs are changing.
$I_{CC} (AC)$	= Power Consumption when system is operating and ZPSD inputs are changing. = ZPLD AC Base @ f_{ZPLD} (I_{CC} vs. frequency as a function of PT) + % of EPROM Access X EPROM AC Adder X $f_{MCU\ BUS}$ + % of SRAM Access X SRAM AC Adder X $f_{MCU\ BUS}$ + Timer AC Adder X f_{timer} (only applicable for the ZPSD5XX)
ZPLD AC Base @ f_{ZPLD}	= ZPLD I_{CC} taken from the graph of the PLD AC curve of I_{CC} Vs. Frequency @ f_{ZPLD} . Based on the number of product terms (PT) used and if the Turbo bit is on or off. The PT number is shown in the PSDsoft fitting report.
EPROM AC Adder	= The power consumption of the EPROM as a function of frequency.
SRAM AC Adder	= The power consumption of the SRAM as a function of frequency.
Timer AC Adder	= The power consumption of the Timer as a function of frequency
% of EPROM Access	= Percent of time the MCU is accessing the EPROM
% of SRAM Access	= Percent of time the MCU is accessing the SRAM
f_{ZPLD}	= Maximum ZPLD input frequency
$f_{MCU\ BUS}$	= MCU bus frequency, usually the ALE (or AS) frequency
f_{timer}	= Maximum Timer input frequency

**Typical
Operating
Power
Calculation
Example of
ZPSD3XX at
V_{CC} = 5.0 V**

This example is based on the ZPSD3XX data sheet of June, 1995. Please review the specification of the particular ZPSD device you are using before calculating actual power consumption in your design.

Example Criteria

- f_{ZPLD} = f_{MCU BUS} 2 Mhz
- % of EPROM Access 80 %
- % of SRAM Access 15 %
- % of I/O Access 5 % (No additional power above the ZPLD AC Base)

- Number of ZPAD Product Terms Used 10 PT

- 8 Bit Data Bus

 CMiser On

 Turbo Off

- EPROM/SRAM access AC mA/MHz adder is found in 5 Volt DC Characteristics in the ZPSD3XX data sheet.

I_{SB} = I_{CC} (DC) = 10 µA

I_{CC} (AC) = ZPLD AC Base @ 2 MHz (see Figure 18 in ZPSD3XX data sheet) (.75mA)
 + % of EPROM access x EPROM AC Adder x 2 MHz
 (+ 80% x 0.8 mA/MHz x 2 MHz)
 + % of SRAM access x SRAM AC Adder x 2 MHz
 (+ 15% x 1.4 mA/MHz x 2 MHz)
= 2.45 mA

**Example of
ZPSD3XXV
Typical
Operating
Power
Calculations at
V_{CC} = 2.7 V**

This example is based on the ZPSD3XX data sheet of June, 1995. Please review the specification of the particular ZPSD device you are using before calculating actual power consumption in your design.

- f_{ZPLD} = f_{MCU BUS} 1 MHz
- % of EPROM Access 80 %
- % of SRAM Access 15 %
- % of I/O Access 5 % (No additional power above the ZPLD AC Base)

- Number of ZPAD Product Terms Used 10 PT

- 8 Bit Data Bus

 CMiser On

 Turbo Off

- EPROM/SRAM access AC mA/MHz adder is found in 3 Volt DC Characteristics in the ZPSD3XX data sheet.

I_{SB} = I_{CC} (DC) = 1 µA

I_{CC} (AC) = ZPLD AC Base @ 1 MHz (Figure 19 in ZPSD3XX data sheet) (0.30 mA)
 + % of EPROM access x EPROM AC Adder x 1 MHz
 (+ 80% x 0.4 mA/MHz x 1 MHz)
 + % of SRAM access x SRAM AC Adder x 1 MHz
 (+ 15% x 0.7 mA/MHz x 1 MHz)
= 0.77 mA x 0.9 (Normalized I_{CC}, See Figure 21) = 693 microamps



PSD3XX Family

1

ZPSD3XX Family

2

PSD4XX/5XX Family

3

Motorola Application Notes

4

***Sales Representatives
and Distributors***

5

Section Index

PSD4XX/5XX Family

Application Note 028	PSD5XX Counter/Timers Operation.....	3-1
Application Note 029	Interfacing PSD4XX/5XX To Microcontrollers	3-73
Application Note 030	PSD4XX/5XX Power Calculations and Reduction	3-145
Application Note 031	PSD4XX/5XX Design Tutorial	3-161
Application Note 033	Keypad Interface to PSD4XX/5XX with Autoscanning	3-245
Application 035	How To Design With The PSD4XX/5XX ZPLD	3-257
Application 036	How To Fit Your Design Into The PSD4XX/5XX	3-265
Application 037	How to Implement a Latch Function in Port A of PSD4XX/5XX that is Independent of the System Clock.....	3-271
Application 038	How to Increase the Speed of the PSD5XX Counter/Timers.....	3-277
Application 039	Encoder for Shaft Direction and Position Recognition Using the PSD5XX.....	3-287
Application 042	Four Axis Stepper Motor Control Using a Programmable PSD5XX MCU Peripheral from WSI, Inc.	3-297

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



Programmable Peripheral Application Note 028 PSD5XX Counter/Timers Operation

By Ravi Kumar

Abstract

This application note explains the operation and programming of Counter/Timers on WSI's PSD5XX Family of Field-Programmable Microcontroller Peripherals.

Initializations required to implement each of the five modes of operation of Counter/Timers are explained. Refer to appendices for all the relevant files.

Introduction

The PSD5XX on-chip Counter/Timers provide additional Timer functions to a Microcontroller.

A typical Microcontroller or a Timer Peripheral chip usually has a set of Timers controlled by either

- External Pins
- Software

The PSD5XX has four identical 16-bit Counter/Timers. Each Counter/Timer is controlled by either

- PPLD* outputs
- External Pins
- Software

Figure 1 shows the I/O pins and the functional block of the Counter/Timers.

The PPLD

The Peripheral Programmable Logic Device (PPLD) provides a powerful mechanism for the user to control the operations of the Counter/Timers and the Interrupt Controller. There are six Peripheral Macrocells in the PPLD, four are dedicated to the Counter/Timers, and two to the Interrupt Controller. Figure 2 shows a PPLD macrocell for the Counter/Timers.

The PSD5XX Counter/Timers have the following features:

- Five Modes of operation
 - **Waveform Mode**
 - **Pulse Mode**
 - **Event Counter Mode**
 - **Time Capture Mode**
 - **Watchdog Mode**

- Each Counter/Timer can be controlled by an input pin, dedicated PPLD macrocell or software.
- The Watchdog output is routed through the ZPLD and can be programmed to output at any PSD output pin.
- Programmable polarity for input control and Timer output.
- Can be programmed to be UP or DOWN Counters.
- Input clock to all Counter/Timers can be from DC to 7.0 MHz. Higher resolution can be achieved by using in conjunction with the GPLD macrocells.
- High resolution Divisor unit to scale down the Counter/Timer input clock.
- Can easily interface with any 8 or 16-bit Microcontroller.
- Terminal Count of each Counter/Timer can be configured as interrupt input to the Interrupt Controller.

3

*Refer to the section "ZPLD Block" in the PSD5XX data sheet.

Figure 1. Counter/Timer Interface With Other Internal Blocks in PSD5XX

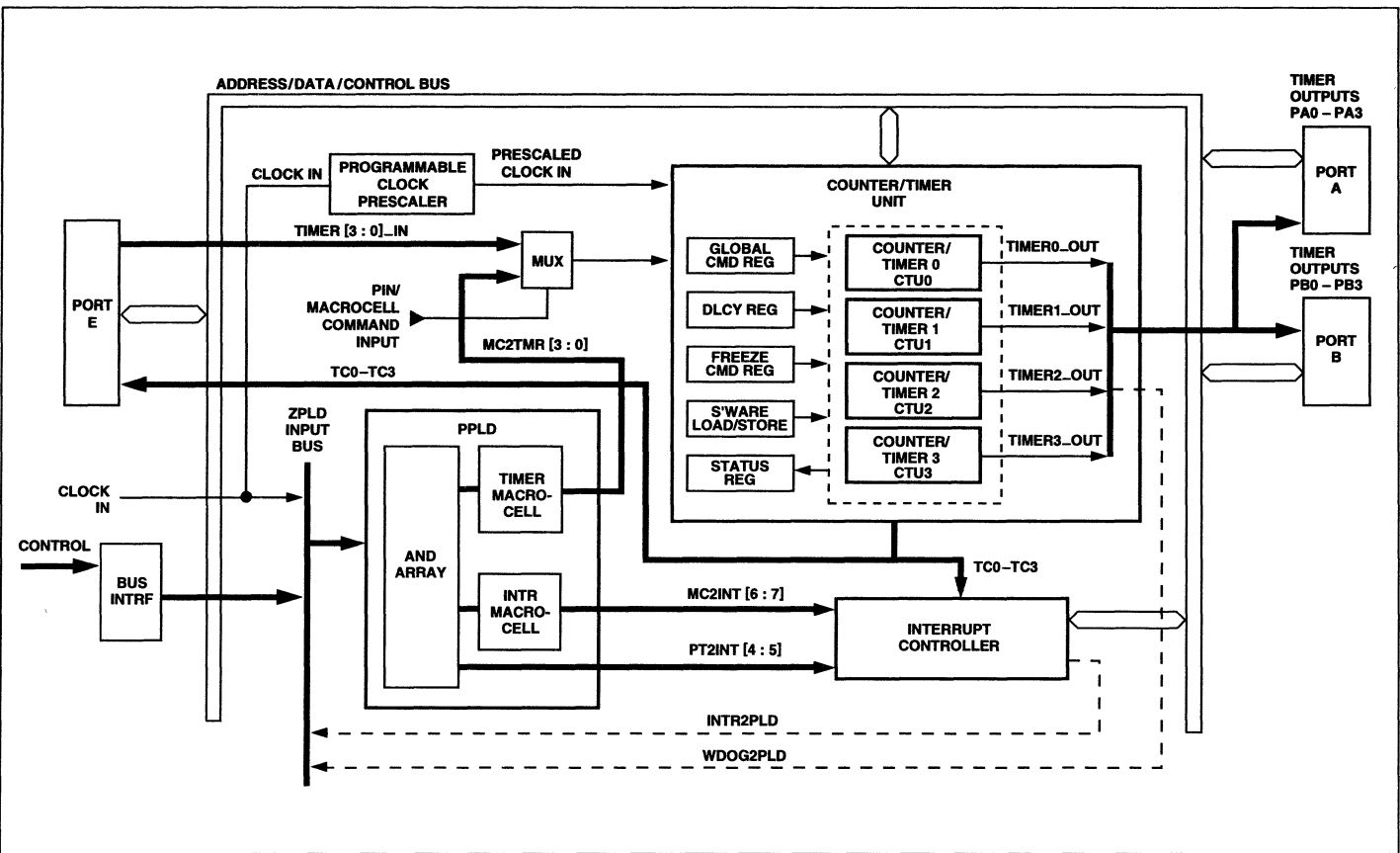
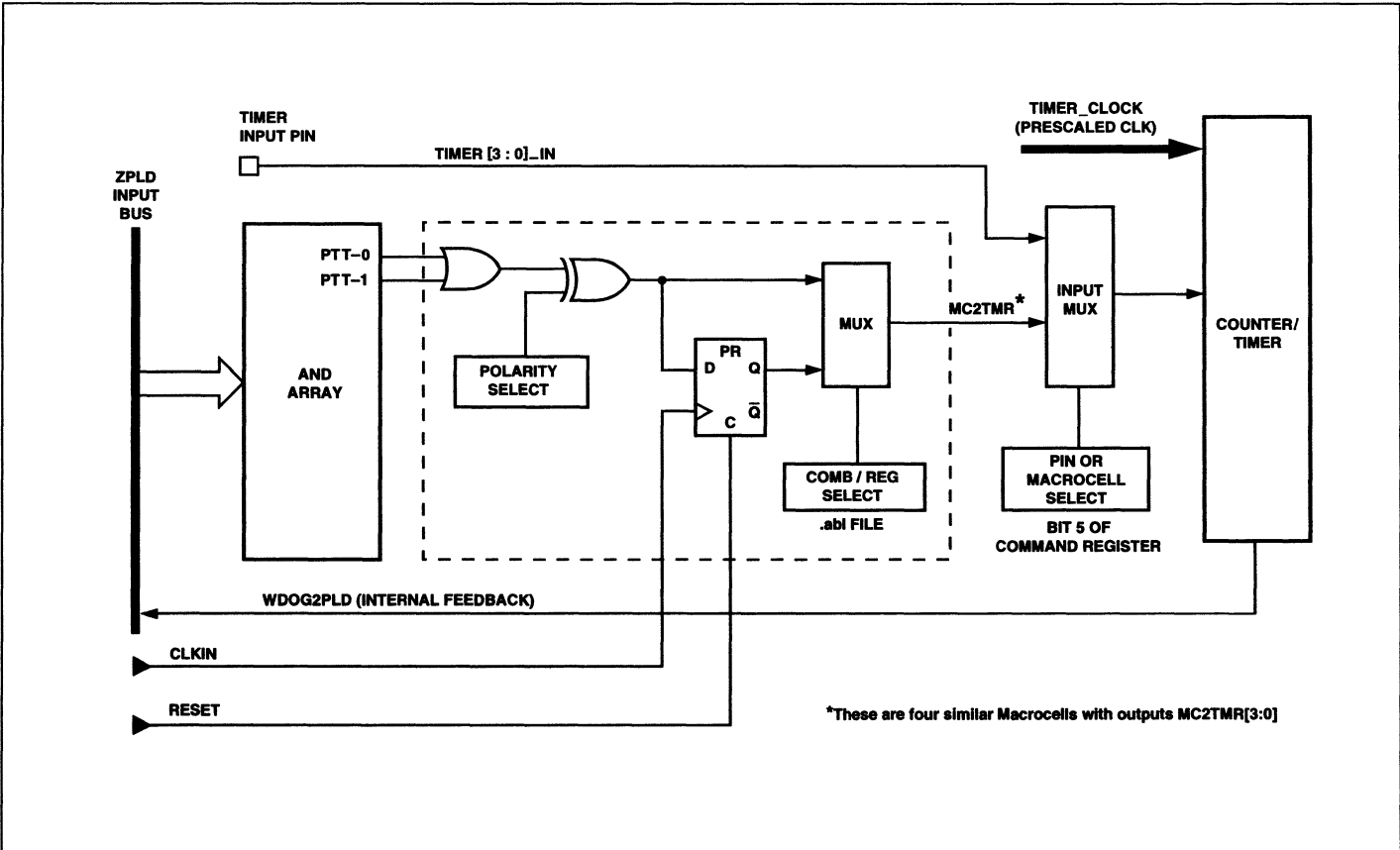


Figure 2. PPLD Macrocell For Each Counter/Timer



*These are four similar Macrocells with outputs MC2TMR[3:0]

Introduction
(Cont.)

The PPLD (Cont.)

The Operation of a Counter/Timer Unit

The basic functional block of a Counter/Timer Unit (CTU) is shown in Figure 4. It consists of a 16-bit up/down Counter and a 16-bit Image Register. The Counter performs a counting operation such as generating a waveform output or counting the event occurrence of an input signal. The Image Register serves as an interface register for the Counter. For example, in Pulse Mode the length of the pulse width is stored in the Image Register. When activated, the Counter is loaded with the contents of the Image Register and generates a pulse output with duration defined by the Image Register. But in the Event Count Mode (or Time Capture Mode), the number of event count (content of Counter) is stored in the Image Register so that it can be read by the Microcontroller.

Both the Image Register and Counter can be accessed by the Microcontroller. Usually the Counter is accessed only for initialization purposes. To access the Image Register when the Counter is running, you need to first freeze the Image Register via the Freeze Command Register and wait for the freeze acknowledge bit by reading the Status Flags Register. If the bit is set, you can then proceed to access the Image Register.

The Counter generates an output if it is configured in Waveform, Pulse or Watchdog Mode. The Waveform/Pulse output can be routed to an output pin on Port A or B. For Watchdog output, the signal must go through the GPLD before it is routed to any selected output pin. For Event Count or Time Capture Mode, the output of the counter is read by the user from the Image Register.

Figure 3 is the PSD5XX equivalent block diagram depicting the basic input and output signals of each Counter/Timer unit.

Figure 3. PSD5XX Counter/Timer Equivalent Block Diagram

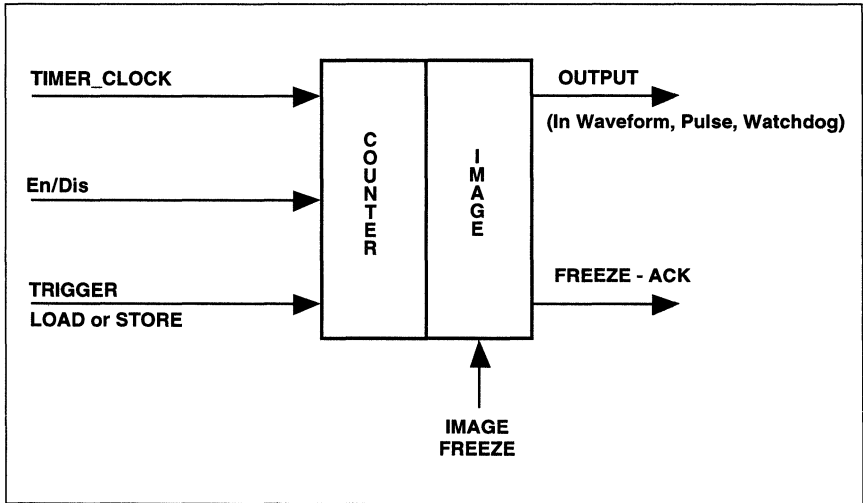
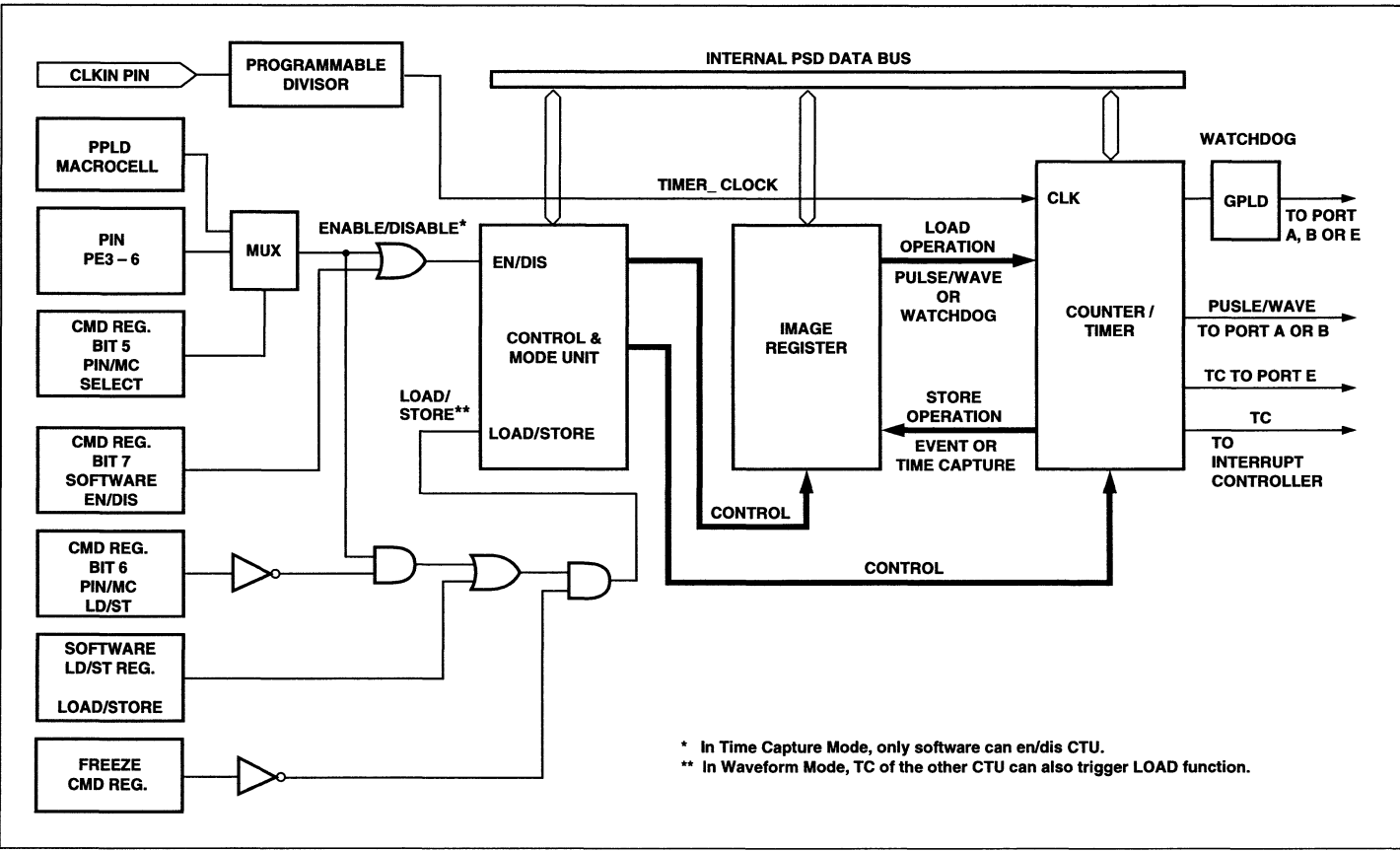


Figure 4. Simplified Block Diagram of Counter/Timers Control Signals



* In Time Capture Mode, only software can en/dis CTU.
 ** In Waveform Mode, TC of the other CTU can also trigger LOAD function.



Introduction

(Cont.)

The PPLD (Cont.)

The operation of the CTU is controlled by two signals:

- The en/dis signal –**
To enable or disable the Counter from counting.
- The load/store signal –**
To load the Image Register contents to the Counter or store the Counter value to the Image Register.

These two signals are defined by the user through the Counter Command Register, the external pin input, the PPLD macrocell output or by the user software. These multiple sources of control enable the user to implement very specific counter/timer applications. Table 1 shows the sources for the load/store operation and the enable/disable function under different modes of operation, and also what the Counter is doing while the Image Register is under freeze.

Table 1. Modes of Operation

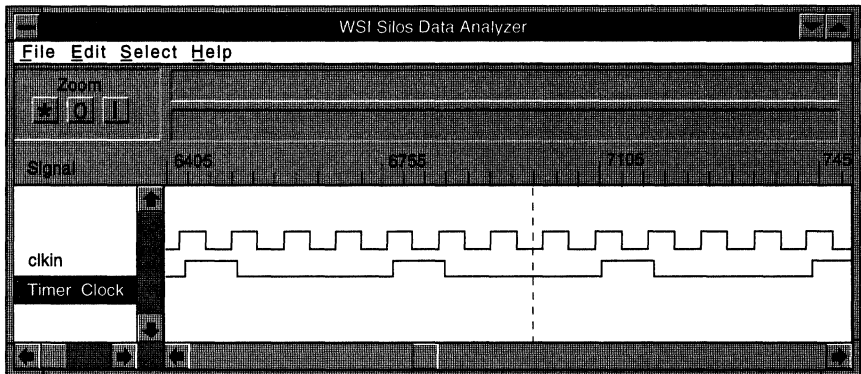
Input Mode of Operation	Possible Load/Store Sources	Load/Store Function	Possible Enable/Disable Sources	Counter During Freeze-Ack
Waveform	Software, pin, PPLD Macrocell, TC of the other counter.	Load counter from Image Register.	Software, pin, PPLD Macrocell.	Continue to count. When TC is reached output level is unchanged.
Pulse	Software, pin, PPLD Macrocell.	Load counter from Image Register.	Software, pin, PPLD Macrocell.	Continue to count. When TC is reached output level changes.
Watch-Dog (Counter-2 only)	Software.	Load counter from Image Register.	Always enabled.	No effect.
Event Count	Software, pin, PPLD Macrocell.	Store counter value in the Image Register.	Pin, PPLD Macrocell. Every specified transition will increment (or decrement) counter.	Counter will continue to count events.
Time capture	Software, pin, PPLD Macrocell.	Store counter value in the Image Register.	Software.	Counter will continue to count timer clock cycles.

PSD5XX Counter/Timers INPUT/CLOCK Scaling

All four Counter/Timers share a common input clock which can be scaled down. The Counter/Timers operate in the frequency range up to 7.0 MHz. The maximum input clock to the PSD5XX is 28 MHz. The default divide factor is 4 and the input clock can further be scaled down through 280 times.

Figure 5 depicts the relationship between PSD5XX clock input (clkin) and the Timer_Clock with the default divide factor 4.

Figure 5.

**3**

PSD5XX
Counter/Timers
INPUT/CLOCK
Scaling
(Cont.)

The following example has been used for this application note.

External input clock of the PSD5XX is 12 MHz.

The expected Count frequency of all Counter/Timers is 3 MHz.

$$\text{Counter/Timer Clock Input} = \frac{(\text{External Clock Input to PSD5XX})}{(\text{DIV})} \dots (1)$$

The range of DIV is $4 = < \text{DIV} = < 280$

Based on the value of DIV the Scale-bit in the Global Command Register and DLCY value in the DLCY Register are loaded.

DLCY is the value loaded into the DLCY Register and represents the number of delay cycles, the range is $0 = < \text{DLCY} = < 31$

The Scale-bit in the Global Command Register when set to

- 0: The clock to all Counter/Timers is divided by 1.
- 1: The clock to all Counter/Timers is divided by 8.

$$\begin{aligned} \text{Therefore from (1) DIV} &= \frac{(\text{External Clock Input to PSD5XX})}{\text{Counter/Timer Clock Input}} \\ &= \frac{12\text{MHz}}{3\text{MHz}} \\ \Rightarrow \text{DIV} &= 4 \dots (2) \end{aligned}$$

Hence from the data sheet Table 16, when DIV = 4, set the Scale-bit in the Global Command Register to 0 and write 00 into the DLCY Register. Note that at power up DLCY contains 00.

The input clock to the Counter/Timers is then 3 MHz, scaled down from the external 12 MHz clock.

Different Operating Modes

The operations and initializations of the five modes of the Counter/Timers are discussed in the following sections.

Waveform Mode

The Waveform Mode is also known as the Pulse Width Modulation (PWM) Mode. In this mode a continuous waveform output is produced using two Counter/Timers. The on and off widths of the output Waveform are programmable and are defined by the user, by writing the desired values into the corresponding Image Registers.

Figure 6 shows a typical PSD5XX based PWM implementation, where the PWM_OUT is the expected output waveform, and Timer Clock is the clock input to the CTUs. As seen in Figure 6 a minimum of two Counter/Timers are required to implement the Waveform mode. There is an alternate implementation of the PWM mode, where each of the four Counter/Timers can generate PWM waveforms. Refer to Appendix 6 for details.

Typical applications of this mode are:

- Automotive engine control
- Motor speed control
- Display intensity control
- Sound generation

A waveform output in the above application can easily be produced using PSD5XX Counter/Timers. Variations of on-time/off-time of the waveform output is done by modifying the Image Register contents. The relationship between on-time and off-time of the waveform output is expressed as duty cycle.

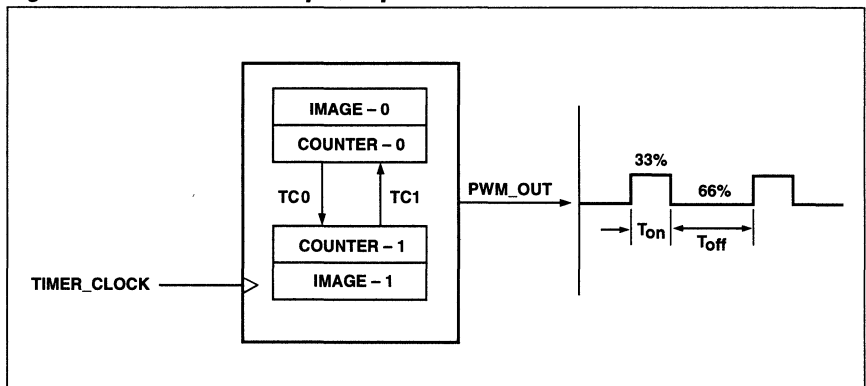
The duty cycle of the waveform in Figure 6 is expressed as:

$$\text{Duty Cycle} = \frac{\text{on-time (Ton)}}{\text{on-time (Ton) + off-time (Toff)}}$$

If Ton equals Toff then the waveform output has a 50% duty cycle and is a square wave. Suppose PWM is used to increase or decrease the power supply to a motor, with the larger duty cycle delivering more power to the motor. Obviously more power to the motor means higher speed in the motor, i.e., motor speed can be controlled by adjusting the ON time of the signal.

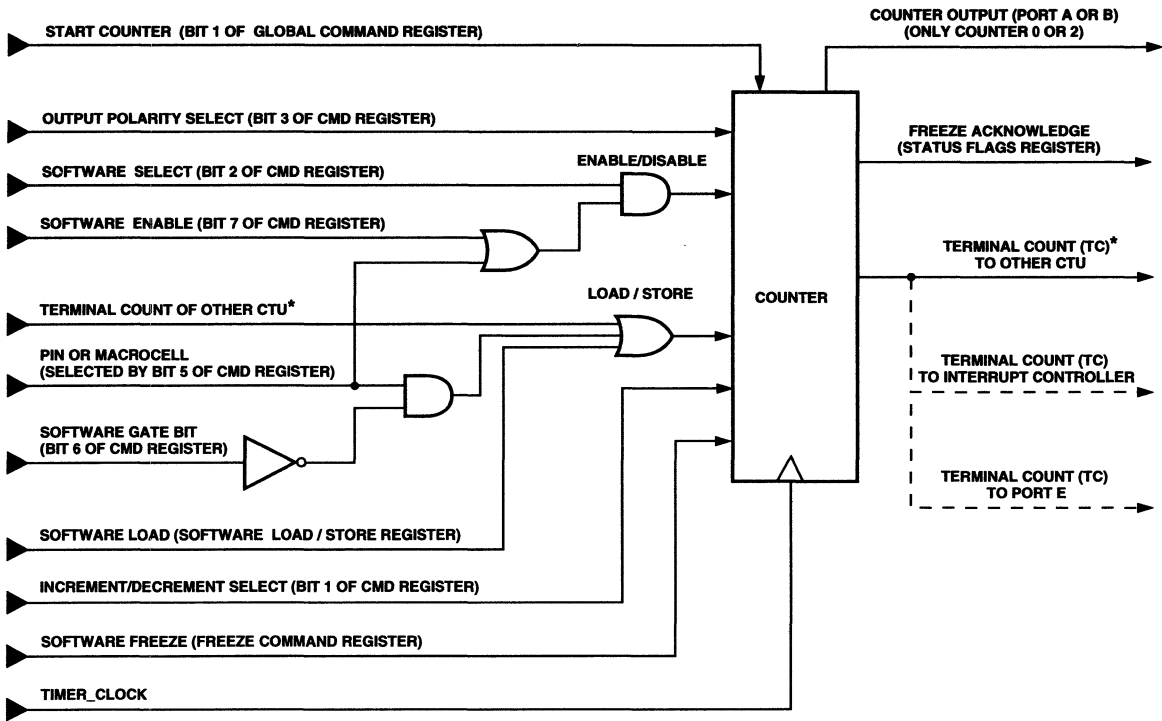
Figure 7.0 depicts the input control signals of PSD5XX in waveform mode.

Figure 6. Waveform Mode Input/Output



Different Operating Modes (Cont.)

Figure 7. CTU Control Signals for Waveform Mode



*Need two CTUs together in Waveform Mode (CTU0 – CTU1 or CTU2 – CTU3).
 The Terminal Count of CTU0 drives CTU1 and the Terminal Count of CTU1 drives CTU0.
 The same applies to CTU2 and CTU3.



Different Operating Modes (Cont.)

Waveform Mode (Cont.)

The features of the Waveform Mode are:

- ❑ Two Timers configured in Waveform Mode are needed to generate a continuous Waveform output of the Image Registered duty cycle. The combinations of the two Counter/Timers are:
 - Counter/Timer-0 and Counter/Timer-1
and/or
 - Counter/Timer-2 and Counter/Timer-3
- ❑ Image Registers of Counter/Timer-0 and Counter/Timer-1 are loaded with proper count values to generate the required Registered duty cycle and duration. The frequency of the waveform is : $1/T(\text{on}) + T(\text{off})$.
- ❑ There are four different sources which can load the Counter with contents from the Image Register:
 - by software
 - by input pin
 - by PPLD macrocell output
 - by Terminal Count (TC) of the other Counter configured in the Waveform Mode.
- ❑ Three different sources are available to enable or disable the Counter:
 - by software
 - by input pin
 - by PPLD macrocell output
- ❑ Outputs of these Counter/Timers are available on Port A or Port B. The fitter reports contain the pin list information. The Counter/Timer outputs are routed to the corresponding pins via software.
- ❑ If required the outputs can be fed back to the GPLD through the I/O ports. This feedback enables the creation of complex waveforms.

A Waveform Mode Design example:

- ❑ In this application example a waveform output (PWM_OUT) of 33.33% duty cycle is generated. The required Ton time is 666 ns while the Toff time is 1332 ns.
- ❑ The Waveform period time is 2000 ns.
- ❑ Counter/Timer-0 is configured to generate the Toff pulse and Counter/Timer-1 is configured to generate the Ton pulse. The Counters are enabled via software. The loading of the Counters from the Image Registers are automatically triggered by the TC of the other Counter. No inputs from the pin or macrocell are used.
- ❑ The Timer input clock is 3MHz, i.e., PSD5XX input clock of 12 MHz is scaled down to 3 MHz. To achieve the desired Ton/Toff time, the Image Register-0 is initialized with a count value of 2 and Image Register-1 with a count value of 4.

Different Operating Modes (Cont.)

Waveform Mode (Cont.)

The program flow to set up the Waveform Mode operation as described in this example is as follows:

1. Define the Timer input clock frequency (see section on clock scaling).
2. Set up Command Registers for CTU0 and CTU1 (CTU is an abbreviation of Counter/Timer unit).
3. Initialize the Image Registers with the proper count values to define Ton/Toff time.
4. Specify the Timer pulse output. This can be done in two ways:
 via .abl as PWM_OUT pin 27, which is a user defined name or
 via PSDconfiguration software by specifying “waveform/pulse output”, which assigns the default name “timerout0”.
5. Set up the Special Function Register to specify the pin which is to be used as the output pin for the signal PWM_OUT.
6. Set up the Software Load/Store Register to load the CTUs with the initial count values.
7. Set up the Global Command Register Start Bit and start the operation of the CTUs.

The procedure to set up Counter/Timer Registers in the Waveform Mode in this design example is:

Counter/Timer-0 Registers Initialization:

- Write “D4”hex to Command Register-0 (CMD0) at offset from base address of CSIOP (Chip Select I/O Port).

CMD0 Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
1	1	X	X	0	1	0	0

- Bit-0:** 0 Mode Select Bit, select Waveform Mode for CTU0.
- Bit-1:** 0 Decrement/Increment Bit: select decrement (CTU0 counts down from 2 to 0. At 0, TC triggers the loading and operation of the CTU1).
- Bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- Bit-3:** 0 Output polarity: Select output to be active low (Toff time).
- Bit-4:** X Input Polarity: No pin input in this mode, don't care.
- Bit-5:** X Pin or Macrocell input: No pin or macrocell input, don't care.
- Bit-6:** 1 Load/Store Bit: No pin or macrocell load/store.
- Bit-7:** 1 EN/DIS Bit: Enable continuous counting.

IMG0 is loaded with 04 (hex) to define the Toff time of the output pulse (PWM_OUT)



Different Operating Modes (Cont.)

Waveform Mode (Cont.)

Counter/Timer-1 Registers Initialization:

- Write “CC”hex to Command Register 1 (CMD1).

CMD1 Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
1	1	X	X	1	1	0	0

Bit-0: 0 Mode Select Bit, select Waveform Mode for CTU1.

Bit-1: 0 Decrement/Increment Bit: select decrement (CTU1 counts down from 4 to 0. At 0, TC triggers the loading and operation of the CTU0).

Bit-2: 1 Select Counter/Timer Bit: Select CTU1.

Bit-3: 1 Output polarity: Select output to be active on (Ton time).

Bit-4: X Input Polarity: No pin input in this mode, don't care.

Bit-5: X Pin or Macrocell input: No pin or macrocell input, don't care.

Bit-6: 1 Load/Store Bit: No pin or macrocell load/store.

Bit-7: 1 EN/DIS Bit: Enable continuous counting.

IMG1 is loaded with 02(hex) to define the Ton time of the output pulse (PWM_OUT)

After Command Registers 0 and 1 are initialized, other Registers (Special Function Register, Software Load/Store Register and Global Command Register) must now be initialized.

- Configure Port A pin PA0 as special function out, dedicating it as a Timer output pin by setting bit-0 to “1” in Port A Special Function Register. This bit is set only when there is a need to bring the timer output pulse out of the PSD5XX. Refer to .frp report file to determine where the output is connected (The device fitter might assign it to Port B pin PB0).

Special Function Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	0	1

Different Operating Modes (Cont.)

Waveform Mode (Cont.)

- Set Load/Store bit-0 and bit-1 (of Timers 0 and 1) to one in the Software Load/Store Register. This transfers the content of the Image Registers to the Timers to initialize the waveform mode. The Software Load/Store bits are automatically cleared whenever the Counter/Timer starts operating.

Software Load/Store Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	1	1

- Now to start the Timers: The Global Command Register has to be initialized to 02(hex), i.e., the following bits are set

Global Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	1	0

Bit-0: 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).

Bit-1: 1 Counter start bit: This bit turns on all the selected Timers.

Bit-2: 0 Global Mode bit: All Counter/Timers operate in Waveform or Pulse Mode.

Bit-3: 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer-2).

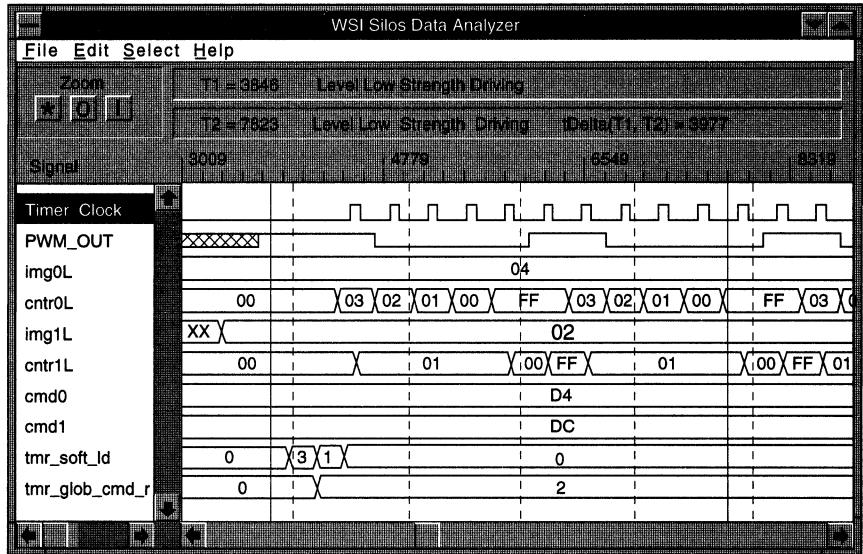
After the Counter/Timers start operating, every time a Timer counts to zero, a transition occurs on the output waveform (PWM_OUT) to generate a pulse with the specified duty cycle. The Counter/timer-0 is reloaded automatically from the Image Register-0 when the Counter/timer-1 reaches the zero count and this process repeats with Counter/timer-1. The terminal count of each Counter/timer drives the loading of the other Counter/timer. This results in a continuous waveform output.

Different Operating Modes (Cont.)

Waveform Mode (Cont.)

Figure 8 shows the simulation result of the Waveform Mode simulated on the PSDsim simulator.

Figure 8. Simulation of Waveform Mode



3

Input Signals:

Prescaled PSD5XX input clock: **Timer_Clock**

Output Signal:

PWM waveform output: **PWM_OUT**

The Counter/Timers are enabled by software and as soon as the Global Counter Start bit is set to 1, the Timers start to output PWM waveform. The total period is the sum of the count values loaded into the IMG0 and IMG1 Registers (04 + 02 = 06). The duty cycle is (02/06 = 0.33) or 33%. Note that every time the contents of the Image Registers are changed to vary the duty cycle of the output waveform, the Load/Store bits of both the Counter/Timers must be set to 1 to initialize a new PWM cycle.

Different Operating Modes (Cont.)

The Pulse Mode

In Pulse Mode a Counter/Timer when enabled outputs a mono-shot pulse. The pulse width is defined by the value loaded into the corresponding Image Register. Any of the four Counter/Timers are capable of Pulse Mode. Figure 9 depicts a pulse output from Counter/Timer-0 initiated by the PPLD input control signal mc2tmr0.

A typical application of the Pulse Mode is in networking applications using CSMA/CA protocol. The transmission line has to be sensed to check if other stations are accessing this line. Therefore, based on a signal transition on this line a mono-shot pulse has to be produced to indicate that the line is busy. This mono-shot pulse stops the host station from accessing the line and hence avoids data collisions.

- ❑ Up to four pulse outputs are available from the PSD5XX, one per each Counter/Timer.
- ❑ Polarity of the pulse output is defined by the output polarity bit in the Counter/Timer Command Register.
- ❑ To generate the required pulse width, load the Image Register with the required pulse width value. As soon as the Counter/Timer trigger occurs, the Image Register contents are transferred to the corresponding Counter/Timer.
- ❑ Unlike Timers on standard microcontrollers, there are three different ways to enable a Counter/Timer on the PSD5XX:
 - Input pins PE3-PE6(port E).
 - mc2tmr0-3 inputs in the PPLD.
 - Software Control.
- ❑ The outputs of the Counter/Timers are available on Port A or Port B. The outputs can be fed back to the ZPLD. Refer to the .frp report file to determine where the outputs are connected.

Figure 10 depicts the input control signals of a PSD5XX in Pulse mode.

Figure 9. Pulse Mode Input/Output

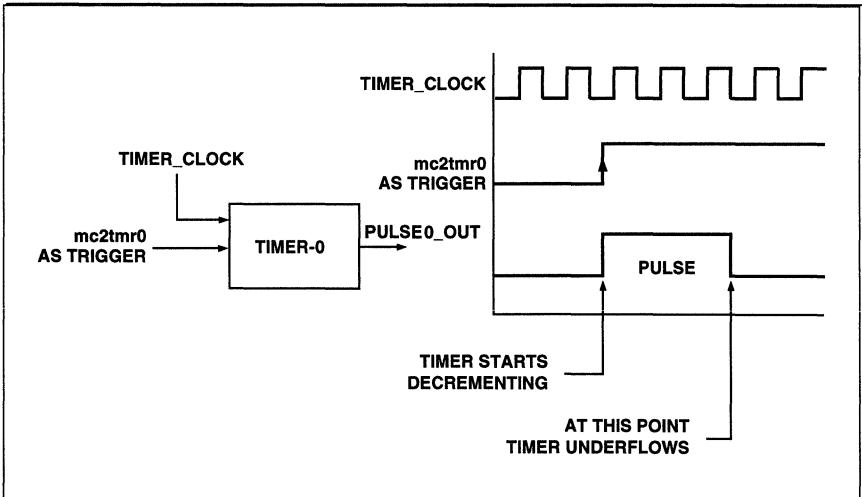
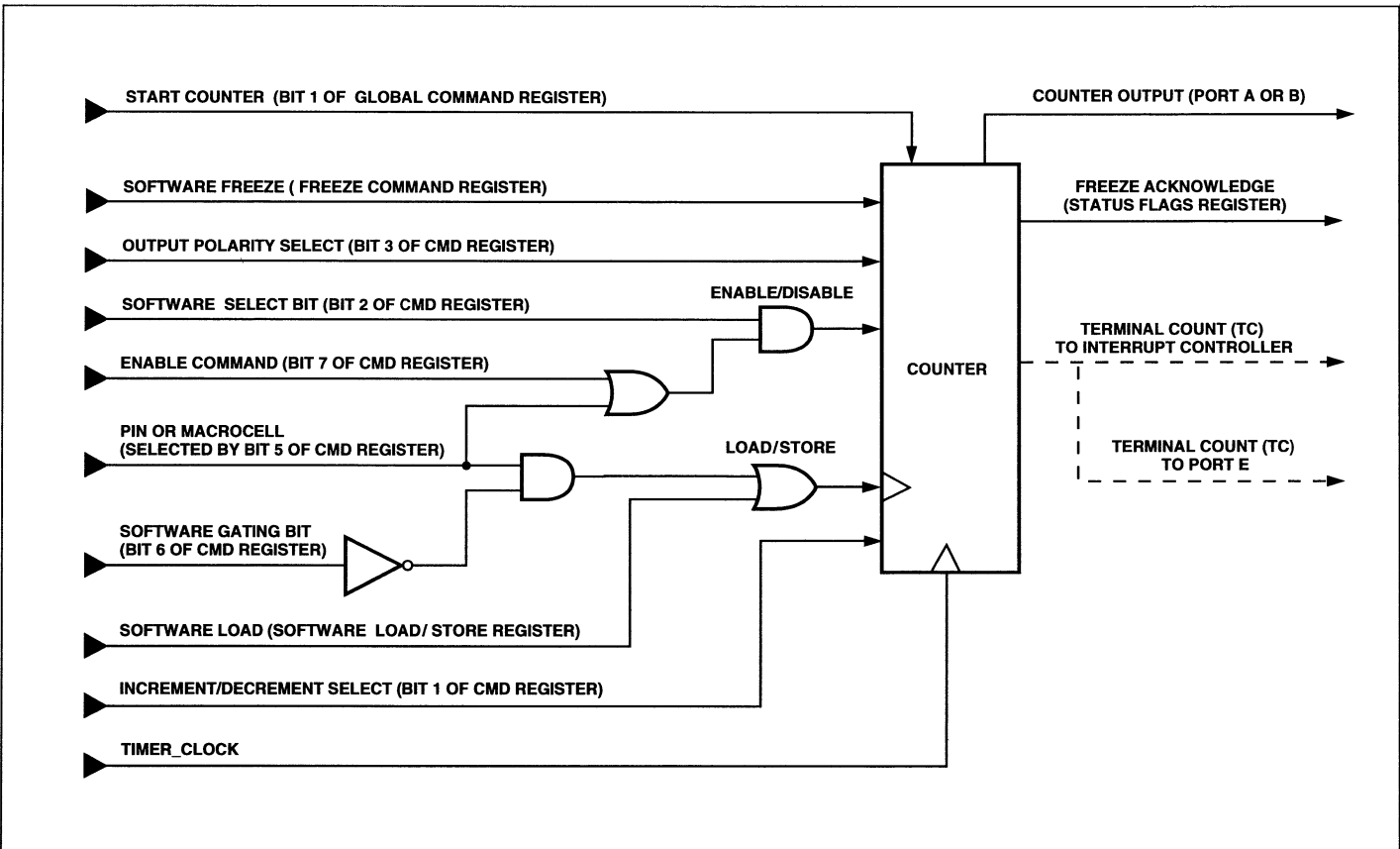


Figure 10. CTU Control Signals for Pulse Mode



**Different
Operating
Modes
(Cont.)**

The Pulse Mode (Cont.)

A Pulse Mode example

The following example explains the Pulse Mode application. The enable inputs to the Timers are generated by the PPLD.

The program flow to set up the Pulse Mode operation as described in this example is as follows:

1. Define the Timer input clock frequency (see section on Clock Scaling). Here 12MHz is scaled down to 3MHz.
2. Set up the Command Register for CTU0.
3. Initialize the Image Register with the proper count values to define the pulse width.
4. Specify the Timer pulse output. This can be done in two ways:
 - via .abl as pulse_out pin 27, which is a user defined name or
 - via PSDconfiguration software by specifying “waveform/pulse output”, which assigns the default name “timerout0”.
5. Set up the Special Function Register to specify the pin which is to be used as the output pin for the signal pulse_out.
6. The following equation is used to trigger Counter/Timer-0 (mc2tmr0). This equation is included in the design entry .abl file.

$$mc2tmr0 = (ext_signal1 \& !ext_signal2 \# !ext_signal3)$$

The ext_signal1 through ext_signal3 signals are the input signals to the PSD5XX pins. If these signals satisfy the above equations then Counter/Timer-0 is loaded on the rising edge of mc2tmr0. A pulse with a pulse width specified by the Image Register is generated on the output pin.

7. Set up the Global Command Register Start Bit and start the operation of the CTU.

The procedure to set up Counter/Timer Registers in the Pulse Mode in this design example is:

Counter/Timer-0 Registers Initialization

- Write “9D”hex to Command Register 0 (CMD0) at offset from base address of CSIOP(Chip Select I/O Port).

CMD0 Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
1	0	0	X	1	1	0	1

- Bit-0:** 1 Mode Select Bit, select Pulse Mode for CTU0.
- Bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU0 counts down from 3 to 0).
- Bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- Bit-3:** 1 Output polarity: Select output pulse to be active high.
- Bit-4:** X Input Polarity: No pin input in this mode, don't care.
- Bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- Bit-6:** 0 Load/Store Bit: Enable Load control by macrocell output.
- Bit-7:** 1 EN/DIS Bit: Enable continuous counting.

IMG0 is loaded with 03(hex) to define the pulse width of the output pulse (pulse0_out)



Different Operating Modes (Cont.)

The Pulse Mode (Cont.)

After Command Register 0 is initialized, other Registers (Special Function Register and Global Command Register) must now be initialized.

- Configure Port A pin PA0 as special function out, dedicating it as a Timer output pin by setting bit-0 to “1” in Port A Special Function Register. This bit is set only when there is a need to bring the timer output pulse out of the PSD5XX. Refer to the .frp report file to determine where the output is connected. Note that the device fitter might assign the pulse output to Port B pin PB0.

Special Function Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	0	1

- Now to start the Timers the Global Command Register has to be initialized to 02(hex), i.e., the following bits are set

Global Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	1	0

3

Bit-0: 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).

Bit-1: 1 Counter start bit: This bit turns on all the selected Timers.

Bit-2: 0 Global Mode bit: All Counter/Timers operate in Waveform or Pulse Mode.

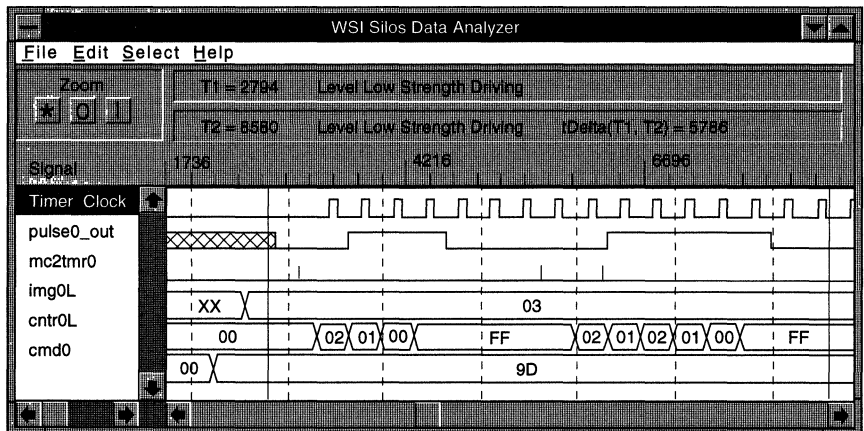
Bit-3: 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer-2).

Different Operating Modes
(Cont.)

The Pulse Mode (Cont.)

Figure 11 illustrates the basic pulse mode operation and retriggerability of the Counter/Timer in the pulse mode.

Figure 11. Simulation of Pulse Mode



Input Signals

Counter/Timer-0 trigger signal: **mc2tmr0**

PSD5XX input clock: **Timer_Clock**

Output Signals

Pulse output at: **pulse0_out**

Note that mono-shot output, active high (pulse width = 03) and retriggered mono-shot output (pulse width = 03 + 02 = 05) are simulated.

Figure 11 shows the simulation result of Pulse Mode on the PSDsim simulator. The Counter/Timer trigger signal mc2tmr0 is enabled as soon as the ext_signal signals listed in the Abel equation are satisfied. When mc2tmr0 is True (= High-State Pulse), Counter/Timer-0 starts outputting a pulse with programmed pulse width equal to 03. The second output pulse (when mc2tmr0 is True again) is longer although the Counter/Timer is loaded with a count value of 03 because the Counter/Timer is re-triggered for the second time before the output pulse generated by the first trigger dies. Therefore the pulse width of the second pulse is 05.

Different Operating Modes (Cont.)

The Event Counter Mode

Event counting is a common feature in many Microcontroller applications. An example could be counting the number of soda bottles on an assembly line or number of positive transitions in an incoming signal.

The advantage of using an event counter on the PSD5XX is that the PPLD allows several external signals to be combined to define an event. The following equation shows how this can be implemented.

$$mc2tmr0 = ext_signal1 \& !ext_signal2 \# !ext_signal3$$

In this example each time mc2tmr0 is true (has a Zero-to-One transition) the count value in Counter/Timer-0 increments by one.

Event Count Mode Features on the PSD5XX:

- Up to 3 Event Counters are available.
- Event latching is input signal edge sensitive.
- If the input control is by macrocell, then the input polarity is defined in the .abl file, which in turn defines the active edge. In order to get falling edge sensitivity the macrocell equation has to be inverted, i.e., preceded by a negation sign (!).
- If the input control is by the pin, then the input polarity bit in the Counter/Timer Command Register (bit-4) is used to define the edge (example: input polarity active high => rising edge sensitive).
- By using Freeze and Freeze acknowledge signals, the count value can be read from the Image Register without affecting the actual event count.
- For an event to be counted, the minimum time distance between two successive events should be at least 1 Timer_Clock period of the Counter/Timer input clock.
- Unlike Timers on standard microcontrollers, there are three different ways to create Counter/Timer events on the PSD5XX:
 - Input pin PE3-PE6 (port E).
 - mc2tmr0-3 inputs in PPLD.
 - Software Load Commands.

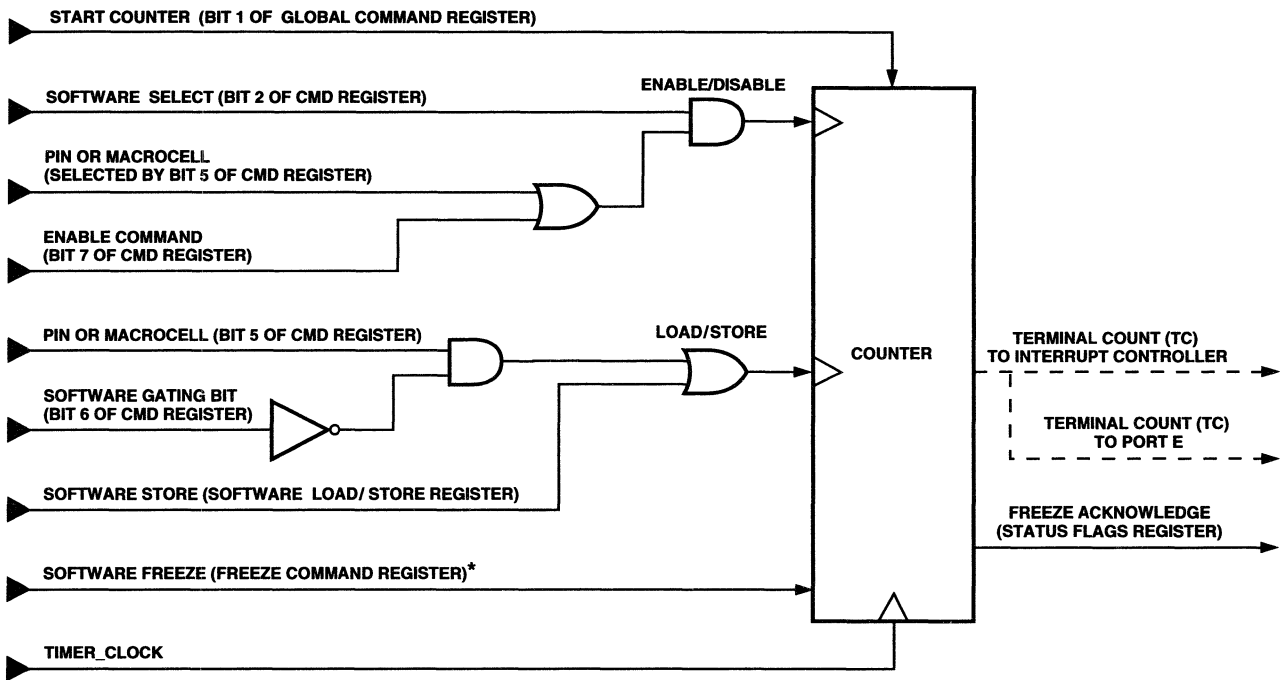
Refer to figure 12 for control signals needed to operate in Event Count Mode.

In this example when the Counter/Timer is active every Low-to-High transition on mc2tmr0 will increment the IMG0 (Image Register) of Counter/Timer-0.

Once the freeze signal is set, the image content is "Frozen" and the counter keeps on counting the events. The moment freeze is cleared, the counter updates the Image Register. Therefore the events will always be counted, independent of the freeze command.

Different Operating Modes
(Cont.)

Figure 12. CTU Control Signals For Event Count Mode



*Count updates are continuously stored in the image register, unless frozen by the software freeze command.



Different Operating Modes

(Cont.)

The Event Counter Mode

Event Counter Design example:

Generating the event input to the Counter/Timer:

The event input (mc2tmr0) is defined in the following equation:

$$\text{mc2tmr0} = (\text{ext_signal1} \ \& \ !\text{ext_signal2} \ \# \ !\text{ext_signal3})$$

where ext_signals are control inputs and mc2tmr0 is the output from the PPLD.

Any rising edge on the mc2tmr0 is counted by the Counter/Timer-0 as one event and thus will increment the counter by one.

Input clock to the Counter/Timer:

In this application note the default scale down factor (4) of the PSD5XX input clock is used:

default PSD5XX input clock = 12 MHz

Scale down factor = 4

Counter/Timer input clock = 12 MHz/4 = 3 MHz.

For a guaranteed event counting without a miss, the events must be separated by at least one timer clock plus 2 CLKIN clock periods.

The program flow to set up the Event Count Mode operation as described in this example is as follows:

1. Define Timer input clock frequency (see section on Clock Scaling). Here 12 MHz is scaled down to 3 MHz.
2. Set up Command Register for CTU0.
3. Initialize IMG0 and CNTR0 Registers to 00.
4. The following equation is used to trigger events on Counter/Timer 0 (mc2tmr0). This equation is included in the design entry .abl file.

$$\text{mc2tmr0} = (\text{ext_signal1} \ \& \ !\text{ext_signal2} \ \# \ !\text{ext_signal3})$$

The ext_signal1 through ext_signal3 signals are the input signals to the PSD5XX pins. If these signals satisfy the above equations then the IMG0 Register gets incremented at every rising edge of mc2tmr0.

5. Set up the Global Command Register Start Bit and start the operation of the CTU0.
6. To read the count event count updates, freeze the Image Register(IMG0) by writing 01 into Freeze Command Register.
7. Wait for the Freeze Acknowledge bit to be set to 1 for Counter/Timer-0 and then read the IMG0 Register.

Different Operating Modes (Cont.)

The Event Counter Mode (Cont.)

The procedure to set up Counter/Timer Registers in the Event Count Mode in this design example is:

Counter/Timer-0 Registers Initialization:

- Write "1E"hex to Command Register 0 (CMD0) at offset from base address of CSIOP (Chip Select I/O Port).

CMD0 Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	X	X	1	1	0

- Bit-0:** 0 Mode Select Bit, select Event Count Mode for CTU0.
- Bit-1:** 1 Decrement/Increment Bit: Increment after every event.
- Bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- Bit-3:** X Output polarity: No timer output, don't care.
- Bit-4:** X Input Polarity: No pin input in this mode, don't care.
- Bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- Bit-6:** 0 Load/Store Bit: Store control from macrocell.
- Bit-7:** 0 EN/DIS Bit: Enable or disable by macrocell.

NOTE: In this mode each event will enable the Counter/Timer for one Timer Clock cycle only.

IMG0 and CNTR0 Registers must be cleared to 00.

After Command Register 0 is initialized, other Registers (Global Command Register and Freeze Command Register) must now be initialized.

- Now to start the Timers: The Global Command Register has to be initialized to 06(hex), i.e., the following bits are set.

Global Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	1	1	0

- Bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between "divide by 8" or "divide by 1").
- Bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
- Bit-2:** 1 Global Mode bit: All Counter/Timers operate in Event Count or Time Capture Mode.
- Bit-3:** 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer-2).



Different Operating Modes (Cont.)

The Event Counter Mode (Cont.)

The Counter/Timer is turned on after a write takes place on the Global Command Register. The Counter/Timer will keep on counting the events and update the Image Register whenever the event count has changed. Follow these steps to read the Image Register:

- The Image Register must be “frozen” before it can be read.
- Write “1” to bit 0 of the Freeze Command Register (Corresponds to Counter/Timer-0).

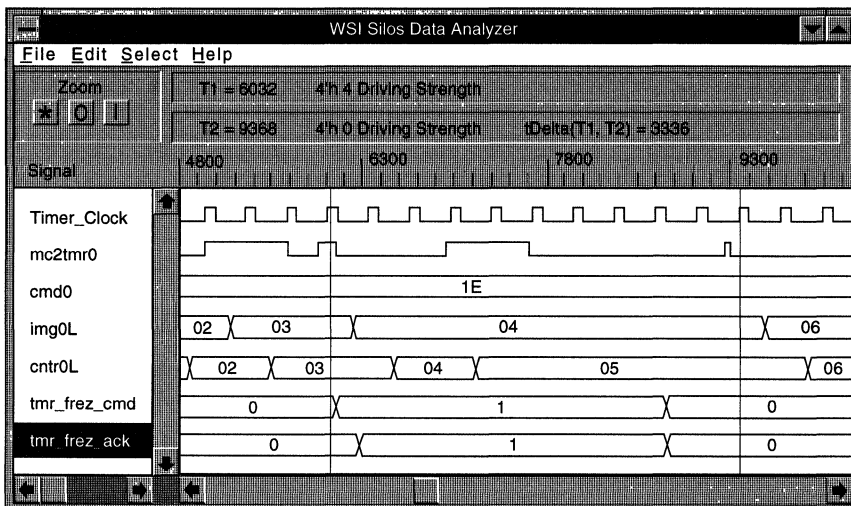
Freeze Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	0	1

- This will freeze the Image Register updates by the counter.
- Check the Freeze Acknowledge bit-0 in the Status Register. A microcontroller can access the Image Register accurately only when the Freeze Acknowledge bit-0 is set to “1”.
- The Freeze Command bit-0 must be cleared to 0 by the Microcontroller to resume normal counting and to negate the Freeze-Acknowledge signal.

Different Operating Modes (Cont.)

Figure 13. Simulation of Event Counter Mode



Input Signals

Counter/Timer-0 event input signal: mc2tmr0
 Every rising edge transition on mc2tmr0 is considered as an event occurrence.
 Prescaled PSD5XX input clock: Timer_Clock

Output Signals: None

In this example each EVENT is counted by the Image Register (Img0L).

Figure 13 illustrates that at every rising edge transition on mc2tmr0, the count in the img0L Register is updated. Register cntr0L serves as the event counter when the Freeze Command is active so the events are not lost during the freeze.

NOTE: If the Store control bit (bit-6) in the CMD0 Register is set to “1” i.e., Store control by “Software”, then before the Freeze Command is issued a Software Store Command must be issued by writing into the Software Load/Store Register. In this mode cntr0L counts the incoming events. The Software Store Command updates the Image Register in this case, prior to the Freeze Command, with the correct numbers of the counted events.

Different Operating Modes (Cont.)

The Time Capture Mode

Up to three out of four Counter/Timers in the PSD5XX can be configured to operate in the Time Capture Mode. In this Mode a counter is continuously counting at the Timer Clock rate. At each transition of the trigger input signal (rising or falling edge), the counter value is transferred to the associated Image Register.

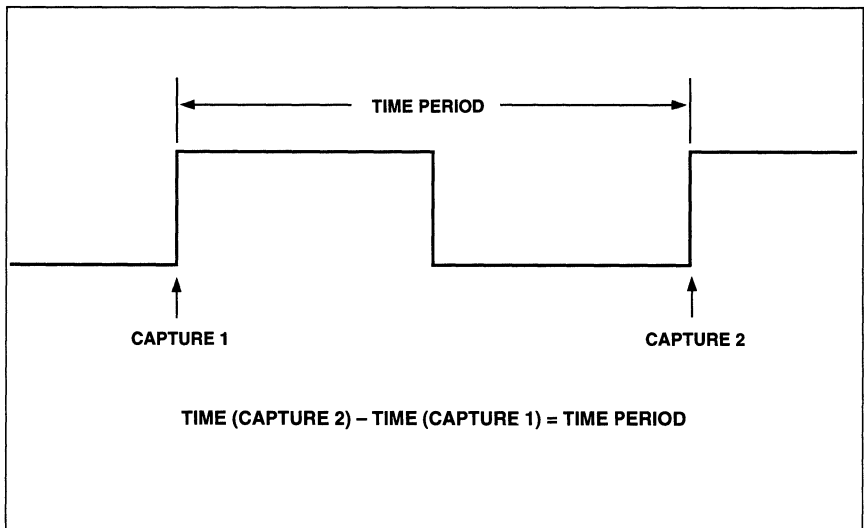
Typical applications of Time capture Mode are:

- Measuring Periods
- Pulse widths
- Frequencies
- Phase differences of signals.

To measure the time period of a signal as seen in Figure 14a, a single Counter/Timer can be used to capture the consecutive rising edges of the input signal. The difference in the value of these two captures is used to calculate the time period of the signal.

In this application note example, the measuring of “pulse width” of an input signal is depicted.

Figure 14a. Measuring Time Period Using Time-Capture



**Different
Operating
Modes
(Cont.)**

The Time Capture Mode (Cont.)

To measure the pulse width of a signal as seen in figure 14b, the Counter/Timers have to capture both the rising (capture1) and the falling (capture2) edges of the signal. The difference in the value of these two captures is used to calculate the pulse width of the signal. Usually one Counter/Timer is configured to capture the falling edge of the signal from the input pin. Another Counter/Timer is used to capture the rising edge. The time distance between the two edges must be greater than one Counter/Timer input clock in order to be captured.

Figure 15 depicts the control signals required for the time capture Mode of operation.

The Counter counts up every Timer Clock cycle. Whenever a Low-to-High transition occurs on the selected event input (mc2tmr1 in this example) the Image Register is updated by the count value in the Counter.

The microcontroller reads the image value using the Freeze/Freeze Acknowledge handshake protocol. The Freeze Command blocks the event input when set to 1.

Figure 14b. Measuring Pulse Width Using Time-Capture

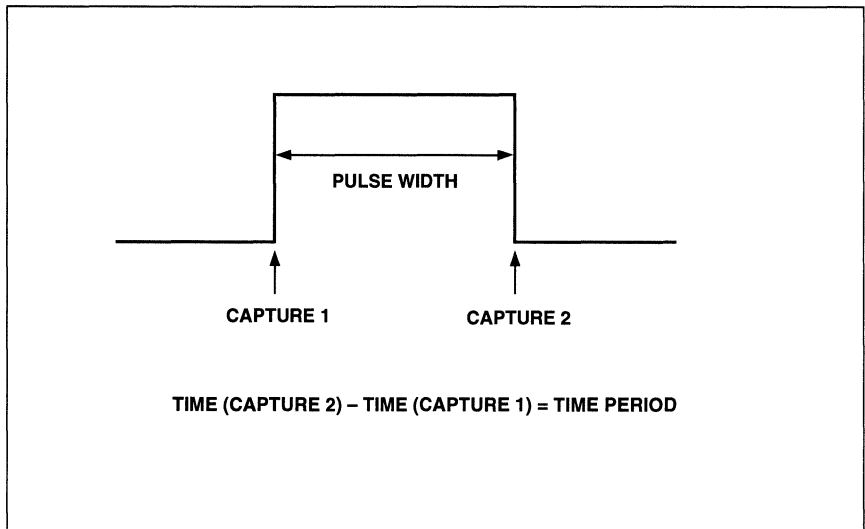
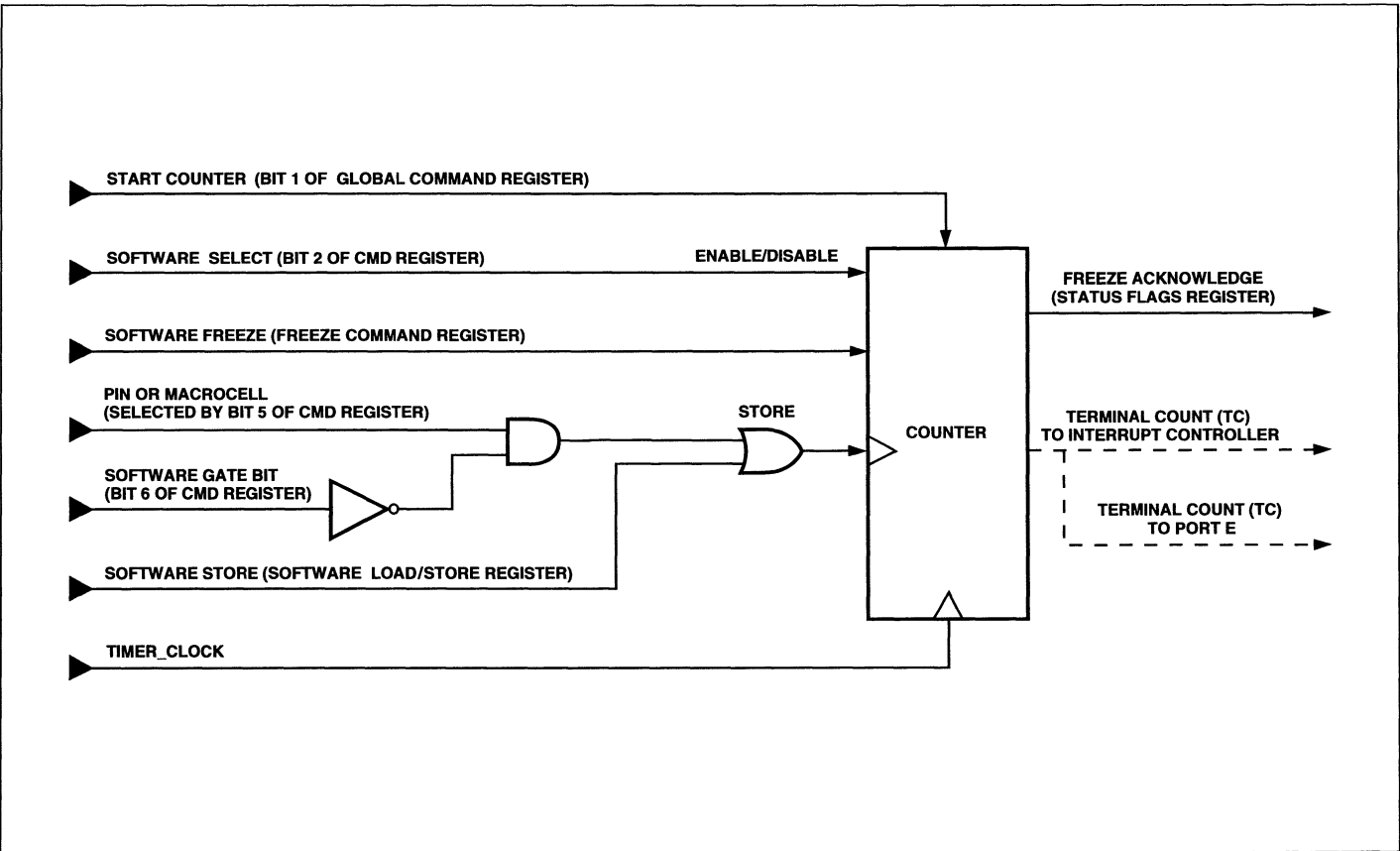


Figure 15. CTU Control Signals For Time-Capture Mode



Different Operating Modes (Cont.)

The Time Capture Mode (Cont.)

Time Capture Design Example of a Pulse Width Measurement:

In this example, the input signal is named as input_pulse. This signal can be input to the Counter/Timers in two ways:

1. Connected to the input pins PE3 and PE4 of Counter/Timer-0 and Counter/Timer-1 respectively. And capture the counters values at the rising and falling edges of input_pulse.
or
2. Connected to the input pin PE3 of Counter/Timer-0. And the input to Counter/Timer-1 comes from the PPLD, defined by
mc2tmr1 = input_pulse; (Refer to the .abl file)

Thereby only one pin (PE3) is used as both Counter/Timers input. This application note uses the second method to input the input_pulse. This method saves pin PE4 for other purposes.

The capturing of the leading and trailing edge values of counters can be done, by properly defining the input polarity on the pin PE3 (for Counter/Timer-0) and defining the input equation in the .abl file for mc2tmr1 (Counter/Timer-1) with proper polarity.

In this example the leading edge of the input signal is captured by Counter/Timer-1 Image Register and the trailing edge is captured by Counter/Timer-0 Image Register.

Clock Input to the Counter/Timer:

In this application note, the default scale down factor(4) of the PSD5XX input clock has been used.

i.e., PSD5XX input clock = 12 MHz

Scale down factor = 4 (default)

Counter/Timer input clock = $12/4$

= 3 MHz => 333ns period

Therefore the input pulse width must be greater than 333 ns in order to be captured by the Counter/Timer. Note that at 3 MHz Timer Clock input the pulse width measurement will have a resolution of $\pm 333/2$ ns.

The program flow to set up the Time Capture Mode operation as described in this example is as follows:

1. Define the Timer input clock frequency (see section on clock scaling). Here 12MHz is scaled down to 3MHz.
2. Set up Command Registers for CTU0 and CTU1.
3. Initialize IMG0, CNTR0, IMG1, CNTR1 Registers to 00.
4. The following equation is used to trigger the Time Capture on Counter/Timer-1 (mc2tmr1). This equation is included in the design entry .abl file.

mc2tmr1 = input_pulse;

If the signal input_pulse satisfies the above equation, every rising edge on mc2tmr1 causes a Time Capture in IMG1 for Counter/Timer-1. Every Falling edge on input_pulse pin causes a Time Capture in IMG0 for Counter/Timer-0.

5. Set up the Global Command Register Start Bit and start the operation of the CTU0 and CTU1.
6. To read the count of Time Capture updates, freeze the Image Registers (IMG0 and IMG1) by writing 03 into Freeze Command Register.
7. Wait for the Freeze Acknowledge bits to be set to 3 for Counter/Timer-0 and Counter/Timer-1 and then read the IMG0 and IMG1 Registers.

Different Operating Modes (Cont.)

The Time Capture Mode (Cont.)

Procedure to set up Counter/Timer Registers for Time Capture Mode in this design example:

Counter/Timer-0 Registers Initialization

- Write "BF"hex to Command Register 0 (CMD0) at offset from base address of CSIOP(Chip Select I/O Port).

CMD0 Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
X	0	1	1	X	1	1	1

Bit-0: 1 Mode Select Bit, select Time Capture Mode for CTU0.

Bit-1: 1 Decrement/Increment Bit: Increment mode.

Bit-2: 1 Select Counter/Timer Bit: Select CTU0.

Bit-3: X Output polarity: No timer output, don't care.

Bit-4: 1 Input Polarity: Active low.

Bit-5: 1 Pin or Macrocell input: Pin input control.

Bit-6: 0 Load/Store Bit: Store control from pin.

Bit-7: X EN/DIS Bit: Don't care. Setting of bit-2 enables the Counter/Timer.

IMG0 and CNTR0 Registers must be cleared to 00.

Counter/Timer-1 Registers Initialization:

- Write "9F"hex to Command Register 1 (CMD1) at offset from base address of CSIOP(Chip Select I/O Port).

CMD1 Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
X	0	X	1	X	1	1	1

Bit-0: 1 Mode Select Bit, select Time Capture Mode for CTU1.

Bit-1: 1 Decrement/Increment Bit: Increment mode.

Bit-2: 1 Select Counter/Timer Bit: Select CTU1.

Bit-3: X Output polarity: No timer output, don't care.

Bit-4: X Input Polarity: Don't care.

Bit-5: 0 Pin or Macrocell input: Macrocell input control.

Bit-6: 0 Load/Store Bit: Store control from macrocell.

Bit-7: X EN/DIS Bit: Don't care. Setting of bit-2 enables the Counter/Timer.

IMG1 and CNTR1 Registers must be cleared to 00.

**Different
Operating
Modes
(Cont.)**

The Time Capture Mode (Cont.)

After Command Registers 0 and 1 are initialized, other Registers (Global Command Register and Freeze Command Register) must now be initialized.

- Now to start the Timers. The Global Command Register has to be initialized to 06(hex), i.e., the following bits are set.

Global Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	1	1	0

Bit-0: 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).

Bit-1: 1 Counter start bit: This bit turns on all the selected Timers.

Bit-2: 1 Global Mode bit: All Counter/Timers operate in Event Count or Time Capture Mode.

Bit-3: 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer-2).

The Counter/Timer is turned on after a write takes place on the Global Command Register. The Counter/Timer will keep on incrementing at every Timer Clock cycle and update the Image Register whenever an event has occurred. Follow these steps to read the Image Register:

- The Image Registers must be “frozen” before they can be read.
- Write “3” into the Freeze Command Register (Corresponds to Counter/Timers 0 and 1).

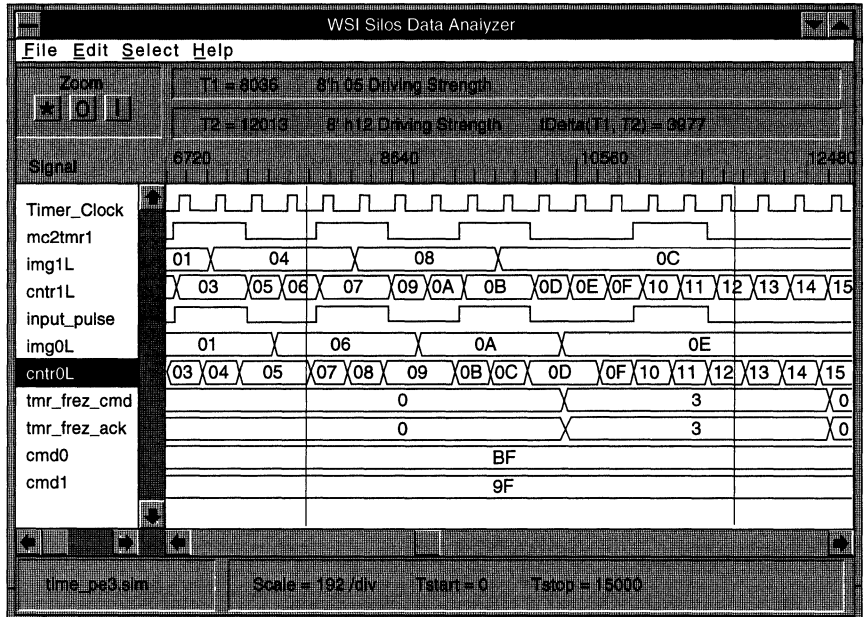
Freeze Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	1	1

- This will freeze the Image Register updates by the counters.
- Check the Freeze Acknowledge bit-0 and bit-1 in the Status Register.
A microcontroller can access the Image Register accurately only when the Freeze Acknowledge bit-0 and bit-1 are set to “3”.
- After the completion of the Image Registers read operation, the microcontroller lowers the freeze command bits which in turn, cause the Freeze Acknowledge signals to go to low.
- The Freeze Command bit-0 and bit-1 must be cleared by the microcontroller to do enable continuous updates of the Image Registers.

Different Operating Modes (Cont.)

Figure 16. Time Capture Mode Simulation



Input Signals

Input Pulse on pin for pulse width measurement: **input_pulse**

PSD5XX input clock: **Timer_Clock**

Output Signals: None

The Pulse Width Computation:

Figure 16 illustrates that at the falling edge of the input signal on **input_pulse** (pin PE3) and the rising edge of the input signal on **mc2tmr1**, the present counts in the Counter/Timer-0 and Counter/Timer-1 Registers are transferred to Image-0 and Image-1 Registers. Here the Image-0 Register is updated to count 0A and Image-1 Register to count 08. Note when the Freeze Command is active high the Image Register updates are frozen.

The pulse width of the **input_pulse** signal is

$$(\text{Image-0 Register} - \text{Image-1 Register}) = 0A - 08 = 02$$

Here each Counter/Timer clock cycle = 333 ns.

Therefore, the pulse width of the sample signal is:

$$\begin{aligned} &= 02 * \text{Counter/Timer clock input period} \\ &= 02 * 333 \text{ ns} \\ &= 666 \text{ ns} \end{aligned}$$

Different Operating Modes
(Cont.)

The Watchdog Mode

Watchdog Timer is very useful in situations where the software program is repeating in an endless loop or the program jumps to an unexpected area. When this happens the Watchdog Timer usually generates a Reset or interrupt to the microcontroller to initialize the system.

Only Counter/Timer-2 in the PSD5XX is capable of the Watchdog function. While Counter/Timer-2 operates in Watchdog Mode, the other three Counter/Timers in the PSD5XX can be configured to operate in different modes. Table 2 shows the possible mode combinations.

Table 2. Possible Mode Combinations

Global Mode Bit (Global Command Register)	Mode Select Bit (CMD0, CMD1, CMD2 CMD3 Registers)	Modes of Counter/Timers 0, 1 and 3	Modes of Counter/Timer-2
0	0	Waveform	Waveform or Watchdog
0	1	Pulse	Pulse or Watchdog
1	0	Event Counter	Watchdog Only
1	1	Time Capture	Watchdog Only

Special Features of the Watchdog Counter/Timer are:

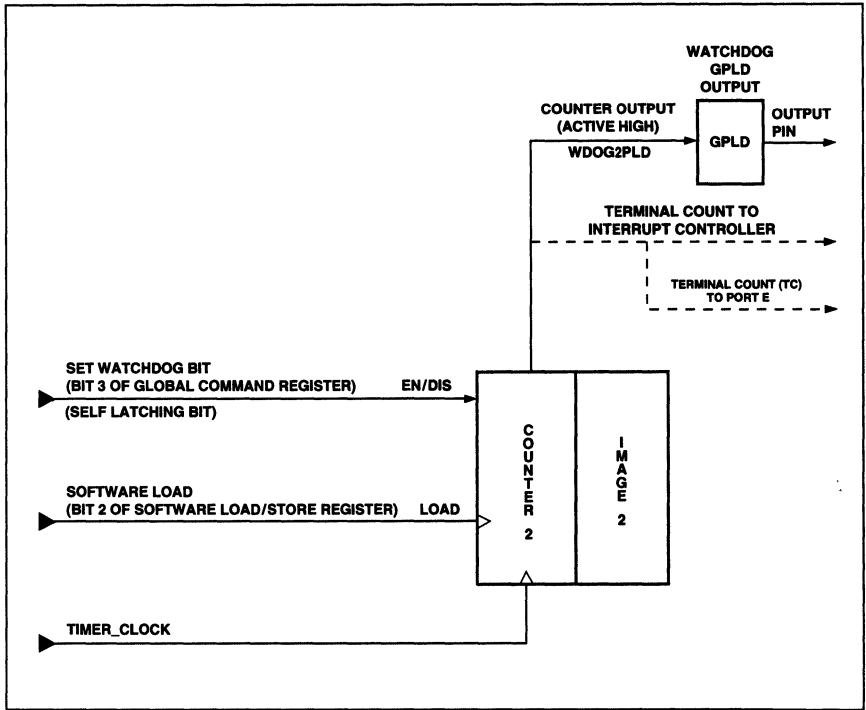
- Once set in Watchdog Mode, Counter/Timer-2 cannot be reconfigured by software. It can get out of the Watchdog Mode only by resetting the PSD5XX.
- Terminal Count signal of a Watchdog results in a pulse with a width equal to the duration count value loaded into the Image Register of the Counter/Timer-2.
- The active-high Watchdog pulse from the Counter/Timer-2 (wdog2pld) is routed as input to the ZPLD. The user can select any of the I/O pins of the GPLD as the Watchdog output and invert its active high level, if needed.
- During Watchdog Mode, Counter/Timer-2 counts down and generates a Watchdog pulse at the terminal count. To avoid the generation of a Watchdog pulse, Counter/Timer-2 has to be reloaded before the terminal count occurs. This can be done by writing "1" into bit-2 of the "Software Load/Store Register" before the terminal count occurs.

Figure 17 depicts the inputs and output of Counter/Timer-2 operating in Watchdog mode.

Different Operating Modes

(Cont.)

Figure 17. CTU Control Signals For Watchdog Mode



3

The Watchdog Mode Design Example:

This example simulates the occurrence of the Watchdog condition and generation of a Watchdog output pulse. The procedure to inhibit the Watchdog occurrence has also been simulated in a later part of this example.

The program flow to set up the Watchdog Mode operation as described in this example is as follows:

1. Define Timer input clock frequency (see section on clock scaling). Here 12MHz is scaled down to 3 MHz.
2. Ignore Command Register for CTU2 in Watchdog mode.
3. Initialize CNTR2 Register and IMG2 Register to a required value.
4. Write "1" into the "Counter Start Bit" and "Watchdog bit" of the Global Command Register simultaneously to start the Watchdog operation.
5. To inhibit the occurrence of the Watchdog and generation of the Watchdog output pulse, write "1" into bit-2 of Software Load/Store Register before Counter/Timer-2 under flows.

Different Operating Modes (Cont.)

The Watchdog Mode (Cont.)

Procedure to set up Counter/Timer Registers for Watchdog Mode in this design example:

Counter/Timer-2 Registers Initialization

- Image-2 Register and CNTR2 are loaded with 02(hex), which is the pulse width of the Watchdog pulse wdog2pld. Normally this value is very large depending on the application, since software is not supposed to clear the Watchdog very often.

After CNTR2 and IMG2 are initialized, other Registers (Global Command Register and Software Load/Store Register) must now be initialized.

- To start the Timers, the Global Command Register has to be initialized to 0A(hex), i.e., the following bits are set.

Global Command Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	1	X	1	0

Bit-0: 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).

Bit-1: 1 Counter start bit: This bit turns on all the selected Timers.

Bit-2: X Global Mode bit: Watchdog is available in both Global modes.

Bit-3: 1 Watch Dog bit: Watchdog Mode (affects only Counter/Timer-2).

The moment the Watchdog bit and Counter start bit in the Global Command Register are set to 1, the Watchdog counter starts counting down. To avoid the generation of the Watchdog output pulse, the software must write “1” to bit 2 of the Software Load/Store Register before the Counter/Timer-2 under flows.

Software Load/Store Register

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	1	0	0

Once the Counter/Timer-2 is configured to operate in the Watchdog Mode, only a hardware reset can get it out of the Watchdog Mode.

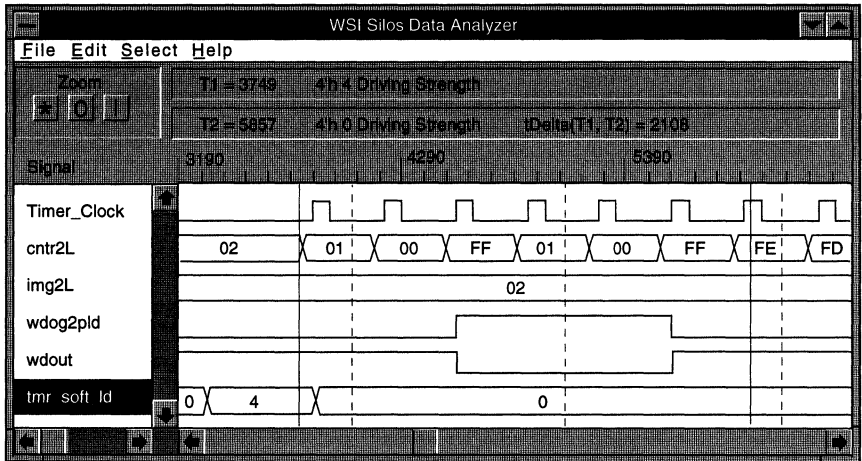


Different Operating Modes (Cont.)

The Watchdog Mode (Cont.)

Figure 18 shows the simulation result of the Watchdog Mode. The Counter/Timer software load bit was not written into after Watchdog Counter-2 decrements from value 2 through 0. Therefore when Counter-2 under flowed, a Watchdog Pulse is generated. This pulse, **wdog2pld**, is inverted by the **GPLD** and is available on the output pin as “**wdout**”.

Figure 18. Simulation of Watchdog Mode



Input Signals:

PSD5XX input clock: **Timer_Clock**

Outputs:

Watchdog output from Counter/Timer-2: **wdog2pld**

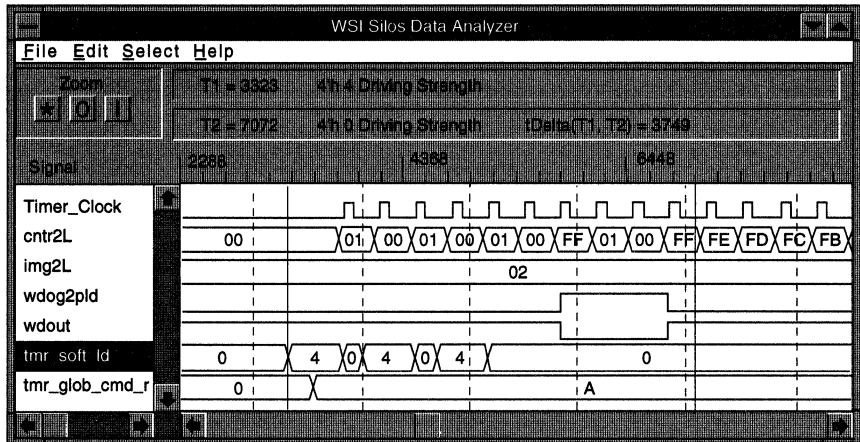
Watchdog output from GPLD as: **wdout**

Different Operating Modes (Cont.)

The Watchdog Mode (Cont.)

Figure 19 shows the Watchdog occurrence is inhibited by writing “04” hex into Software Load/Store Register before COUNTER-2 under flows. There are four software loads: the second, third and fourth software loads are done before COUNTER-2 under flowed (count of 02 = 666ns)

Figure 19. Simulation of Inhibition of Watchdog Occurrence



The `tmr_soft_ld` Register is the Software Load/Store Register. The first software load pulse initializes the Watchdog operation.

Conclusion

PSD5XX offers a powerful set of four **PLD macrocell controlled Counter/Timers**.

Included in this application note are some of the files generated for the Waveform Mode application:

Appendix 1. .abl file

Abel file with Counter/Timer logic equations (Waveform mode).

Appendix 2. .crp file

PSD-Global Configuration1 report file (Waveform mode).

Appendix 3. .stl file

Stimulus file simulating Waveform Mode operation.

Appendix 4. .c file

Initializations of PSD5XX Counter/Timer Registers (each of all 5 modes) based on 80C196 “C”.

Appendix 5. .asm file

Initializations of PSD5XX Counter/Timer Registers (each of all 5 modes) based on 68HC11 assembly.

Appendix 6. 4-PWM Timers

This article depicts the realization of 4-PWM Timers on PSD5XX using PPLD.

The .c and .asm initialization files related to PSD5XX operating in each of the five modes i.e., the Waveform mode, the Pulse mode, the Event Count mode, the Time Capture mode and the Watchdog mode are available on WSI's Bulletin Board.

Appendix 1. .ABL File

```
module wavfrm " 7/20/93
" Full pathname - c:\psdsoft\simulate\wavfrm\wavfrm.abl
title 'wavform mode';
```

"Input signals

"Address lines, using reserved names.

```
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
```

"Output signals

"The Output signal here has been declared in .abl itself, it can
"also be declared in the configuration file as Waveform/Pulse o/p.
"The user can declare it here or in the configuration menu.

PWM_OUT pin 27; "Port A PA0 has been aliased as PWM_OUT,
"if selected in the configuration file instead
"of in the .abl file, pin PA0's default name will be ' Timerout0 '

"Internal PSD5XX PLD output signals.

```
csiop node ; "More outputs using reserved names.
```

"Definitions

```
X = .x. ; "Don't care
Address = [a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,a7,X,X,X,X,a2,a1,a0];
```

equations

```
csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF) ; " 256 block
```

END

**Appendix 2.
Waveform
Mode
Configuration**

W S I - PSDsoft Version 1.02B
Output of PSD Configurations

PROJECT: wavfrm DATE: 08/13/1993
DEVICE: PSD503B1 TIME: 10:17:15

BUS INTERFACE

Data bus width = 8-Bits
Address/Data Mode = Multiplexed
ALE/AS signal = Active High
Read/Write signals = /WR, /RD

OTHER CONFIGURATIONS

Timer/Counter 0 INPUT = OFF
Timer/Counter 0 OUTPUT = ON
Timer/Counter 1 INPUT = OFF
Timer/Counter 1 OUTPUT = OFF
Timer/Counter 2 WATCH DOG = OFF
Timer/Counter 2 INPUT = OFF
Timer/Counter 2 OUTPUT = OFF
Timer/Counter 3 INPUT = OFF
Timer/Counter 3 OUTPUT = OFF

Power Down Clock = OFF
Security Function = OFF
Interrupt Function = OFF

END OF REPORT FILE: wavfrm.crp

Appendix 3. .STL File

```
//This is a stimulus file to simulate "Waveform" mode of operation of PSD5XX.
//Counter/Timers 0 and 1 are used in this simulation. The input clock to PSD is 12 MHz.
The //duty cycle of the output waveform (PWM_OUT) is 33%
//
//          User defined parameters
//

parameter load_store = 'hC0A5, dlcyl='hC0A6, cmd0 = 'hC0A0, cmd1 = 'hC0A1;
parameter img0_lobyte = 'hC090, img0_hibyte = 'hC091, img1_lobyte = 'hC092;
parameter img1_hibyte = 'hC093, cntr0_lobyte='hC098, cntr0_hibyte = 'hC099;
parameter cntr1_lobyte='hC09A, cntr1_hibyte = 'hC09B, global_command = 'hC0A8;
parameter special_func = 'hC008;

//User-Defined tasks

task write (addr_bus, data_in);

input [15:0] addr_bus;
input [7:0] data_in;

    begin

        #20    ale = 1;
        #20    adio = addr_bus;
        #20    ale = 0;

        #20    adio = data_in;
        #40    wr = 0;
        #100   wr = 1;

    end

endtask

task read (addr_bus);

input [15:0] addr_bus;

    begin

        #20    ale = 1;
        #20    adio = addr_bus;
        #20    ale = 0;

        #20    adio = Z16;
        #40    rd = 0;
        #100   rd = 1;

    end

endtask

//End user-defined tasks
```

Appendix 3.
.STL File
(Cont.)

```
initial
begin

    clkIn = 0; reset = 0; csi = 0;
    rd = 1; wr = 1; ale = 0;
    adio = 'h0000; PWM_OUT = Z;

    #560 reset = 1;

// CSIOP selection
read('hC002);

//Counter-0 Low Byte data initialized to 0
write(cnr0_lobyte, 00);
//end of writing into Counter0 Low-byte reg

//Counter-0 High Byte data initialized to 0
write(cnr0_hibyte, 00);
//end of writing into Counter0 High Byte reg

//Counter-1 Low Byte data initialized to 0
write(cnr1_lobyte, 00);
//end of writing into Counter0 Low Byte reg

//Counter-1 High Byte data initialized to 0
write(cnr1_hibyte, 00);
//end of writing into Counter0 High Byte reg

//Writing DLCY data
write(dlcy, 00);

//end of writing into Dlcy reg
```

Appendix 3. .STL File (Cont.)

```
//Writing CMD-0 data
```

```
write(cmd0, 'hD4');
```

```
/*
```

Write "D4"hex to Command Register-0 (CMD0) at offset from base address of CSIOP(Chip Select I/O Port).

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
1	1	X	X	0	1	0	0

CMD0 Register

- **bit-0:** 0 Mode Select Bit, select Waveform Mode for CTU0.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU0 counts down from 2 to 0. At 0, TC triggers the loading and operation of the CTU1).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** 0 Output polarity: Select output to be active low (Toff time).
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** X Pin or Macrocell input: No pin or macrocell input, don't care.
- **bit-6:** 1 Load/Store Bit: No pin or macrocell load/store.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

```
*/
```

```
//end of writing into CMD0 reg
```

```
//Writing CMD-1 data
```

```
write(cmd1, 'hDC');
```

```
/*
```

Write "CC"hex to Command Register 1 (CMD1).

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
1	1	X	X	1	1	0	0

CMD1 Register

- **bit-0:** 0 Mode Select Bit, select Waveform Mode for CTU1. .
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU1 counts down from 4 to 0. At 0, TC triggers the loading and operation of the CTU0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU1.
- **bit-3:** 1 Output polarity: Select output to be active on (Ton time).
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** X Pin or Macrocell input: No pin or macrocell input, don't care.
- **bit-6:** 1 Load/Store Bit: No pin or macrocell load/store.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

```
*/
```

```
//end of writing into CMD1 reg
```


Appendix 3.
.STL File
(Cont.)

```
//IMG-0 high byte data written to
write(img0_hibyte, 00);
//end of writing into IMG0 high byte reg

//IMG-0 data written to Low Byte
write(img0_lobyte, 04);
//end of writing into IMG0 reg

//IMG-1 high byte data written to
write(img1_hibyte, 00);
//end of writing into IMG1 high byte reg

//IMG-1 data written to Low Byte
//Setting up address C092h
write(img1_lobyte, 02);
//end of writing into IMG1 reg low byte

//Declare port A as special function port, so that Timer0 o/p is on PA0
//Write Data of 01 so that PA0 has timer-0 o/p available on its pin
write(special_func, 01);
//end of writing into special function reg
```

Appendix 3. .STL File (Cont.)

```
//Waveform output is first initialized by setting its two corresponding
//software Load/Store bits after loading the Image Registers.

//Writing into software load/store bit_0 and bit_1 for cntr_0 and cntr_1

write(load_store, 03);

//Start Counter/Timer-0 and Counter/Timer-1 to produce waveform

//Global Reg data written to
```

```
write(global_command, 02);
/*
```

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
0	0	0	0	0	0	1	0

Global Command Register

- **bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between "divide by 8" or "divide by 1").
 - **bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
 - **bit-2:** 0 Global Mode bit: All Counter/Timers operate in Waveform or Pulse Mode.
 - **bit-3:** 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer 2).
- ```
*/
```

```
//end of writing into Global Reg

//read data on Load/Store reg, it should be F0

read(load_store);

end

always
#42 clkIn = ~clkIn; //12 Mhz PSD5xx input clock
```

## Appendix 4. .C File

```

/*****
The following C program illustrates the initialization procedure to operate the PSD5XX
Counter/Timers interfaced with Intel's 80C196.
*****/

#define CSIOP 0x3000 /*Chip select & I/O offset base address */

/*Global declarations*/
int *CNTR0, *CNTR1, *CNTR2, *CNTR3 ;
int *IMG0, *IMG1, *IMG2 *IMG3 ;
char *dlyc_reg, *CMD0, *CMD1, *CMD2, *PORTA, *GLOB, *FREEZE_CMD ;
char *STATUS_FLAGS, *SOFTWARE_LD_ST ;

main()
{
/* Initialization of PSD5XX Counter/Timers */

CNTR0 = (int *) (CSIOP + 0x98); /*CNTR0 offset from CSIOP */
*CNTR0 = 00; /*CNTR0 initialized to 0 */

CNTR1 = (int *) (CSIOP + 0x9a); /*CNTR1 offset from CSIOP */
*CNTR1 = 00; /*CNTR1 initialized to 0 */

CNTR2 = (int *) (CSIOP + 0x9c); /*CNTR2 offset from CSIOP */
*CNTR2 = 00; /*CNTR2 initialized to 0 */

CNTR3 = (int *) (CSIOP + 0x9e); /*CNTR3 offset from CSIOP */
*CNTR3 = 00; /*CNTR3 initialized to 0 */

IMG0 = (int *) (CSIOP + 0x90); /*IMG0 offset from CSIOP */
*IMG0 = 00; /*IMG0 initialized to 0 */

IMG1 = (int *) (CSIOP + 0x92); /*IMG1 offset from CSIOP */
*IMG1 = 00; /*IMG1 initialized to 0 */

IMG2 = (int *) (CSIOP + 0x94); /*IMG2 offset from CSIOP */
*IMG2 = 00; /*IMG2 initialized to 0 */

IMG3 = (int *) (CSIOP + 0x96); /*IMG3 offset from CSIOP */
*IMG3 = 00; /*IMG3 initialized to 0 */

/*Scaling of Clock input, common to all Counter/Timers */

dlyc_reg = (char *) (CSIOP + 0xa6); /*dlyc reg offset from CSIOP */
*dlyc_reg = 00; /*Also Scale-bit is 0 */
 /*Refer to Timer Clock Initialization in the
 App note for more details*/

/* Now any one of the following subroutines pulse(), waveform(), event_count(),
time_capture() and watchdog() can be called */
}

/*****

```

## Appendix 4. .C File (Cont.)

```

waveform()
{

/*Loading of Command Register-0 (CMD0) */

CMD0 = (char *) (CSIOP + 0xA0); /*CMD0 register offset from CSIOP*/
*CMD0 = 0xd4;

/* Write "D4"hex to Command Register-0 (CMD0) at offset from base address of CSIOP
(Chip Select I/O Port).

```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 1     | X     | X     | 0     | 1     | 0     | 0     |

### CMD0 Register

- **bit-0:** 0 Mode Select Bit, select Waveform Mode for CTU0.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU0 counts down from 2 to 0. At 0, TC triggers the loading and operation of the CTU1).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** 0 Output polarity: Select output to be active low (Toff time).
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** X Pin or Macrocell input: No pin or macrocell input, don't care.
- **bit-6:** 1 Load/Store Bit: No pin or macrocell load/store.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

```

*/

```

```

/*Loading of Command Register 1 (CMD1) */

```

```

CMD1 = (char *) (CSIOP + 0xA1); /*CMD1 register offset from CSIOP*/
*CMD1 = 0xCC;

```

```

/*

```

```

Write "CC"hex to Command Register 1 (CMD1).

```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 1     | X     | X     | 1     | 1     | 0     | 0     |

### CMD1 Register

- **bit-0:** 0 Mode Select Bit, select Waveform Mode for CTU1.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU1 counts down from 4 to 0. At 0, TC triggers the loading and operation of the CTU0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU1.
- **bit-3:** 1 Output polarity: Select output to be active on (Ton time).
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** X Pin or Macrocell input: No pin or macrocell input, don't care.
- **bit-6:** 1 Load/Store Bit: No pin or macrocell load/store.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

```

*/

```

**Appendix 4.**  
**.C File**  
**(Cont.)**

**/\*\*\*\*\*\* Image Registers Loading \*\*\*\*\*/**

```
IMG0 = (int *) (CSIOP + 0x90); /* Load the Counter/Timer_0 with necessary */
IMG0 = 0x0004; / off-time needed according to the duty cycle*/
```

```
IMG1 = (int *) (CSIOP + 0x92); /* Load the Counter/Timer_1 with necessary */
IMG1 = 0x0002; / on-time needed according to the duty cycle*/
```

/\*Configure portA output pin in Special Function Out \*/

```
PORTA = (char *) (CSIOP + 0x08); /* Get Special Function Register of PORT A */
PORTA = 0x0001; / Activate pin PA0 as PWM_OUT.
Timer-0 and Timer-1 are internally connected */
```

/\*A waveform output is first initialized by setting its two corresponding software Load/Store bits after loading the image registers \*/

```
SOFTWARE_LD_ST = (char *) (CSIOP + 0xA5);
*SOFTWARE_LD_ST = 0x0003;
```

```
/*** Global register configuration *****/
GLOB = (char *) (CSIOP + 0xA8);
*GLOB = 0x02;
```

/\*

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     |

**Global Command Register**

- **bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).
- **bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
- **bit-2:** 0 Global Mode bit: All Counter/Timers operate in Waveform or Pulse Mode.
- **bit-3:** 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer 2).

\*/

/\*When the Global Command Register is written to the Counter/Timers start decrementing and at underflow of each timer corresponding waveforms can be noticed on port A at PWM\_OUT.\*/

```
printf("waveform program");
}
/******
```

## Appendix 4. .C File (Cont.)

```

pulse()
{
 /* Loading of Counter/Timer-0 i.e Command Register 0 (CMD0) */

 CMD0 = (char*)(CSIOP + 0xa0); /* CMD0 register offset from CSIOP */
 *CMD0 = 0x9D;
 /*

```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 0     | 0     | X     | 1     | 1     | 0     | 1     |

### CMD0 Register

- **bit-0:** 1 Mode Select Bit, select Pulse Mode for CTU0.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU0 counts down from 3 to 0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** 1 Output polarity: Select output pulse to be active high.
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Enable Load control by macrocell output.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

```
*/
```

```

/***** Image Register Loading *****/
/* Load the Counter/Timer_0 with necessary count i.e Pulsewidth needed */

```

```

IMG0 = (int*)(CSIOP + 0x90);
*IMG0 = 0x0003;

```

```

/* Configure portA output as Special Function Out so that Timer0 o/p is on PA0 as pulse0_out */

```

```

PORTA = (char*)(CSIOP + 0x08); /* Get Special Function Register of PORT A */
PORTA = 0x0001; / Activate pin PA0 as Timer-0 output */

```

**Appendix 4.**  
**.C File**  
**(Cont.)**

/\*\* Global register configuration \*\*\*\*\*/

GLOB = (char \*) (CSIOP + 0xa8);  
 \*GLOB = 0x02;  
 /\*

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     |

**Global Command Register**

- **bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).
  - **bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
  - **bit-2:** 0 Global Mode bit: All Counter/Timers operate in Waveform or Pulse Mode.
  - **bit-3:** 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer 2).
- \*/

/\*Now, if the conditions setup in PPLD for mc2tmr0 are satisfied, a pulse of pulsewidth = 03 can be noticed on port A PA0 (pulse0\_out) until Counter/Timer-0 underflows .\*/

```
printf("pulsewidth program");
}
```

/\*\*/

## Appendix 4. .C File (Cont.)

```

event_count()
{
/*Loading of Counter/Timer-0 i.e Command Register 0 (CMD0) */

CMD0 = (char *) (CSIOP + 0xa0); /*CMD0 register offset from CSIOP */
*CMD0 = 0x1E;
/*
Loading of Command Register 0 (CMD0) at offset A0(hex) from CSIOP1.

```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | X     | X     | 1     | 1     | 0     |

### CMD0 Register

- **bit-0:** 0 Mode Select Bit, select Event Count Mode for CTU0.
- **bit-1:** 1 Decrement/Increment Bit: Increment after every event.
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** X Output polarity: No timer output, don't care.
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Store control from macrocell.
- **bit-7:** 0 EN/DIS Bit: Enable or disable by macrocell.

```
*/
```

```

/***** Image Register Loading *****/
/* Initialize IMG0 at 0000 */

```

```

IMG0 = (int *) (CSIOP + 0x90);
*IMG0 = 0x0000;

```

```

/*** Global register configuration *****/

```

```

GLOB = (char *) (CSIOP + 0xa8);
*GLOB = 0x06;
/*

```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 1     | 1     | 0     |

### Global Command Register

- **bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between "divide by 8" or "divide by 1").
- **bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
- **bit-2:** 1 Global Mode bit: All Counter/Timers operate in Event Count or Time Capture Mode.
- **bit-3:** 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer 2).

```
*/
```

<sup>1</sup>CSIOP is Chip Select of the Input/Output Port.



**Appendix 4.**  
**.C File**  
**(Cont.)**

```
/* Now, if the conditions setup in PPLD for mc2tmr0 are satisfied as specified in the Abel
software, at every transition on mc2tmr0 Counter-0 increments its count */

/* Freeze Command Register*/
/* If the Event count value in the counter needs to be read, the count updates to the image
register have to be frozen*/

/* To Freeze updates to IMG0 Register, set bit-0 of Freeze Command Register to "1" */
FREEZE_CMD = (char*)(CSIOP + 0xa4);
*FREEZE_CMD = 0x01;

/* Freeze acknowledge Register*/
/* Wait for the freeze acknowledge bit to be set and then proceed to read the Event count
value stored in the image register*/

STATUS_FLAGS = (char*)(CSIOP + 0xa9);
while (((*STATUS_FLAGS) & 0x01) == 0x01); /* FREEZE ACKNOWLEDGE BIT 0 */
printf("Image_0 register = %x", (*IMG0));

/* The FREEZE CMD bit 0 of Counter/Timer0 must be cleared to 0 and set back to 1, if
another IMG0 Register reading needs to be done*/

FREEZE_CMD = (char*)(CSIOP + 0xa4);
*FREEZE_CMD = 0x00;

printf("Event Count program");
}
/*****/
```

## Appendix 4. .C File (Cont.)

```
time_capture()
{
/*Loading of Counter/Timer 0 i.e Command Register 0 (CMD0) */

CMD0 = (char *) (CSIOP + 0xa0); /*CMD0 register offset from CSIOP*/
*CMD0 = 0xBF;
/*
Loading of Command Register-0 (CMD0) at offset A0(hex) from 2CSIOP.
```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| X     | 0     | 1     | 1     | X     | 1     | 1     | 1     |

### CMD0 Register

- **bit-0:** 1 Mode Select Bit, select Time Capture Mode for CTU0.
- **bit-1:** X Decrement/Increment Bit: Increment mode.
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** X Output polarity: No timer output, don't care.
- **bit-4:** 1 Input Polarity: Active low.
- **bit-5:** 1 Pin or Macrocell input: Pin input control.
- **bit-6:** 0 Load/Store Bit: Store control from pin.
- **bit-7:** X EN/DIS Bit: Don't care, setting of bit-2 enables the Counter.

```
*/
```

```
/***** Image Register Clearing *****/
```

```
IMG0 = (int *) (CSIOP + 0x90);
*IMG0 = 0x0000;
```

```
/*Loading of Counter/Timer-1 i.e Command Register 1 (CMD1) */
```

```
CMD1 = (char *) (CSIOP + 0xa1); /*CMD1 register offset from CSIOP*/
*CMD1 = 0x9f;
/*
```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| X     | 0     | 0     | 1     | X     | 1     | 1     | 1     |

### CMD1 Register

- **bit-0:** 1 Mode Select Bit, select Time Capture Mode for CTU0.
- **bit-1:** 1 Decrement/Increment Bit: Increment mode.
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** X Output polarity: No timer output, don't care.
- **bit-4:** 1 Input Polarity: Active low.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Store control from pin.
- **bit-7:** X EN/DIS Bit: Don't care, setting of bit-2 enables the Counter.

```
*/
```

**Appendix 4.**  
**.C File**  
**(Cont.)**

/\*\*\*\*\* Image Register Clearing \*\*\*\*\*/

```
IMG1 = (int *) (CSIOP + 0x92);
*IMG1 = 0x0000;
```

/\*\*\* Global register configuration \*\*\*\*\*/

```
GLOB = (char *) (CSIOP + 0xa8);
*GLOB = 0x06;
/*
```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 1     | 1     | 0     |

**Global Command Register**

- **bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between “divide by 8” or “divide by 1”).
- **bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
- **bit-2:** 1 Global Mode bit: All Counter/Timers operate in Event Count or Time Capture Mode.
- **bit-3:** 0 Watch Dog bit: Not Watchdog Mode (affects only Counter/Timer 2).

\*/

/\*Now, if the conditions setup in PPLD for mc2tmr1 are satisfied as specified in the Abel software , at every rising edge transition on input pin PE3 and mc2tmr1, Counter-0 and Counter-1 transfer their count values into the image register-0 and image register-1\*/

/\*Freeze Command Register\*/

/\*If Time Capture value in the Image register need to be read, the count updates to the image register have to be frozen \*/

/\* To freeze updates to IMG0, set bit-0 of Freeze Command Register to "1" \*/

```
FREEZE_CMD = (char *) (CSIOP + 0xa4);
```

```
FREEZE_CMD = 0X01; / A high going signal of write freezes the image updates */
```

/\*Freeze acknowledge Register\*/

/\*Wait for the freeze acknowledge bit to be set and then proceed to read the Time Capture value stored in the image register\*/

```
STATUS_FLAGS = (char *) (CSIOP + 0xa9);
```

## Appendix 4. .C File (Cont.)

```

/*Microcontroller loops around the Freeze Acknowledge bit 0: if it's set to 1, then it
proceeds to read the image-0 register for time-capture value */

 while(((*STATUS_FLAGS) & 0x01) == 0X01);
 printf("Image_reg_0=%x\n", (*IMG0));

/*The FREEZE CMD bit 0 of Counter/Timer-0 must be cleared to 0 and set back to 1 if
another IMG0 Register reading needs to be done */

FREEZE_CMD = (char *) (CSIOP + 0xa4);
*FREEZE_CMD = 0x00;

/*Now to read the IMG1 Register */
*FREEZE_CMD = 0X02; /*A high going signal of write freezes the image updates */

/*Freeze acknowledge Register */
/*Wait for the freeze acknowledge bit to be set and then proceed to read the Time Capture
value stored in the image register */

STATUS_FLAGS = (char *) (CSIOP + 0xa9);

/*Microcontroller loops around the Freeze Acknowledge bit 1: if it's set to 1, then it
proceeds to read the image-1 register for time-capture value */

 while (((*STATUS_FLAGS) & 0x02) == 0X02);
 printf("Image_reg_1%x =\n", (*IMG1));

/*The FREEZE CMD bit 1of Counter/Timer1 must be cleared to 0 and set back to 1 if
another IMG1 Register reading needs to be done */

FREEZE_CMD = (char *) (CSIOP + 0xa4);
*FREEZE_CMD = 0x00;

/*From captured values in image registers pulsewidth can be computed.*/
printf("Time_capture program");
}

/*****/

```

**Appendix 4.**  
**.C File**  
**(Cont.)**

```

watchdog()
{
/*Ignore Counter/Timer-2 Command Register 2 (CMD2) */

/***** Image Register Loading *****/

IMG2 = (int *) (CSIOP + 0x94);
*IMG2 = 0x0002;

/*Writing into SOFTWARE LOAD reg to load Counter_2 */

SOFTWARE_LD_ST = (char *) (CSIOP + 0xa5);
SOFTWARE_LD_ST = 0x04;

/*** Global register configuration *****/

GLOB = (char *) (CSIOP + 0xa8);
*GLOB = 0x0a;
/*

```

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 1     | X     | 1     | 0     |

**Global Command Register**

- **bit-0:** 0 Scale Bit: The clock input to the Timers is divided by 1. This is the first clock pre-scaler stage (selecting between "divide by 8" or "divide by 1").
- **bit-1:** 1 Counter start bit: This bit turns on all the selected Timers.
- **bit-2:** X Global Mode bit: Watchdog is available in both Global modes.
- **bit-3:** 1 Watch Dog bit: Watchdog Mode (affects only Counter/Timer-2).

```

*/

/*As soon as the WatchDog bit in the Global Command Register is set to one,
Counter/Timer-2 starts decrementing. If a software load command is not executed before
Counter-2 underflows, watchdog condition occurs */

printf("watchdog program");
}
/*****

```



## Appendix 5. .ASM File

\*This is the common initialization of WSI's PSD5XX timers  
\*interfaced with Motorola's 68HC11, using 68HC11 Assembly.

### \*Memory Map

```
*
*
* EPROM(1) a000 - ffff (Program PSD5XX)
* RAM 4000 - 4fff (RAM PSD5XX)
* I/O 3000 - 3fff (CSIOP PSD5XX)
* EPROM(2) 5000 - 6fff (Data PSD5XX)
* RAM 1000 - 10ff (68HC11)
* I/O & REG 0000 - 003f (68HC11)
```

```
 org $a000 Program Memory

init equ $103d RAM & I/O mapping reg (68HC11)
csiop equ $3000 chip select i/o port addr (PSD5XX)
stack equ $10ff stack area
stor equ $1002 basic RAM storage area
```

### \*\*\*\*\* Begin main program \*\*\*\*\*

```
start sei Set IRQ mask
 ldaa #010h set RAM at 1000
 staa init and set registers at 103d

 nop slight delay to allow registers setup
 ldaa #0e3h setup option reg.-ADPU = 1,CSEW = 1,IRQE = 1

 lds #stack setup stack
```

### \*\*\*\*\* Initialize PSD5XX COUNTER/TIMER Counter and Image Registers\*\*\*\*\*

```
ldx #0
stx csiop+98h clear counter/timer 0 CNTR0(low byte)
stx csiop+99h clear counter/timer 0 CNTR0(high byte)
stx csiop+9ah clear counter/timer 1 CNTR1(low byte)
stx csiop+9bh clear counter/timer 1 CNTR1(high byte)
stx csiop+9ch clear counter/timer 2 CNTR2(low byte)
stx csiop+9dh clear counter/timer 2 CNTR2(high byte)
stx csiop+9eh clear counter/timer 3 CNTR3(low byte)
stx csiop+9fh clear counter/timer 3 CNTR3(high byte)

stx csiop+90h clear counter/timer 0 IMG0(low byte)
stx csiop+91h clear counter/timer 0 IMG0(high byte)
stx csiop+92h clear counter/timer 1 IMG1(low byte)
stx csiop+93h clear counter/timer 1 IMG1(high byte)
stx csiop+94h clear counter/timer 2 IMG2(low byte)
stx csiop+95h clear counter/timer 2 IMG2(high byte)
stx csiop+96h clear counter/timer 3 IMG3(low byte)
stx csiop+97h clear counter/timer 3 IMG3(high byte)
```

### \*\*\*\*\*Scaling of clock input, common to all counter/timers\*\*\*\*

```
ldx #0000h
stx csiop+$a6 dicy = 0 (delay cycle reg)
*
* Regarding scale-bit, it'll be set in global reg
```

\*\*\*\*\*END OF COMMON INITIALIZATION \*\*\*\*\*

**Appendix 5.**  
**.ASM File**  
**(Cont.)**

\*BASED ON THE MODE OF OPERATION THE FOLLOWING ROUTINES CAN BE USED\*

\*This is the implementation of "WAVEFORM MODE" of operation of WSI's PSD5XX  
\*interfaced with Motorola's 68HC11

\*\*\*\*\*Counter/Timer 0 initialization (Command Reg 0 i.e. CMD0)

```
 ldx #00d4h
 stx csiop+$a0 Waveform mode
*
* Decrement mode
* select counter/timer
* output pulse active low
* input polarity on input pin (doesn't matter here)
* input control not from PPLD or PIN, don't care
* load/store by software
* enable continuous counting
*
```

\*\*\*\*\*Counter/timer 1 initialization (Command Reg 1 i.e. CMD1)

```
 ldx #00cch
 stx csiop+$a1 Waveform mode
*
* Decrement mode
* select counter/timer
* output pulse active high
* input polarity on input pin (doesn't matter here)
* input control not from PPLD or PIN, don't care
* load/store by software
* enable continuous counting
*
```

\*\*\*\*\*Image reg (IMG0) loading\*\*\*\*\*

```
 ldx #0004h load counter/timer0 image reg with necessary
 stx csiop+90h count i.e. pulse width needed (off time)
 ldx #0000h
 stx csiop+91h
```

\*\*\*\*\* image reg(IMG1) loading\*\*\*\*\*

```
 ldx #0002h load counter/timer1 image reg with necessary
 stx csiop+92h count i.e. pulse width needed (on time)
 ldx #0000h
 stx csiop+93h
```

**Appendix 5.**  
**.ASM File**  
**(Cont.)**

\*\*A waveform output is first initialized by setting its two corresponding\*\*  
 \*\*software load/store bits after loading the image registers \*\*\*

```
ldx #0003h
stx csiop+$a5
```

\*\*Configure port A output as special function so that waveform output is available at PA0\*\*

```
ldx #0001h Activate pin PA0 as waveform output
stx csiop+08h special function reg of port A
```

\*\*\*\*\* global register configuration \*\*\*\*\*

```
ldx #0002h
stx csiop+$a8 non-watchdog mode
* Waveform mode/pulse
* all ctus enabled
* Scale bit 0
```

\*The counters are always enabled by the software and a waveform

\*can be noticed on port A PA0.

\*PWM pulse widths equal to count of 04 loaded in the image-0 register and 02 in IMG1.

\*\*\*\*\* END OF WAVEFORM MODE \*\*\*\*\*



**Appendix 5.**  
**.ASM File**  
(Cont.)

\* This is the implementation of "PULSE MODE" of operation of WSI's PSD5XX  
\* interfaced with Motorola's 68HC11.

\*\*\*\*\* Counter/Timer 0 initialization (Command Reg 0 i.e. CMD0)

```
 ldx #009dh
 stx csiop+$a0 pulse mode
* decrement mode
* select counter/timer
* output pulse active high
* input polarity on input pin(doesn't matter, here it's mcell)
* input control from PPLD (not PIN)
* load control activated by macrocell output
* enable continuous counting
*
```

\*\*\*\*\* image-0 reg loading \*\*\*\*\*

```
 ldx #0003h load counter/timer0 image reg with necessary
 stx csiop+90h count i.e. pulse width needed
 ldx #0000h
 stx csiop+91h
```

\*\*\* Configure port A output as special function so that Timer0 output is available at PA0 \*\*\*

```
 ldx #0001h Activate pin PA0 as Timer0 output
 stx csiop+08h special function reg of port A
```

\*\*\*\*\* global register configuration \*\*\*\*\*

```
 ldx #0002h
 stx csiop+$a8 non-watchdog mode
* Waveform/pulse mode
* all ctus enabled
* Scale bit 0
```

\*Appropriate signals on mc2tmr0 starts the counter and a pulse

\*can be noticed on port A PA0 (based on the .abl equation) till Counter/timer-0 under flows .

\*\*\*\*\* END OF PULSE MODE \*\*\*\*\*

## Appendix 5. .ASM File (Cont.)

\* This is the implementation of "EVENT MODE" of operation of WSI's PSD5XX  
\* interfaced with Motorola's 68HC11

\*\*\*\*\*Counter/Timer 0 initialization (Command Reg 0 i.e. CMD0)

```

ldx #1eh
stx csiop+$a0 Event Count mode
*
* increment mode
* select counter/timer
* Counter/Timer output (don't care)
* input polarity on input pin(doesn't matter not PIN)
* input control from PPLD (not PIN)
* Store control activated by macrocell
* enable activated by macrocell

```

\* Image and Counter Registers are initialized to "00" in the beginning

\*\*\*\*\*global register configuration\*\*\*\*\*

```

ldx #0002h
stx csiop+$a8 non-watchdog mode
*
* Event Count/Time Capture mode
* all ctus enabled
* Scale bit 0

```

\* Now, if the conditions setup in PPLD for mc2tmr0 are satisfied as specified  
\* in Abel software, at every rising transition on mc2tmr0 Counter-0 increments its count.  
\* and updates the IMG0 Register.

\*\*\*\*\*Freeze Command register\*\*\*\*\*

\*If the Event Count value in the Counter register need to be read, the count  
\*updates to the image register have to be frozen.

```

ldx #0001h a high going signal of write freezes IMG0 updates
stx csiop+$a4h

```

\* Now check if Freeze Acknowledge bit of Counter-0 is set and read IMG0 reg ev\_loop

```

ldx #(csiop+$a9h) freeze acknowledge bits(status flag reg)
ldaa #$f0 bit 0 for timer-0(upper 4 bits are 1's)
cmpa 0,x checking if it's set to 1
bne ev_loop

```

\* Read IMG0 reg

```

ldx #(csiop+$90h) low byte
ldx #(csiop+$91h) high byte

```

\*To do another read of the Event Count value, the Freeze Command Register bit 0 has to  
cleared to 0 and set back to 1, when necessary.

```

ldx #0000h a 0 clears the FREEZE CMD bit 0 to 0
stx csiop+$a4h

```

\*\*\*\*\*END OF EVENT COUNT MODE \*\*\*\*\*

**Appendix 5.**  
**.ASM File**  
**(Cont.)**

\* This is the implementation of "TIMER CAPTURE MODE" of operation of WSI's PSD5XX  
\* interfaced with Motorola's 68HC11

\*\*\*\*\* Counter/Timer 0 initialization (Command Reg 0 i.e. CMD0)

```
 ldx #00bfh
 stx csiop+$a0 Time-Capture mode
* increment mode(don't care)
* select counter/timer
* Counter/Timer output(don't care)
* Input polarity active low
* input control from PIN(not PPLD macrocell)
* Store control activated by pin
* en/dis bit don't care, select counter bit is enough
*
```

\* Image and Counter Registers are initialized to "00" in the beginning

\*\*\*\*\* Counter/timer 1 initialization (Command Reg 1 i.e. CMD1)

```
 ldx #009fh
 stx csiop+$a1 Time-Capture mode
* increment mode
* select counter/timer
* Counter/Timer output (don't care)
* input polarity on input pin (doesn't matter)
* input control from PPLD (not PIN)
* Store control activated by macrocell
* en/dis bit don't care, select counter bit is enough
*
```

\* Image and Counter Registers are initialized to "00" in the beginning

\*\*\*\*\* global register configuration\*\*\*\*\*

```
 ldx #0006h
 stx csiop+$a8 non-watchdog mode
* Event Count/Time Capture mode
* all ctus enabled
* Scale bit 0
```

\* Now, if the conditions setup on PE3 and in PPLD for mc2tmr1 are satisfied as specified in  
\* Abel software, at every transition on PE3 and mc2tmr1, the Counter-0 transfers its count  
\* value into the image register-0 and the Counter-1 to IMG1.

\* Freeze Command register bit 0 for Counter/Timer-0

\* If the Time Capture value in the image register needs to be read, the count  
\* updates to the image register have to be frozen.

```
 ldx #0001h a high going signal of write freezes IMG0 updates
 stx csiop+$a4h
```

## Appendix 5. .ASM File (Cont.)

\*Now check if Freeze Acknowledge bit of Counter-0 is set and read IMG0 reg tc0\_loop

```
ldx #(csiop+$a9h) freeze acknowledge bits(status flag reg)
ldaa #$f0 bit 0 for timer-0(upper 4 bits are 1's)
cmpa 0,x checking if it's set to 1
bne ev_loop
```

\*Read IMG0 reg

```
ldx #(csiop+$90h) low byte
ldx #(csiop+$91h) high byte
```

\*To do another read of the Time Capture value, the Freeze Command Register bit 0

\*has to cleared to 0 and set back to 1, when necessary.

```
ldx #0000h a 0 clears the FREEZE CMD bit 0 to 0
stx csiop+$a4h
```

\*Freeze Command register bit 1 for Counter/Timer-1

\*If the Time Capture value in the image register needs to be read, the count

\*updates to the image register have to be frozen.

```
ldx #0002h a high going signal of write freezes IMG1 updates
stx csiop+$a4h
```

\*Now check if Freeze Acknowledge bit of Counter-1 is set and read IMG1 reg tc1\_loop

```
ldx #(csiop+$a9h) freeze acknowledge bits (status flag reg)
ldaa #$f1 bit 1 for timer-1 (upper 4 bits are 1's)
cmpa 0,x checking if it's set to 1
bne tc1_loop
```

\*Read IMG1 reg

```
ldx #(csiop+$92h) low byte
ldx #(csiop+$93h) high byte
```

\*To do another read of the Time Capture value, the Freeze Command Register bit 1

\*has to cleared 0 and set back to 1, when necessary.

```
ldx #0000h a 0 clears the FREEZE CMD bit 1 to 0
stx csiop+$a4h
```

\*\*\*\*\* END OF TIME CAPTURE MODE \*\*\*\*\*

**Appendix 5.**  
**.ASM File**  
**(Cont.)**

\*This is the implementation of "WATCHDOG MODE" of operation of WSI's PSD5XX  
\*interfaced with MOTOROLA's 68HC11

\*\*\*\*\* Counter/Timer 2 initialization

\*\*\*\*\* Ignore (Command Reg 02ie CMD2)

\*\*\*\*\* image reg (IMG2)loading\*\*\*\*\*

```
ldx #0002h load counter/timer2 image reg with necessary
stx csiop+94h count i.e. pulse width needed
ldx #0000h
stx csiop+95h
```

\*\* Writing into SOFTWARE LOAD reg to load Counter-2 \*\*\*\*\*

```
ldx #0004h Software load for counter-2
stx csiop+$a5h
```

\*\*\*\*\* global register configuration \*\*\*\*\*

```
ldx #000ah
stx csiop+$a8 watchdog mode
* Waveform/pulse mode (don't care)
* all ctus enabled
* Scale bit 0
```

\*\*\* As soon as the WatchDog bit in the Global Command Register is set to one,  
\* Counter/Timer-2 starts decrementing. If a write into bit-2 of software load register is not  
\* done before Counter-2 under flows, a watchdog condition occurs.

```
loop nop
 jmp loop end of main loop
```

\*\*\*\*\* END OF WATCHDOG MODE \*\*\*\*\*

## Appendix 6. Realizing 4 PWM Timers on the PSD5XX

### Abstract

The PSD5XX has four timers. The Waveform mode or the PWM mode needs two timers, i.e. one to drive the ON time and second one to drive the OFF time. Therefore a maximum of two PWM timer sets can be realized on the PSD5XX. Using the resources of the PPLD however, effectively four PWM timers can be realized. This application brief explains the procedure to "realize 4 PWM timers on the PSD5XX".

### The PPLD method for realizing 4 PWM Timers

In the normal pulse mode operation, a Counter/Timer outputs a mono-shot pulse of pulsewidth equal to the value loaded into its associated image register. After the pulse is output the Counter/Timer waits for another load signal to output another pulse and so on. The procedure used here to generate 4 PWM timers is as follows:

- Set all the four timers into the "pulse mode" of operation.
- Use the "PWM\_CYCLE" signal which is described below, to generate the LOAD signal for Counter/Timer 0, 1, 2 and 3.
- Counter/Timer Image Registers (IMG0,IMG1,IMG2,IMG3) can be loaded with different "OFF" or "ON" time values based on the mode set up in CMD Registers.
- Regarding the duty cycle calculation of the resulting PWM waveform, refer to the section "Duty cycle calculation".

### The PWM\_CYCLE Signal

In this application note example the period of the output PWM waveforms generated by the Counter/Timers operating in the PULSE mode is controlled by the PWM\_CYCLE signal.

The PWM\_CYCLE signal in this example has been generated internally in the PSD5XX itself as a GPLD macrocell output. Here the period of the PWM\_CYCLE signal is equal to the PSD5XX clock input divided by 32.

How the generation of the PWM\_CYCLE signal is done is up to the user, typically it is generated externally with longer period.

Input the PWM\_CYCLE signal into the Counter/Timer macrocell input, which is the input control for loading the Counter/Timer Registers from the corresponding Image Registers. This can be done in the .ABL file using the following PPLD equation:

```
mc2tmrX := PWM_CYCLE.fb ;
```

where X = 0, 1, 2, 3 for the Counter/Timers 0,1,2 and 3 and all Counter/Timers are operating in PULSE mode.

**Note:** In case PWM\_CYCLE is an external input signal, the expression for mc2tmrX will be:

```
mc2tmrX := PWM_CYCLE ;
```

## Appendix 6. Realizing 4 PWM Timers on the PSD5XX (Cont.)

### Counter/Timer Registers set-up procedure to realize the 4 PWM Timers

#### CMD0, CMD1, CMD2 and CMD3 Registers Initialization

- Write "95"hex to Command Register 0 (CMD0) at offset from base address of CSIOP (Chip Select I/O Port).

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 0     | 0     | X     | 0     | 1     | 0     | 1     |

#### CMD0 Register

- **bit-0:** 1 Mode Select Bit, select Pulse Mode for CTU0.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU0 counts down from 1 to 0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU0.
- **bit-3:** 0 Output polarity: Select output to be active low.
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Load control from macrocell.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

- Write "9D"hex to Command Register 1 (CMD1) at offset from base address of CSIOP (Chip Select I/O Port).

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 0     | 0     | X     | 1     | 1     | 0     | 1     |

#### CMD1 Register

- **bit-0:** 1 Mode Select Bit, select Pulse Mode for CTU1.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU1 counts down from 2 to 0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU1.
- **bit-3:** 1 Output polarity: Select output to be active high.
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Load control from macrocell.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

- Write "95"hex to Command Register 2 (CMD2) at offset from base address of CSIOP (Chip Select I/O Port).

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 0     | 0     | X     | 0     | 1     | 0     | 1     |

#### CMD2 Register

- **bit-0:** 1 Mode Select Bit, select Pulse Mode for CTU2.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU2 counts down from 3 to 0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU2.
- **bit-3:** 0 Output polarity: Select output to be active low.
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Load control from macrocell.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

## Appendix 6. Realizing 4 PWM Timers on the PSD5XX (Cont.)

- Write "9D" hex to Command Register 3 (CMD3) at offset from base address of CSIOP (Chip Select I/O Port).

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | 0     | 0     | X     | 1     | 1     | 0     | 1     |

### CMD3 Register

- **bit-0:** 1 Mode Select Bit, select Pulse Mode for CTU3.
- **bit-1:** 0 Decrement/Increment Bit: Select decrement (CTU3 counts down from 4 to 0).
- **bit-2:** 1 Select Counter/Timer Bit: Select CTU3.
- **bit-3:** 1 Output polarity: Select output to be active high.
- **bit-4:** X Input Polarity: No pin input in this mode, don't care.
- **bit-5:** 0 Pin or Macrocell input: Macrocell input control.
- **bit-6:** 0 Load/Store Bit: Load control from macrocell.
- **bit-7:** 1 EN/DIS Bit: Enable continuous counting.

### Image Registers loading:

- IMG0 Register is loaded with a value 01 to define pulse width (**off time**).
- IMG1 Register is loaded with a value 02 to define pulse width (**on time**).
- IMG2 Register is loaded with a value 03 to define pulse width (**off time**).
- IMG3 Register is loaded with a value 04 to define pulse width (**on time**).

After the Command Registers and Image Registers are initialized, the Registers common to all the Counter/Timers(Special function Register, Global Command Register) are initialized.

- Write "0F" to Port A Special Function Register. This specifies pins PA0,PA1,PA2 and PA3 as pulse output pins.

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 1     | 1     | 1     | 1     |

### Special Function Register

- Now to start the Counter/Timers: write "02" hex to the Global Command Register.

| Bit-7 | Bit-6 | Bit-5 | Bit-4 | Bit-3 | Bit-2 | Bit-1 | Bit-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 0     | 0     | 0     | 0     | 1     | 0     |

### Global Command Register

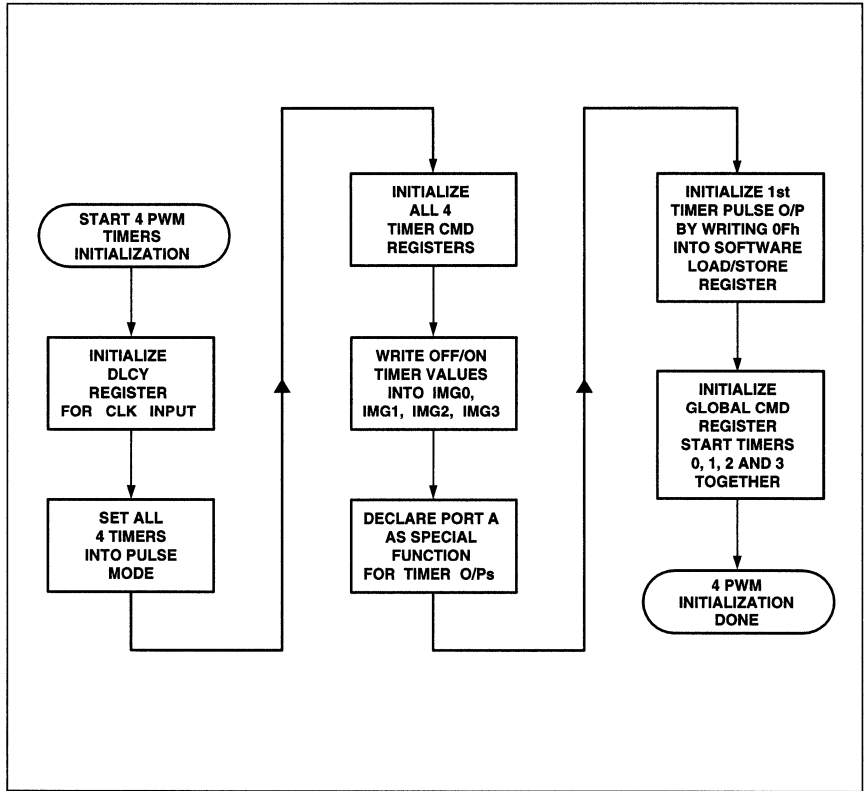
- **bit-0:** 0 Scale Bit: Clock input to Timers is divided by 1.
- **bit-1:** 1 Counter start bit: This bit turns on the Timer operation.
- **bit-2:** 0 Global Mode bit: All Counter/Timers operate in the Waveform or the Pulse Mode.
- **bit-3:** 0 Watch Dog bit: Not Watchdog Mode.

Figure 20 illustrates the flow chart for initialization procedure to realize 4 PWM timers on PSD5XX.



**Appendix 6.**  
**Realizing**  
**4 PWM**  
**Timers on the**  
**PSD5XX**  
*(Cont.)*

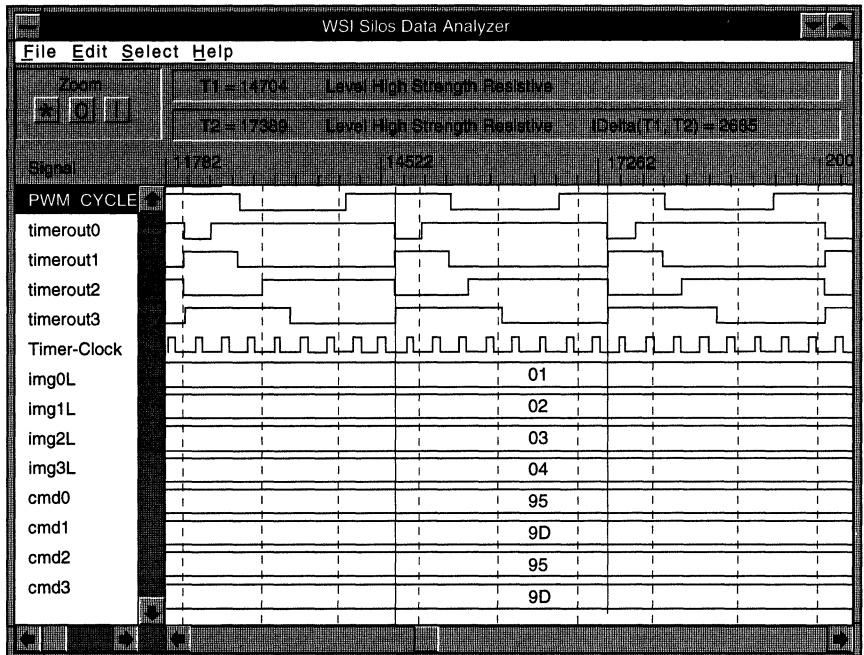
**Figure 20. PSD5XX Initialization To Generate 4 PWM Timers**



**Appendix 6.**  
**Realizing**  
**4 PWM**  
**Timers on the**  
**PSD5XX**  
**(Cont.)**

As illustrated in the following Figure 21, the PWM\_CYCLE signal is used as the Counter/Timer load control signal. The period of the PWM\_CYCLE is the period of the PWM waveforms.

**Figure 21.**



3

Notice in Figure 21, the Image Registers are loaded as follows:

Image Register0 = 01hex and the output of this Counter/Timer0 is **timerout0(on time)**  
 Image Register1 = 02hex and the output of this Counter/Timer1 is **timerout1(off time)**  
 Image Register2 = 03hex and the output of this Counter/Timer2 is **timerout2(on time)**  
 Image Register3 = 04hex and the output of this Counter/Timer3 is **timerout3(off time)**

The Command Registers are loaded as follows:

CMD0 = 95hex  
 CMD1 = 9Dhex  
 CMD2 = 95hex  
 CMD3 = 9Dhex

**Appendix 6.  
Realizing  
4 PWM  
Timers on the  
PSD5XX  
(Cont.)**

**Duty Cycle Calculation**

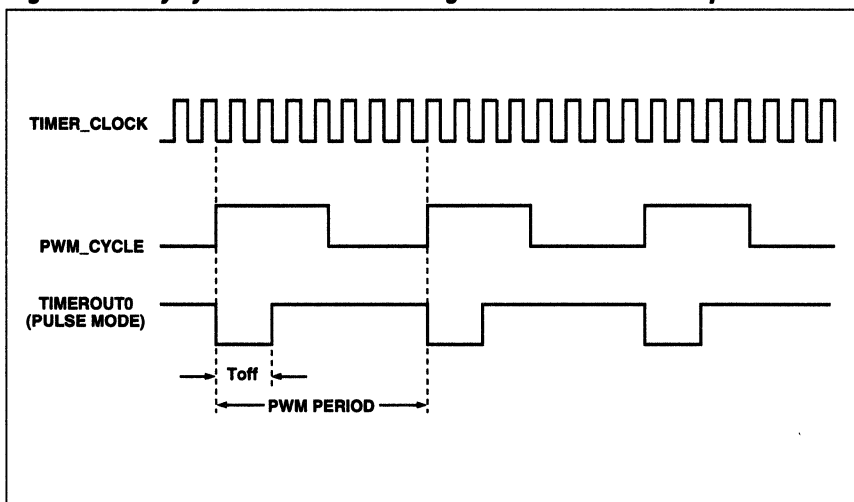
Figure 22 explains the duty cycle calculations. The "PWM PERIOD" and "off" time of the PWM waveform are known. To compute the "on" time of the PWM waveform and hence the duty cycle:

$$T_{on} = (\text{PWM PERIOD}) - T_{off}$$

$$\text{PWM Duty Cycle} = \frac{T_{on}}{\text{PWM PERIOD}}$$

This calculation is applicable to Counter/Timers 0 and 2 whose outputs are programmed to be active low. For Counter/Timers 1 and 3 the outputs are programmed to be active high, hence the "on" time is the value directly loaded into the related Image Registers multiplied by the Timer Clock unit.

**Figure 22. Duty Cycle Period of PWM Using PPLD Macrocell Technique**



**Conclusion**

This Application Brief explained how PSD5XX PPLD macrocells could be used in combination with timers to generate 4 PWM timers. Relevant PSDlabel file, Configuration File are enclosed.

**Appendix 6.**  
**Realizing**  
**4 PWM**  
**Timers on the**  
**PSD5XX**  
**(Cont.)**

```

This is the .abl file required to do the 4-PWM timers simulation

module gpld_pwm " 9-2-93

title '4 pwm channels';

"Input signals

cntout_en pin; "Enable counter outputs to drive out.
loadws pin; " Load and enable generator.
d4,d3,d2,d1,d0 pin; "Number of wait-states to load.

clkln, reset pin; "Default these signals are not needed to be defined

"Addr. lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

timer0_in,timer1_in,timer2_in,timer3_in pin;

" Internal PSD5XX PLD output signals.

csiop node ;
mc2tmr0, mc2tmr1, mc2tmr2, mc2tmr3 node;

X = .x. ; " Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

"Output signals

wstc pin ; " Wait-State Terminal Count.

PWM_CYCLE pin istype 'reg';
gpld_cnt3,gpld_cnt2,gpld_cnt1,gpld_cnt0 node istype 'reg'; " This counter outputs are
embedded.

equations

csiop = (Address >= ^hC000) & (Address <= ^hC0FF) ; " 256 block

@IF (0) {
Reset is available
there through the enable gated reset configuration bit.
}

[PWM_CYCLE].oe = cntout_en;

[PWM_CYCLE,gpld_cnt3,gpld_cnt2,gpld_cnt1,gpld_cnt0].re = reset;

wstc = !PWM_CYCLE.fb & !gpld_cnt3.fb & !gpld_cnt2.fb & !gpld_cnt1.fb & !gpld_cnt0.fb;

```

**Appendix 6.**  
**Realizing**  
**4 PWM**  
**Timers on the**  
**PSD5XX**  
**(Cont.)**

"my stuff to generate PWM\_CYCLE using GPLD

```
[gpld_cnt0].clk := clkkin;
gpld_cnt0 := !gpld_cnt0.fb;
[gpld_cnt1].clk := gpld_cnt0.fb;
gpld_cnt1 := !gpld_cnt1.fb;
[gpld_cnt2].clk := gpld_cnt1.fb;
gpld_cnt2 := !gpld_cnt2.fb;
[gpld_cnt3].clk := gpld_cnt2.fb;
gpld_cnt3 := !gpld_cnt3.fb;
[PWM_CYCLE].clk := gpld_cnt3.fb;
PWM_CYCLE := !PWM_CYCLE.fb;
"Pin counter/Timer control inputs.

mc2tmr0 := PWM_CYCLE.fb;
mc2tmr1 := PWM_CYCLE.fb;
mc2tmr2 := PWM_CYCLE.fb;
mc2tmr3 := PWM_CYCLE.fb;

END
```



# Programmable Peripheral Application Note 029 Interfacing PSD4XX/5XX To Microcontrollers

By Ravi Kumar

## Abstract

This application note is intended to give the reader a general guideline on how to interface PSD4XX/5XX Field Programmable Microcontroller Peripherals to specific microcontrollers. Relevant PSDabel files, bus simulation results and the PSD bus configurations of the interface examples are included in this application note.

The microcontrollers covered in this application note are:

- 80C31
- 80C161
- 80C196
- 68302
- 68332
- Z8/Z80
- 80C166
- ST9026
- NEURON® 3150™

## PSD4XX/5XX Architecture

The PSD4XX/5XX series provides the user with an innovative architecture for embedded applications. A PSD5XX device has the following features:

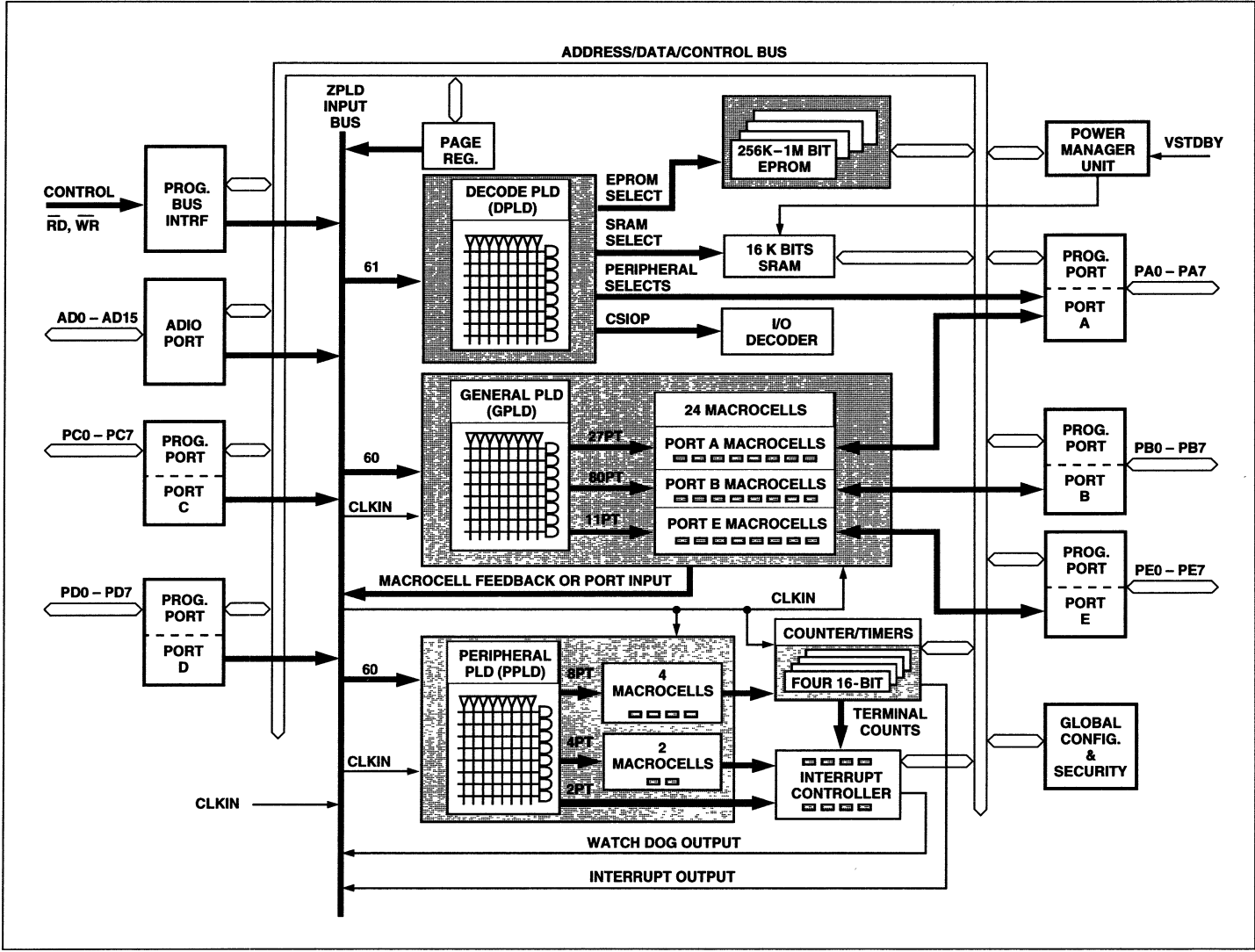
- Programmable bus interface, "no glue" logic interface to microcontrollers.
- Three ZPLDs (Zero Power PLDs) with a total of 61 inputs, 140 product terms outputs, 30 macrocells and 24 I/O pins.
- Forty individually programmable I/O pins that are divided into 5 ports.
- Four 16-bit Counter/Timers that perform pulse, waveform, time capture, event counting and watchdog functions.
- Eight input priority encoded Interrupt Controller. Four Interrupts are generated internally by Counter/Timers and the other four can be user defined through the ZPLD.
- 4-bit Page Register.
- Up to 1 Mbit Reprogrammable EPROM, consists of four 256 Kbit blocks.
- 16 Kbit of SRAM with battery backup mode.
- Power management unit with automatic power down and sleep modes.
- Security mode for code protection.

Figure 1 is a top level diagram of PSD4XX/5XX.

At the core of the PSD4XX/5XX are 3 dedicated ZPLDs:

- DPLD:** The Decoding ZPLD.  
Its main function is to perform address decoding for the internal I/O ports, EPROM, SRAM and peripheral mode of Port A.
- GPLD:** The General Purpose ZPLD.  
The user can implement state machines and other logic functions in the GPLD. It can also generate chip selects for external memories and peripheral devices.
- PPLD:** The Peripheral ZPLD.  
It provides additional control for the operation of the Counter/Timer Units and the Interrupt Controller. The PPLD is available only in the PSD5XX series.

Figure 1. PSD5XX Block Diagram



## The Bus Interface Of The PSD4XX/5XX

The PSD4XX/5XX have a user configurable bus interface. This interface can be configured to allow the PSD4XX/5XX to interface directly to most microcontrollers with "no glue" logic.

There are only five bus control pins on the PSD4XX/5XX. Each pin has multiple functions as shown in Table 1. For example, the "RD" pin can act as a "RD", or "E", or  $\overline{DS}$ , or  $\overline{LDS}$ , depending on the microcontroller bus interface. Please note the "RD" and "WR" pins are dedicated bus pins, but PE0 and PE1 are two general purpose I/O pins on port E. If the bus interface does not require these two pins, they can be configured to perform any of the other Port E functions.

The selection of these pin functions is implemented in the PSDconfiguration menu inside the PSDsoft.

**Table 1. Alternate Pin Functions**

| Pin Name | Pin Function 1   | Pin Function 2    | Pin Function 3   | Pin Function 4   | Pin Function 5 |
|----------|------------------|-------------------|------------------|------------------|----------------|
| RD       | $\overline{RD}$  | E                 | $\overline{DS}$  | $\overline{LDS}$ |                |
| WR       | $\overline{WR}$  | R/W               | $\overline{WRL}$ |                  |                |
| PE0      | $\overline{BHE}$ | $\overline{PSEN}$ | $\overline{WRH}$ | $\overline{UDS}$ | SIZ0           |
| PE1      | ALE              |                   |                  |                  |                |
| AD0      | A0               | BLE               |                  |                  |                |

3

The multiple functions of the PSD bus pins allow the PSD4XX/5XX to support a large number of microcontrollers. Table 2 shows some of these microcontroller families, the bus type and control signals associated with the microcontrollers.

**Table 2. Typical Microcontroller Bus Types**

| Multiplexed    | Data Bus Width | Bus Control Signals                                        | Microcontrollers                    |
|----------------|----------------|------------------------------------------------------------|-------------------------------------|
| Mux            | 8              | $\overline{WR}$ , $\overline{RD}$ , $\overline{PSEN}$      | 80C31 Family                        |
| Mux<br>Non-Mux | 8/16           | R/W, E, $\overline{BHE}$                                   | 68HC11 Family                       |
| Mux            | 8/16           | $\overline{WR}$ , $\overline{RD}$ , $\overline{BHE}$       | 80196/80186 Family<br>80C166 Family |
| Mux            | 16             | $\overline{WRL}$ , $\overline{RD}$ , $\overline{WRH}$      | 80196SP                             |
| Mux            | 8              | R/W, $\overline{DS}$                                       | ST9 Family                          |
| Non-Mux        | 16             | R/W, $\overline{LDS}$ , $\overline{UDS}$                   | 68302                               |
| Non-Mux        | 8/16           | R/W, $\overline{DS}$ , SIZ0                                | 683XX Family                        |
| Non-Mux        | 8/16           | R/W, $\overline{DS}$ , $\overline{BHE}$ , $\overline{BLE}$ | 68330                               |



**PSD4XX/5XX  
Interface  
To a  
Multiplexed  
Bus**

**PSD4XX/5XX Interface  
To a Multiplexed Bus**

Figure 2 shows a typical connection to a microcontroller with a multiplexed bus. The ADIO port of the PSD4XX/5XX is connected directly to the microcontroller address/data bus. For an 8-bit bus, the low byte of the ADIO port is connected to AD0 – AD7 and the high byte to A8 – A15 of the microcontroller. For 16-bit bus, the ADIO port connects to AD0-AD15. The address lines are latched internally by the ALE signal. In a read bus cycle, data is driven out through the ADIO Port transceivers after the specified access time. The ADIO Port is in tristate mode if none of the internal PSD resources are selected.

---

**PSD4XX/5XX Interface  
To a Non-Multiplexed Bus**

Figure 3 shows a PSD4XX/5XX interfacing to a microcontroller with a non-multiplexed address/data bus. The address bus is connected to the ADIO Port, and the data bus is connected to Port C and/or Port D, depending on the bus width. If the microcontroller has an address strobe signal, the user has an option to latch or not to latch the address by the ALE/AS signal internally in the PSD.

---

**Optional Features**

The PSD4XX/5XX provides two optional features to add flexibility to the Bus Interface:

**1. Address In**

Port A can be configured as high order address (A16-A23) inputs to the ZPLD for DPLD or other decoding. Any other signals which also are included in the DPLD chip select equations must come from Port A.

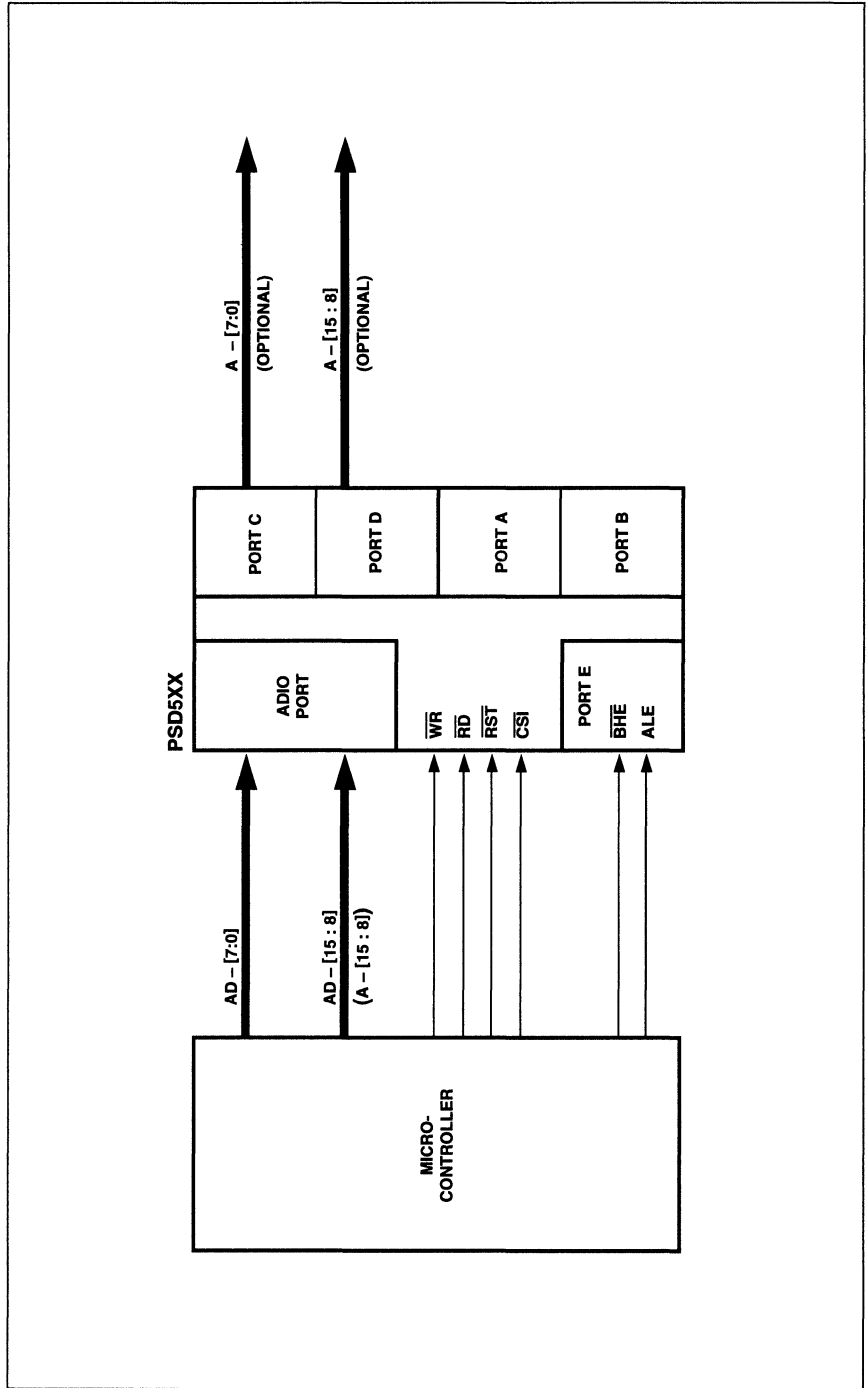
Port C & D can be configured as address input ports for the ZPLD. These inputs should not be used for EPROM decoding.

**2. Address Out**

For multiplexed bus only. Latched address lines A0-A15 are available on Port A, B, C or D. The latched address can be used as address to external memory or I/O devices.

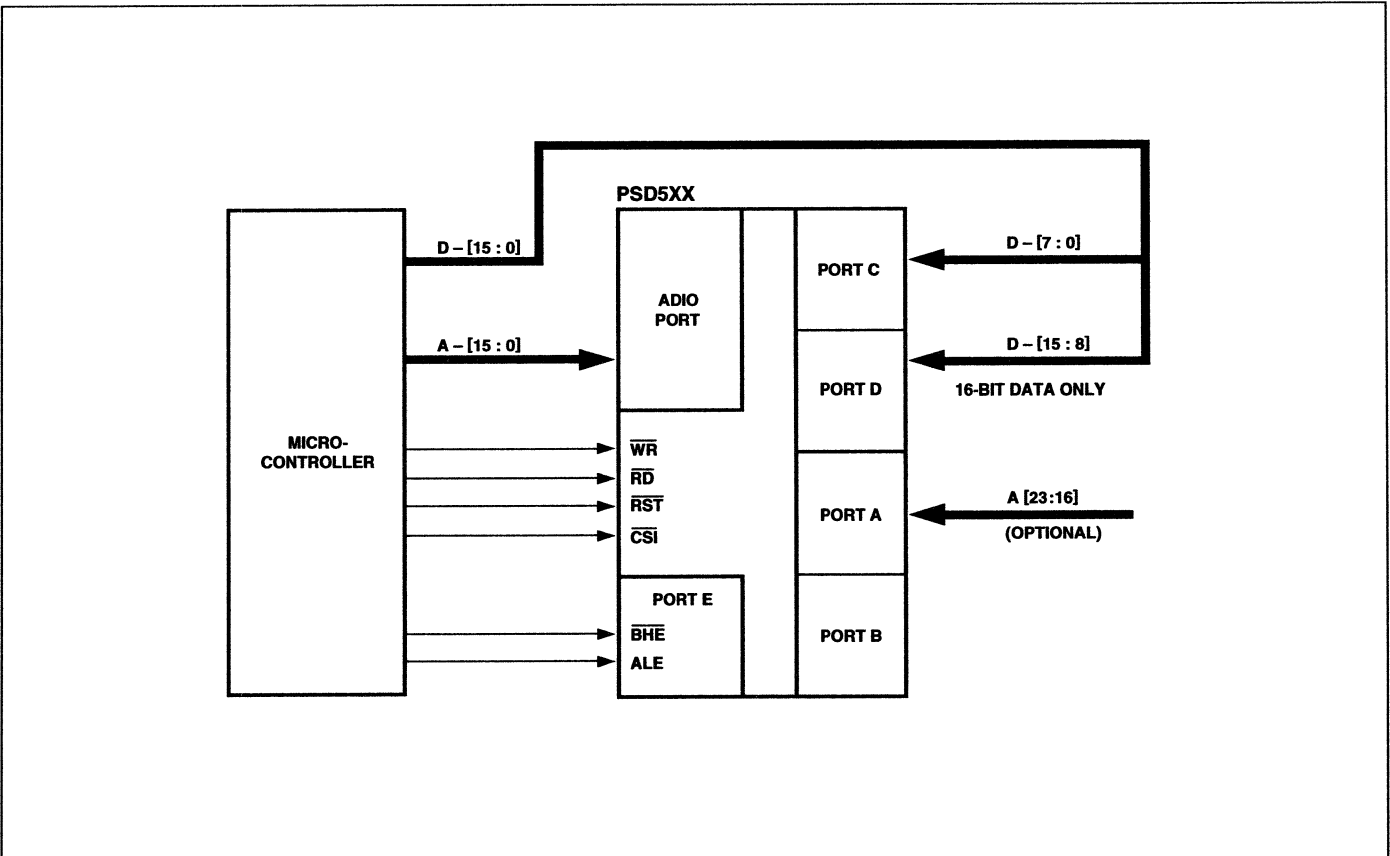
**Bus Interface Of The PSD4XX/5XX (Cont.)**

**Figure 2. Bus Interface – Multiplexed Bus**



**Bus Interface Of The PSD4XX/5XX (Cont.)**

**Figure 3. Bus Interface – Non-Multiplexed Bus**



**Bus Timing Consideration**

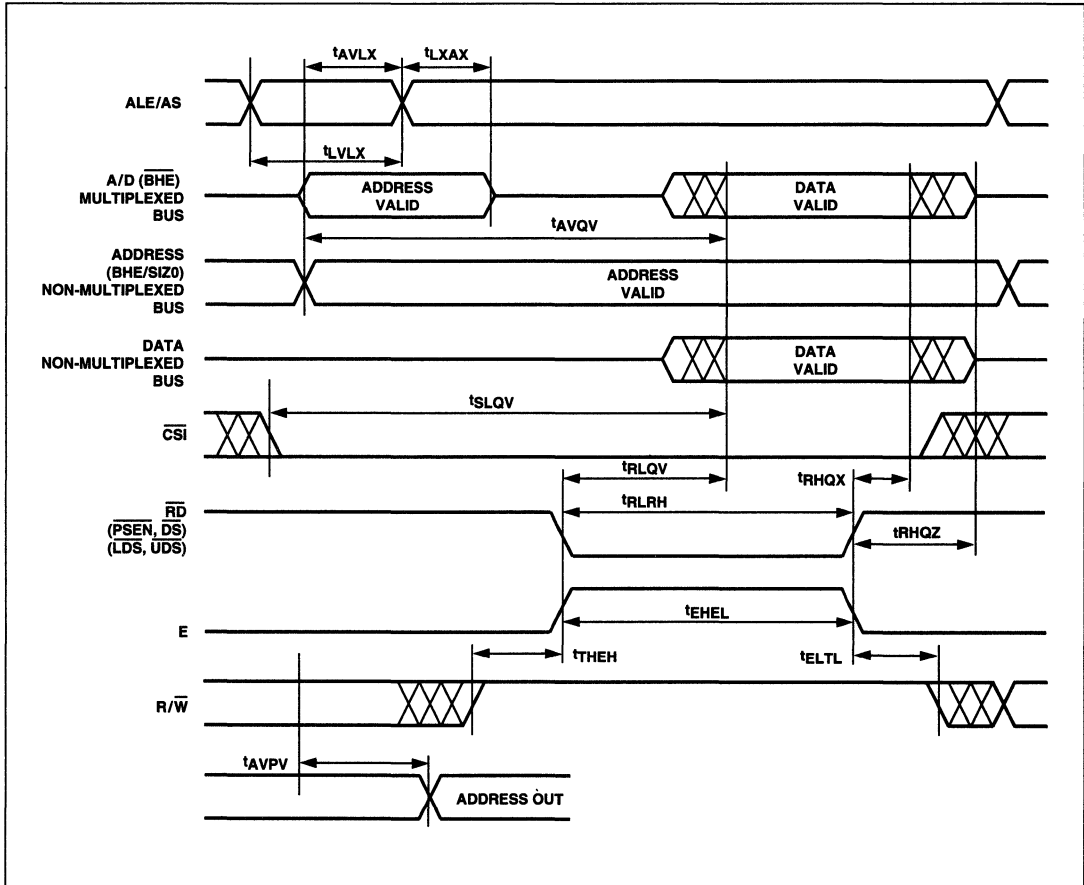
**Access Time Calculation**

Access time of PSD4XX/5XX (EPROM or SRAM ) is the time measured when the address is valid on the Microcontroller address bus to the time the data is available on the data bus. This access time ( $t_{AVQA}$ , see Figure 4) includes any delay on internal address latches and DPLD decoding.

**EPROM CMiser Option**

The PSD4XX/5XX devices have a power management unit which enables the user to configure the power consumption level. The EPROM power is controlled by the EPROM\_CMiser bit (bit 3) in the PMMR0 Register. If this bit is set to "1", the EPROM power consumption is lower but the access time is increased by 10 nanoseconds.

**Figure 4. Read Timing**



3

**Bus Timing  
Consideration  
(Cont.)**

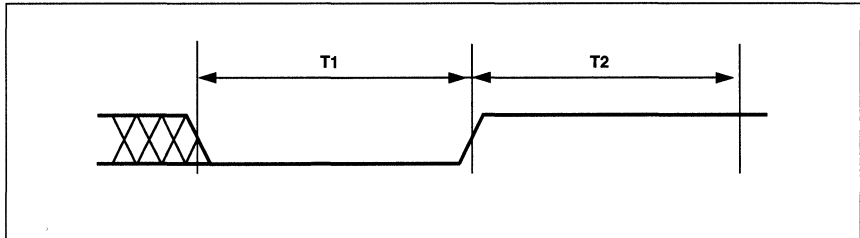
**Reset timing**

Figure 5 shows the reset signal timing requirement of the PSD4XX/5XX devices. The active low ( $T1$ ) has a minimum of 300 ns. After the rising edge of RESET, the PSD4XX/5XX remains inactive during  $T2$  (minimum of 300 ns).

**RST\_OUT Signal (Optional)**

The reset circuit of the PSD4XX/5XX has a Schmitt trigger that senses the RESET line logic level. The PSD4XX/5XX is able to output a RESET signal (referred to as RST\_OUT in this application note) through the GPLD to the microcontroller based on its own reset input. The RST\_OUT signal is not recommended in 68HC11 based design.

**Figure 5. Reset Timing Requirement**



**Microcontrollers  
Supported**

The PSD4XX/5XX is able to support, but is not limited to, the following list of microcontrollers:

- 16-Bit Multiplexed Mode**  
Intel 8096, 80C196, 80C186 families.  
National HPC16000 family.  
Siemens 80C166 families.
- 8-Bit Multiplexed Mode**  
Intel/Philips 80C51/52, 80C31/32, 80C451 families.  
Motorola/Toshiba 68HC11 families.  
Intel 80C188 and 80C198 families.  
SGS-Thomson ST9 families.
- 8-Bit Non-multiplexed Mode**  
Zilog Z80 family  
Motorola 68008/6809 family.  
Echelon 3150™.
- 16-Bit Non-multiplexed Mode**  
Motorola 68302, 68331, 68302, 68HC16 families.

## How To Configure The PSD Bus Interface

The design, configuration and programming of the PSD4XX/5XX is created in the PSDsoft Development Software Tools. The PSDsoft consists of five submodules:

- PSDabel**  
To generate a PLD-ABEL description file which defines the functions of the DPLD (decoder), GPLD and the PPLD (PSD5XX).
- PSDconfiguration**  
To configure the bus interface and other I/O functions.
- PSDcompiler**  
To fit the functions defined in the ABEL file to the PSD and map program codes to the PSD EPROM
- PSDsimulator**  
Chip level simulation based on the ABEL, configuration and stimulus files.
- PSDprogrammer**  
To program the chip with the .obj file generated in the PSDcompiler.

There are two places in the PSDsoft where you specify and define the bus interface for your application:

- 1. In PSDabel**  
Define the DPLD equations (chip select equations) for EPROM, SRAM and I/O.
- 2. In PSDconfiguration:**  
Select the bus type/interface for the PSD such as data bus width, control signals, etc.

3

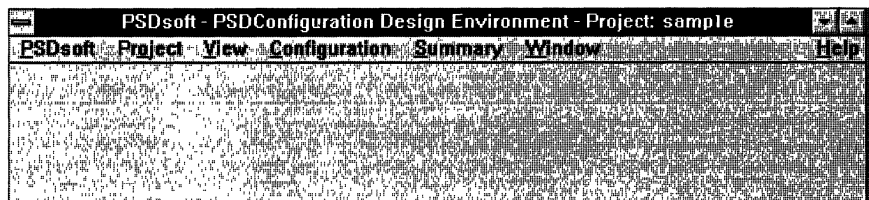
## Select The Bus Interface In PSDconfiguration

The main screen of the PSDconfiguration is shown in Figure 6. Click on the Configuration menu to get to the next screen in Figure 7 where you specify the data bus width and whether the bus is a multiplexed or non-multiplexed bus.

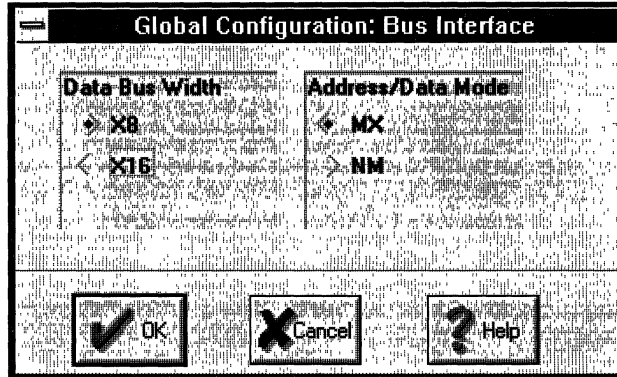
In the next window, as shown in Figure 8, specify the bus control signals of your microcontroller. The polarity of ALE is defined as high if the falling edge of the signal is used to latch the address. In Figure 8, the signals specified are for the 80C31 family of microcontrollers. Please note the question "Use the read signal to access the EPROM" is applicable only to 80C31 type controllers.

This completes the specification of the bus interface. The PSDcompiler will generate the necessary fuse map for the specified bus interface in the .obj file which is to be programmed into the PSD4XX/5XX.

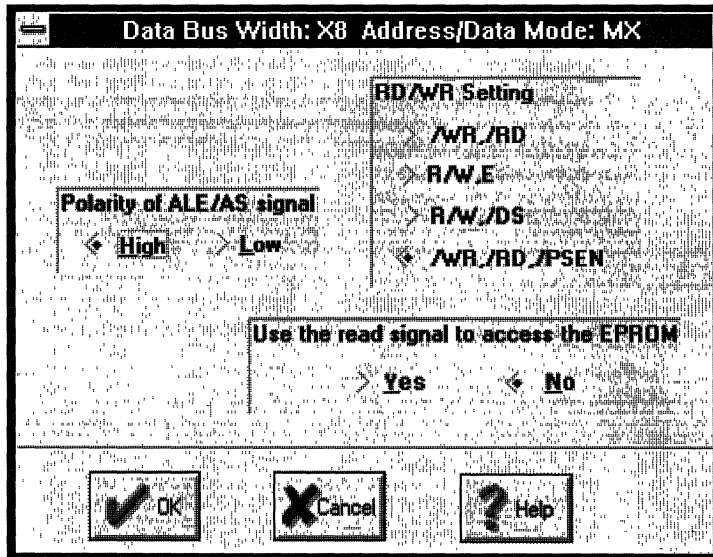
**Figure 6.**



**Select The Bus Interface In PSDconfiguration (Cont.)** **Figure 7.**



**Figure 8.**



## **Definition Of The DPLD Equations In The ABEL File**

The DPLD is the address decoder for the PSD. It generates the chip select signal for all the internal PSD devices. These signals include:

- ES0 – ES3**  
EPROM chip selects (4 blocks)
- RS0**  
SRAM chip select
- CSIOP**  
Port select, Counter/Timer, Interrupt Controller
- PSEL0-1**  
Port A Peripheral Mode selects

The chip select equations normally consist of address inputs and Page Register outputs. You define only the chip selects which you need in the ABEL file. For example, you don't have to define ES3 if the fourth EPROM block is not used. The following is an ABEL example file in which the address lines a0 to a18 of the microcontroller, and pgr0-3 of the Page Register outputs are used as inputs to the DPLD. The memory map of the example is shown in Table 3.

**Table 3. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 0000 – 7FFF         | Page 0             |
| EPROM, Block 1 | 0000 – 7FFF         | Page 1             |
| EPROM, Block 2 | 4800 – 4FFF         | Any page           |
| SRAM           | 8000 - 87FF         | Any page           |
| I/O Devices    | C000 - CFFF         | Any page           |



**Definition  
Of The DPLD  
Equations In The  
ABEL File  
(Cont.)**

The example ABEL file shows only the DPLD portion of the equations (GPLD equations are not included here). A typical ABEL file consists of a module name and/or title; a declarations area where the input/output signals are defined ; and an equations area where the logic equations, and the state machines are defined. An optional test\_vectors area is used for the logic simulation.

```

module dpld
title 'DPLD chip select equations source file';

declarations

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

a18,a17,a16 pin ; "high order address
pgr3,pgr2,pgr1,pgr0 pin; "input for fitting
 "page register embedded inputs

"Output signals

csiop,rs0,es0,es1,es2 node; "DPLD output chip selects

"DEFINITIONS

page = [pgr3,pgr2,pgr1,pgr0];
X = .x ; " Don't care
Address =
[a18,a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS
csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF) ; "Chip Select 256 byte block
rs0 = (Address <= ^h087FF) & (Address >= ^h08000); "SRAM 2k block
es0 = (Address <= ^h07FFF) & (page == 0); "EPROM 32k block only at page 0
es1 = (Address <= ^h07FFF) & (page == 1); "EPROM 32k block only at page 1
es2 = (Address <= ^h4FFFF) & (Address >= ^h48000); "EPROM 32k block, always visible

END dpld

```

## Bus Interface Examples

This section demonstrates the interface between the PSD4XX/5XX and some microcontrollers. The following documents are included in each of the microcontroller interface examples:

- The Bus Configuration (PSDconfiguration) screens captured from the PSDsoft design tool.
- The ABEL file which shows only the declaration and DPLD equations of the targeted microcontroller.
- The logic interface schematic showing the connection between the PSD4XX/5XX and the microcontroller.
- The bus interface simulation screen captured from the SILOSIII Simulator. The Simulator provides a full function, chip level simulation of the PSD4XX/5XX for design verification. The stimulus input file to the Simulator is written in Verilog. The results of the simulation is shown in the SDA (Silos Data Analyzer) window where user defined signals or PSD internal nodes can be traced/displayed.

In the following examples, only the bus interface function of the PSD4XX/5XX is simulated. This includes read bus cycles to the PSD EPROM and SRAM, and write cycles to the SRAM. The EPROM blocks have pre-filled data per Table 4 as the default configuration. The data should give you an indication if the PSD is enabling the right block and byte of the EPROM.

Although the SDA can display many PSD signals, only bus related signals are shown in the examples in this Application Note. Please note the signal names displayed in SDA do not indicate the signal's polarity. As a rule, internal PSD signals all have active high polarity. The bus control signals have the same polarity as defined by the individual microcontroller. The displayed signals include:

- Control Signals**  
Such as  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{DS}$ , ALE,  $\overline{PSEN}$ , etc.
- Address/Data Bus**  
ADIOH and ADIOL (high and low byte of microcontroller address/data bus)
- Data Bus**  
DATAH and DATAL (high and low byte of data bus, for non-mux bus only)
- Chip Selects**  
Chip select signals to EPROM (es0 – es3) and SRAM (rs0).

**Table 4.**

| <b>EPROM Block</b> | <b>Odd Byte</b> | <b>Even Byte</b> |
|--------------------|-----------------|------------------|
| Block0 (ES0)       | 01h             | 23h              |
| Block1 (ES1)       | 45h             | 67h              |
| Block2 (ES2)       | 89h             | abh              |
| Block3 (ES3)       | cdh             | efh              |

**Interfacing  
To The 80C31  
Family Of  
Microcontrollers**

**The 80C31 Bus**

80C31 is an 8-bit microcontroller with multiplexed address/data bus. It has the following bus signals:

- Address/Data Bus:** AD7 – AD0
- Address Bus:** A15 – A8
- Address Strobe:** ALE
- Control Signals:**  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{PSEN}$

The PSEN signal is used to fetch code and  $\overline{RD}$  is used to read data. This allows the 80C31 to address up to 64KB of data memory and 64KB of program memory.

**Two Modes of Memory Access**

The PSD4XX/5XX provides two modes of memory access: the Separated Space Mode and the Combined Space Mode (see Tables 5 and 5a). In Separated Mode, the  $\overline{PSEN}$  signal can access the EPROM only and the  $\overline{RD}$  signal can access the SRAM only. In Combined Space Mode, the EPROM can be accessed both by the  $\overline{PSEN}$  and  $\overline{RD}$  signal. The Combined Mode is for application where blocks of data or look up tables are required to reside in the EPROM.

The PSD4XX/5XX also provide an option for program code to be stored and executed from the SRAM. This option is enabled if the SRCODE bit in the VM Register is set to “1” during run time.

**Table 5. Separated Space Mode**

|             | <b>EPROM Access</b> | <b>SRAM Access</b>     |
|-------------|---------------------|------------------------|
| RD Signal   | No                  | Yes                    |
| PSEN Signal | Yes                 | Yes only if SRCODE = 1 |

**Table 5a. Combined Space Mode**

|             | <b>EPROM Access</b> | <b>SRAM Access</b>     |
|-------------|---------------------|------------------------|
| RD Signal   | Yes                 | Yes                    |
| PSEN Signal | Yes                 | Yes only if SRCODE = 1 |

---

**Interfacing  
To The 80C31  
Family Of  
Microcontrollers  
(Cont.)****The 80C31 and PSD4XX/5XX Interface Schematic**

Figure 9 shows the 80C31 and PSD4XX/5XX interface schematic. The address/data bus and the bus control signals such as ALE,  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{PSEN}$  etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic. Reset for the 80C31 is generated from the  $\overline{RESET}$  input to the PSD4XX/5XX and outputs on pin PE2 in this example. If clock input is not required, the CLKIN pin should always be grounded.

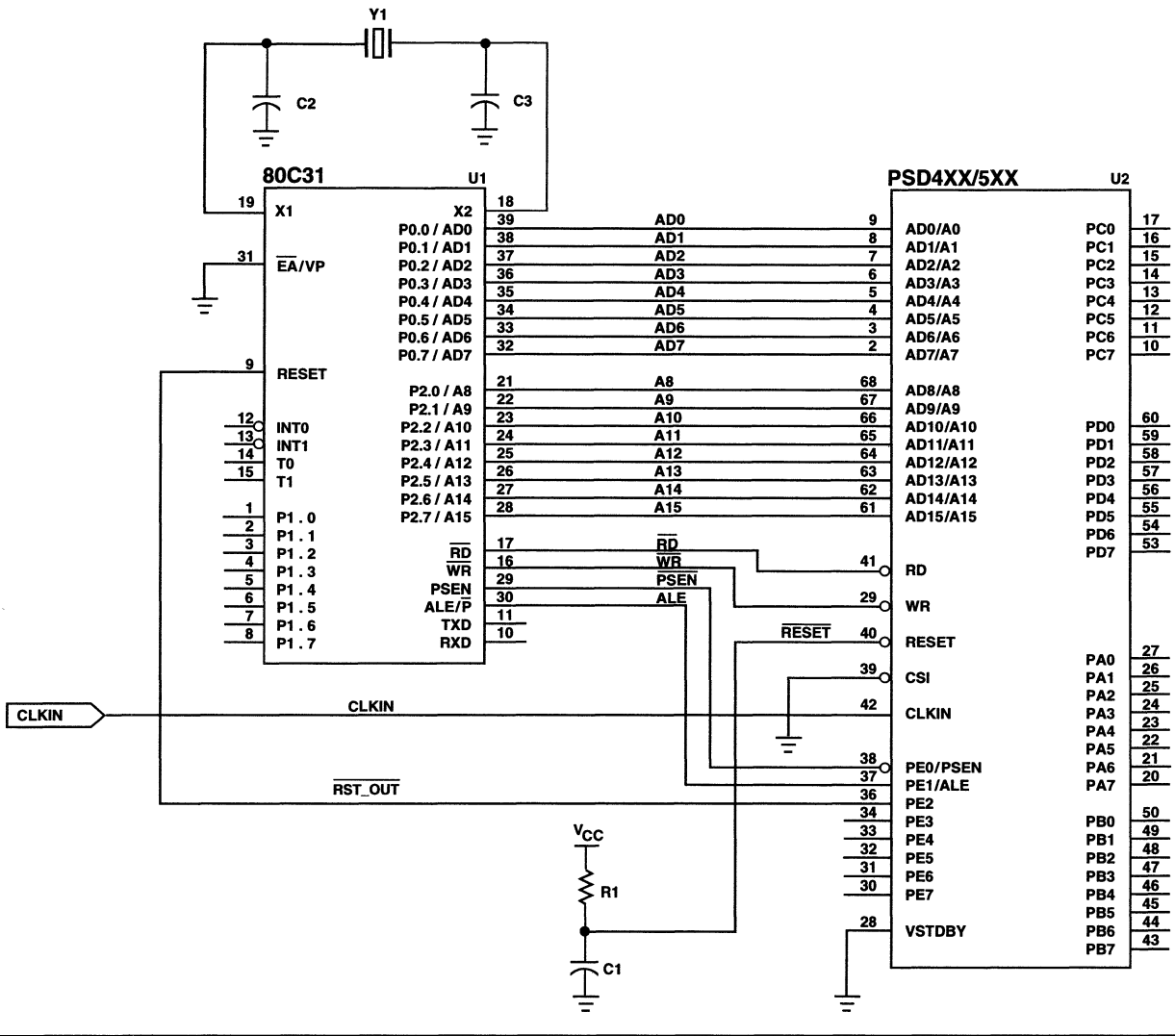
---

**Reset Circuit Recommendations**

The following three reset circuits are recommended for use with 80C31 and PSD4XX/5XX based designs:

1. Input RESET signal into the PSD4XX/5XX RESET pin. Based on the polarity of the RESET INPUT signal of the microcontroller interfaced to the PSD, generate RST\_OUT through the GPLD and connect it to the microcontroller's RESET INPUT pin (as illustrated in this application note.)
2. Use a Reset Chip such as Dallas Semiconductor's DS1232, or Maxim's Max 699. In case of Maxim's Max 699, the Small Outline (SO) package should be used where the RESET output without inversion is also available. The inverted RESET signal goes to the PSD RESET input pin and the non-inverted RESET signal is connected to the RESET input of 80C31.
3. Use two separate RC reset circuits: one which generates a high reset pulse to the 80C31 and the other one generates a low reset pulse to the PSD4XX/5XX. The RC constant of the PSD4XX/5XX reset circuit should be less than that of the 80C31 such that the PSD4XX/5XX reset signal has a shorter pulse and eliminates any race condition.

Figure 9. 80C31 and PSD4XX/5XX Interface



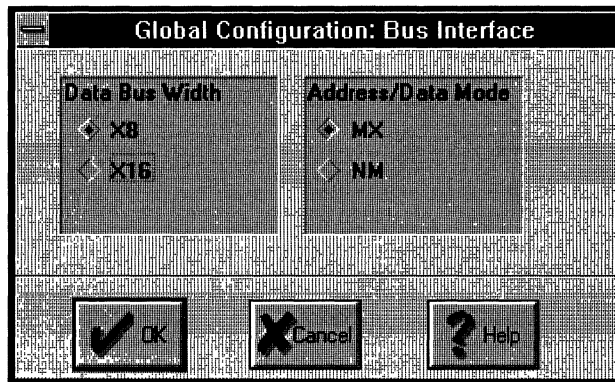
## Interfacing To The 80C31 Family Of Microcontrollers (Cont.)

### Specify The 80C31 Bus Interface In PSDconfiguration

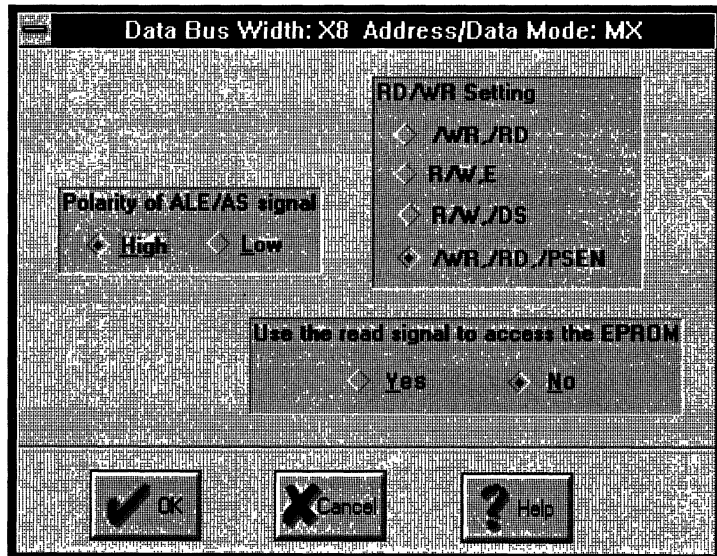
As shown in the following windows which are captured from PSDconfiguration, the 80C31 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: MX
- Polarity of ALE: High
- RD/WR Setting:  $\overline{WR}$ , RD,  $\overline{PSEN}$

The PSDconfiguration also asks the question “Use the read signal to access the EPROM”. A click on “Yes” means you are selecting the Combined Space Mode and that both the  $\overline{PSEN}$  or RD signal can access the EPROM. A “no” will select the Separated Space Mode and EPROM can be accessed by  $\overline{PSEN}$  only.



3



**Interfacing  
To The 80C31  
Family Of  
Microcontrollers  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL file**

The following is an example of defining the decoding function for the 80C31 based application. Please note the reset input to the 80C31, rst\_out, is also included in the file. This file is applicable to both the Separated or Combined Space mode. The memory map is shown in Table 6.

```

module psen
title 'Design example of 80C31 DPLD source file';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

"Output signals

csiop, rs0, es0, es1, es2 node; "DPLD output chip selects
reset pin; "reset is declared here, used in rst_out generation
rst_out pin 36;

"DEFINITIONS

X = .x. ; " Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hC000) & (Address <= ^hC0FF); "CSIOP 256bytes block
rs0 = (Address <= ^h0000) & (Address >= ^h03FF); "SRAM 2KB block

es0 = (Address >= ^h0000) & (Address <= ^h3FFF); "1st EPROM block cs
es1 = (Address >= ^h4000) & (Address <= ^h7FFF); "2nd EPROM block cs
es2 = (Address >= ^h8000) & (Address <= ^hBFFF); "3rd EPROM block cs
es3 = (Address >= ^hC000) & (Address <= ^hFFFF); "4th EPROM block cs

"GPLD EQUATIONS

rst_out = !reset; "generate a high active reset to 80C31

END

```

**Table 6. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 0000 – 3FFF         |                    |
| EPROM, Block 1 | 4000 – 7FFF         |                    |
| EPROM, Block 2 | 8000 – BFFF         |                    |
| SRAM           | 0000 – 03FF         |                    |
| I/O Devices    | C000 – CFFF         |                    |



## Interfacing To The 80C31 Family Of Microcontrollers (Cont.)

### Overlapping EPROM Space In Combined Mode

If your application requires the data and program to be resided in the EPROM (Combined Space Mode) and share the same address space, you need to modify the chip select equations. For example, if EPROM blocks 0-1 are used as code area and blocks 2-3 are used as data area, and that code and data space share the same address. In this case, the RD signal is used to separate the program and data space. The program space is enabled by an active PSEN, while the data space is enabled by an active RD. The RD signal now is considered as an address input and thus the access time of the EPROM starts from when RD is valid, instead of when address is valid. The following is the chip select equations of the EPROM blocks.

es0 = (Address >= ^h0000) & (Address <= ^h3FFF) & RD ; " program area

es1 = (Address >= ^h4000) & (Address <= ^h7FFF) & RD ; " program area

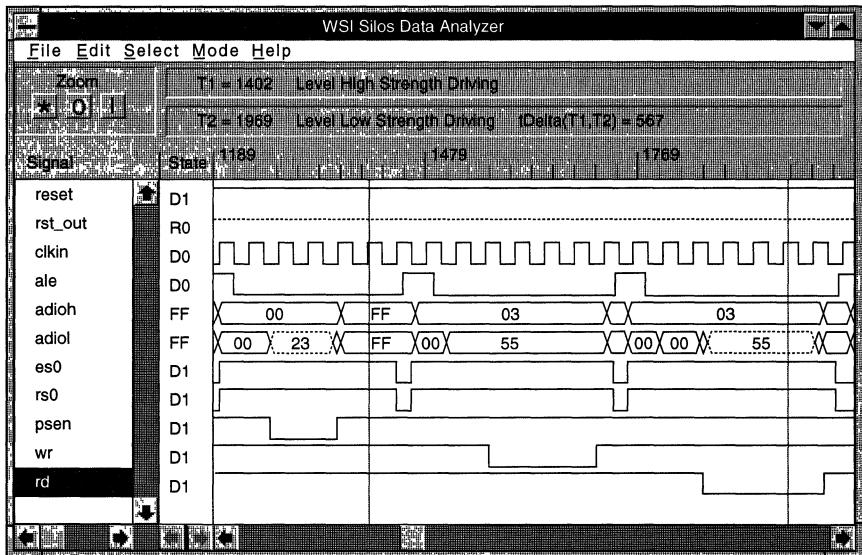
es2 = (Address >= ^h0000) & (Address <= ^h3FFF) & !RD ; "data area

es3 = (Address >= ^h4000) & (Address <= ^h7FFF) & !RD ; "data area

### Simulation of 80C31 Bus Cycles With The PSD4XX/5XX

Figure 10 shows the simulation of three 80C31 bus cycles. The first cycle is a code fetch from EPROM block 0 where code "23" is driven on to the ADIOL bus by the PSD. The next two cycles are SRAM write (data = 55h) and read cycles to location 0300h.

Figure 10.

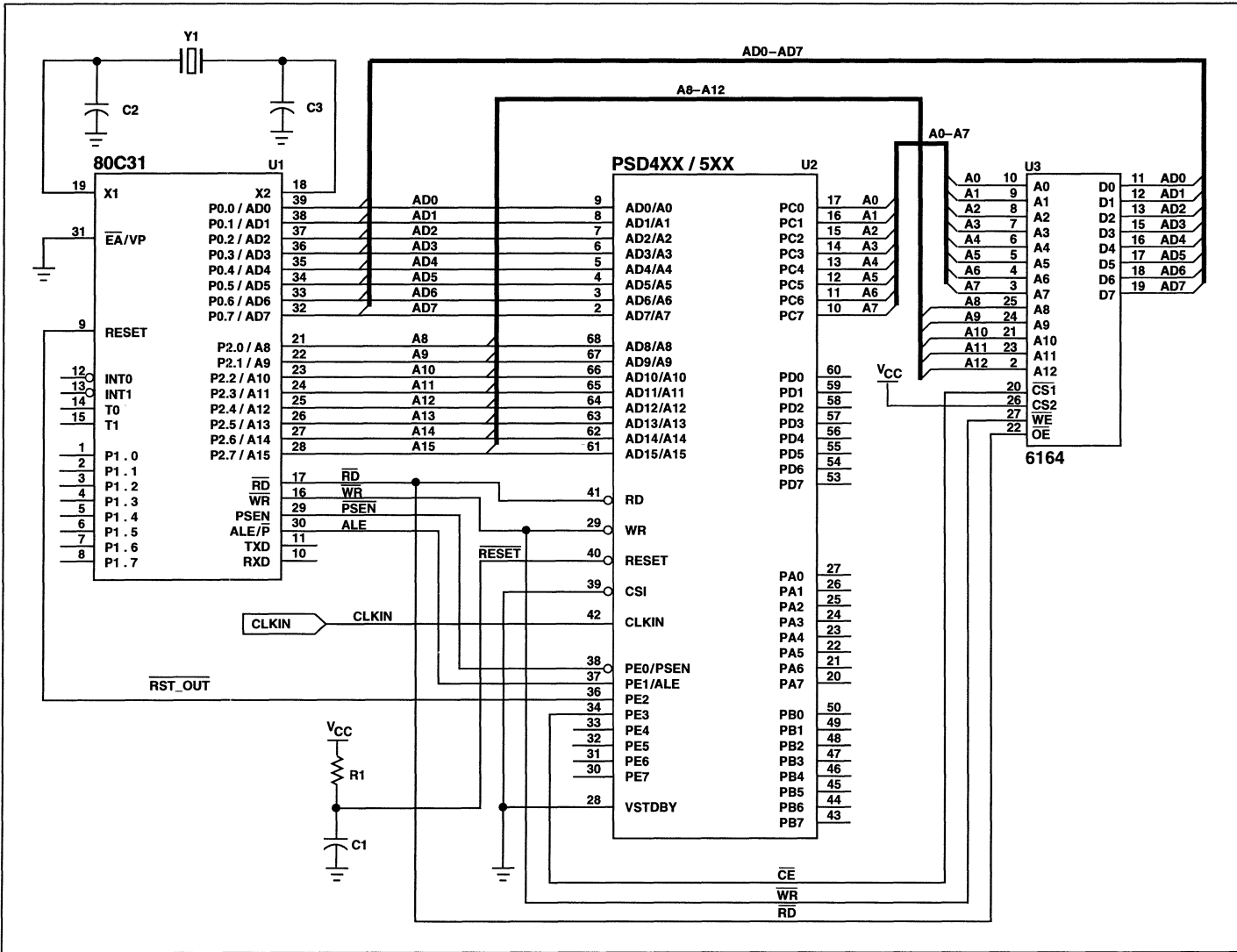


### 80C31 With PSD4XX/5XX and External Memory

In applications where large amount of SRAM is required, the PSD4XX/5XX is able to support an additional external SRAM. Figure 11 illustrates how an external SRAM (6164) can be interfaced to the PSD4XX/5XX and the 80C31 without additional hardware. Port C (or any other port) is configured to provide latched output addresses A0 – A7, and the SRAM chip select is generated from the GPLD.



Figure 11. 80C31, PSD4XX/5XX and External SRAM Interface



## **Interfacing To The 68HC11 Family Of Microcontrollers**

The 68HC11 family of microcontrollers have two types of bus interfaces. The standard HC11 has a multiplexed bus, while the 68HC11K4 has a non multiplexed bus. The example here covers both bus configurations.

### **The 68HC11 Bus**

The standard 68HC11 has a multiplexed bus where the lower address lines multiplex with an 8-bits data bus. It has the following bus signals:

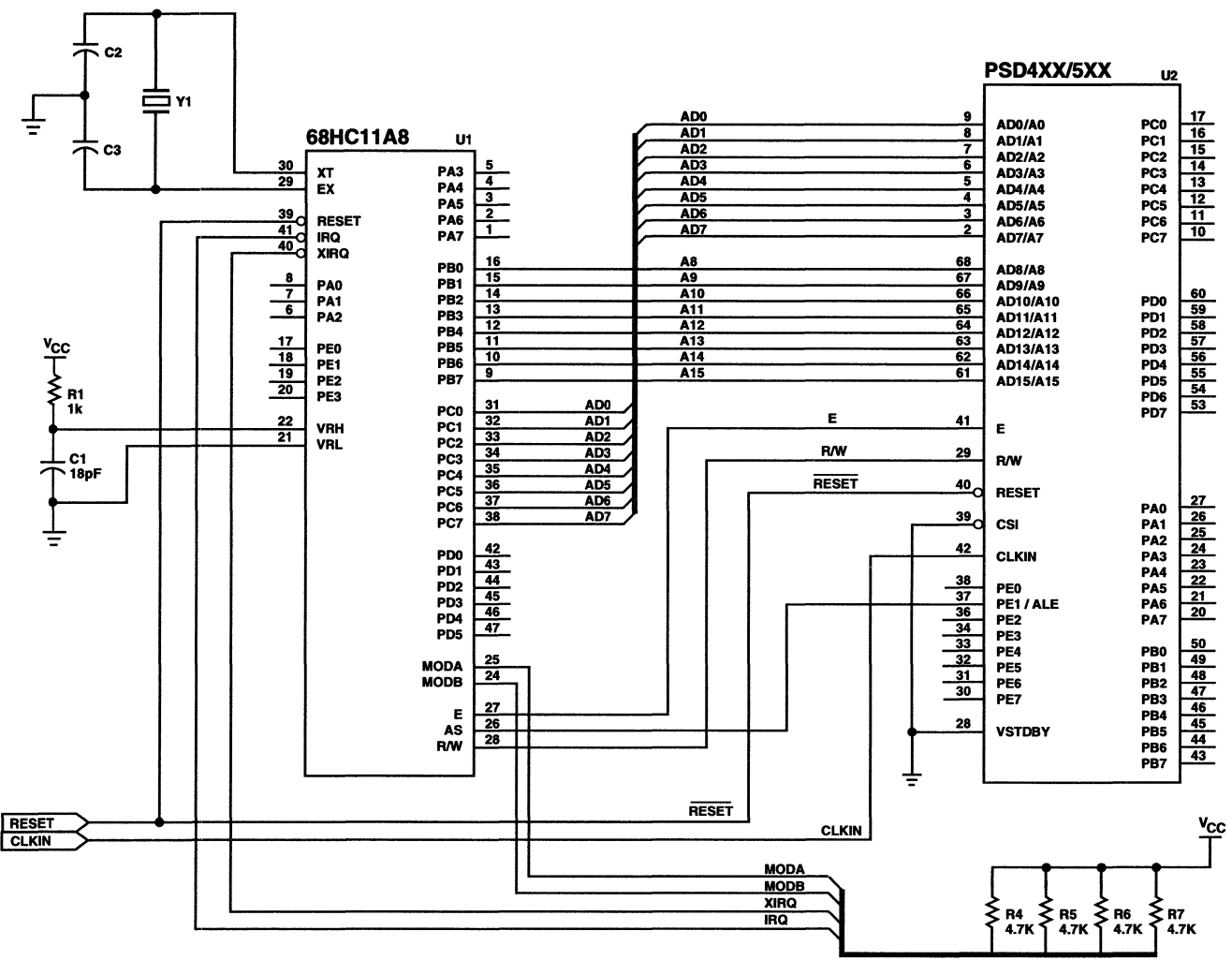
- Address/Data Bus:** AD7 – AD0
- Address Bus:** A15 – A8
- Address Strobe:** AS
- Control Signals:** E, R/W

### **68HC11 Interface to PSD4XX/5XX**

The 68HC11 can interface directly to the PSD4XX/5XX without any additional glue logic. As shown in Figure 12, the E clock is connected to the “RD” pin, which is configured to act as the E clock input. The R/W signal is connected to the “WR” pin, which is configured to act as the R/W input.

The PSD4XX/5XX generates internal “write” and “read” signals based on the E clock and the R/W inputs. If E clock is high and R/W is high, then PSD4XX/5XX sees it as a read bus cycle and drives data on to the data bus through the ADIO Port if any of its internal devices are selected.

Figure 12. 68HC11 and PSD4XX/5XX Interface

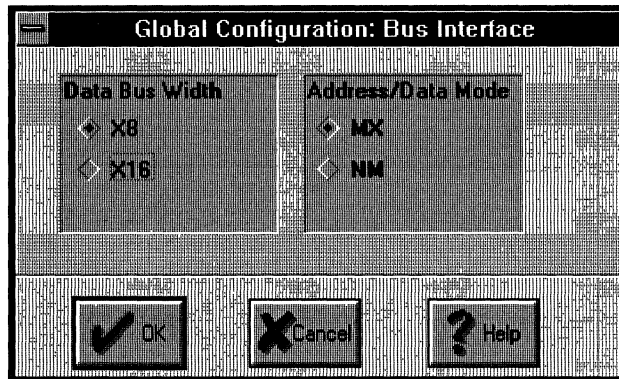


## Interfacing To The 68HC11 Family Of Microcontrollers (Cont.)

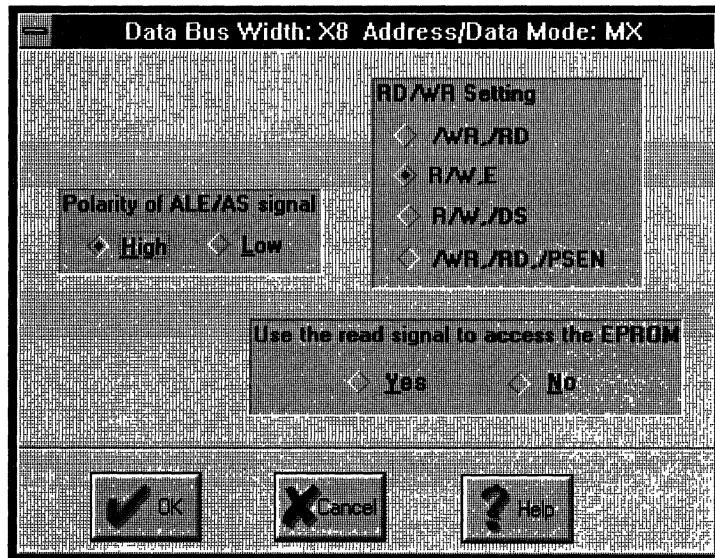
### Specify the 68HC11 Multiplexed Bus Interface in PSDconfiguration

As shown in the following windows which are captured from PSDconfiguration, the 68HC11 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: MX
- Polarity of ALE: High
- RD/WR Setting: R/W, E



3



**Interfacing  
To The 68HC11  
Family Of  
Microcontrollers  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL File**

The following is an example of defining the decoding function for the 68HC11 based application. Table 7 shows the memory map implemented by the DPLD.

**Table 7. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 1 | 4000 – 7FFF         |                    |
| EPROM, Block 2 | 8000 – BFFF         |                    |
| EPROM, Block 3 | C000 – FFFF         |                    |
| SRAM           | 1000 – 13FF         |                    |
| I/O Devices    | 0000 – 00FF         |                    |

```

module hc11
title 'DPLD chip select equations source file';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

"Output signals
as,rd_wr,e pin 37,29,41; "Motorola related ale and read/write signals
csiop, rs0, es1, es2, es3 node; "DPLD output chip selects

"DEFINITIONS

X = .x.; " Don't care
Address =
[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^h0000) & (Address <= ^h00FF); "CSIOP 256bytes block
rs0 = (Address <= ^h1000) & (Address >= ^h13FF); "SRAM 2KB block

es1 = (Address >= ^h4000) & (Address <= ^h7FFF); "2nd EPROM block cs
es2 = (Address >= ^h8000) & (Address <= ^hBFFF); "3rd EPROM block cs
es3 = (Address >= ^hC000) & (Address <= ^hFFFF); "4th EPROM block cs

"The first EPROM block is not used and it is not required to define es0

END

```

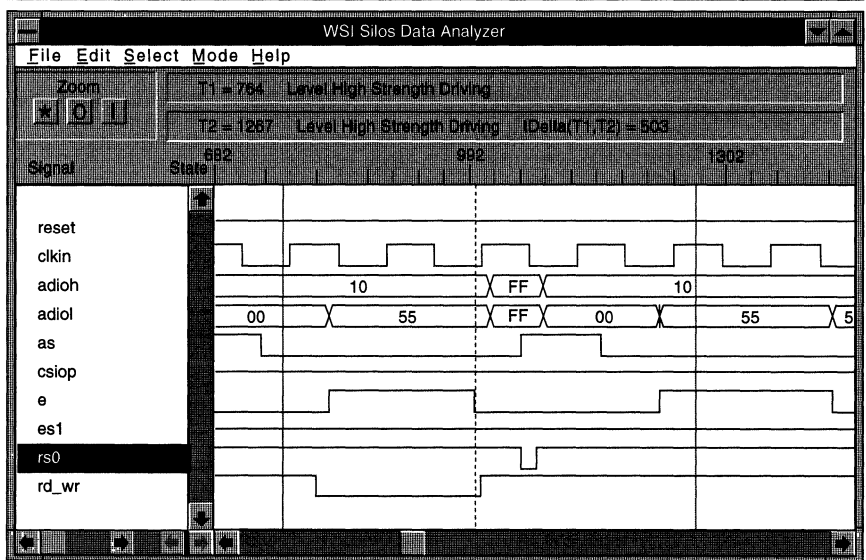


## Interfacing To The 68HC11 Family Of Microcontrollers (Cont.)

### Simulation of 68HC11 Bus Cycles With the PSD4XX/5XX

Figure 13 shows the output of the 68HC11 bus cycle simulation. Data byte 55h is written to location 1000h of the SRAM in a write bus cycle with the R/W signal low. In the next cycle, the R/W signal is high and the same data byte is being read back as shown in the ADIOL bus.

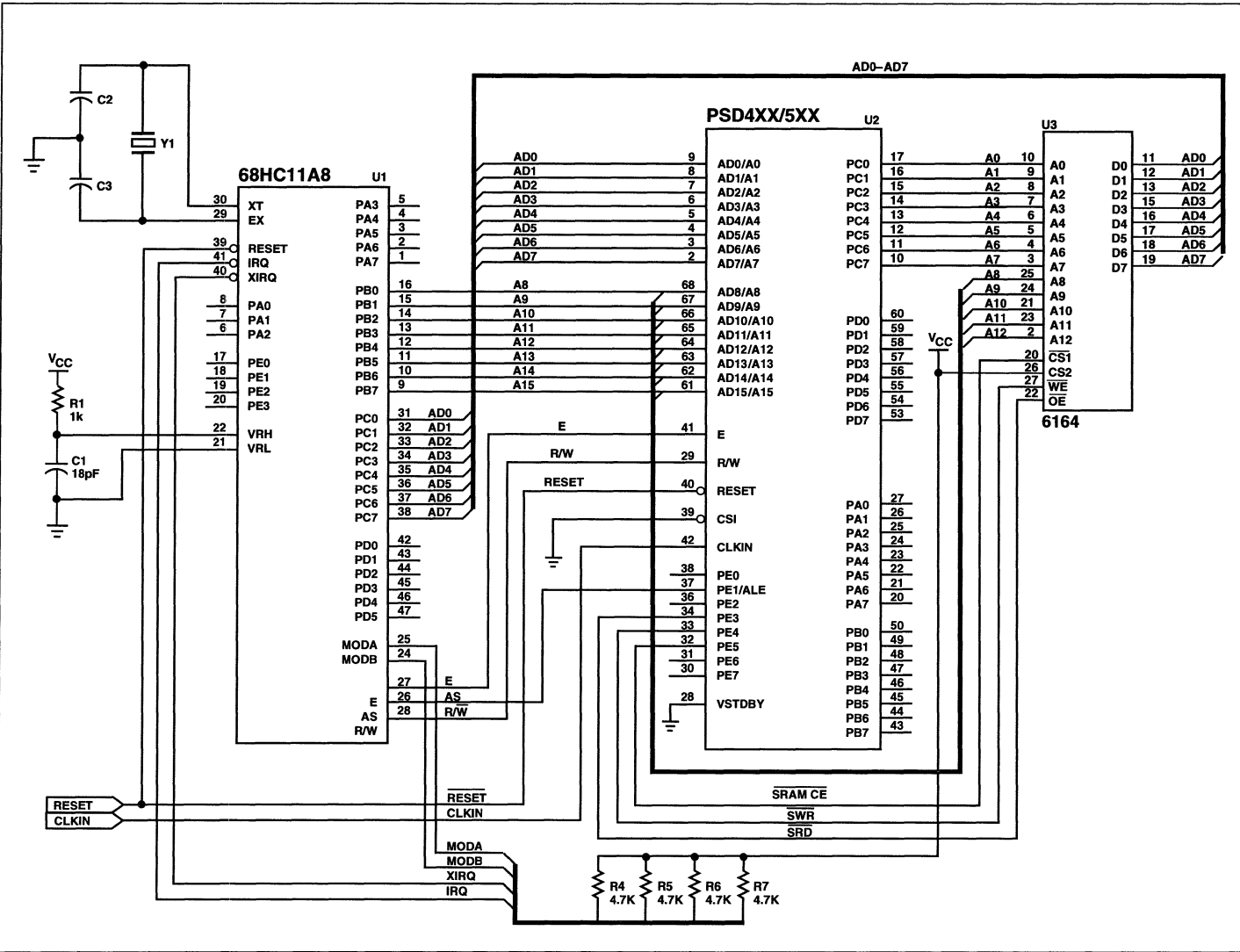
**Figure 13.**



### 68HC11 With PSD4XX/5XX and External Memory

In applications where a large amount of SRAM is required, the PSD4XX/5XX is able to support an additional external SRAM. Figure 14 illustrates how an external SRAM (6164) can be interfaced to the PSD4XX/5XX and the 68HC11 with multiplexed address/data bus without additional hardware. Port C is configured to provide the latched output addresses A0 – A7 (A8 – A15 come directly from the 68HC11). The SRAM chip select and the read/write signals are generated from the GPLD.

Figure 14. 68HC11 and PSD4XX/5XX and External SRAM Interface



**Interfacing  
To The 68HC11  
Family Of  
Microcontrollers  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL File For External SRAM**

The following is an example of defining the decoding function for the 68HC11 based application with external SRAM. The latched address A0 – A7 are assigned to Port C. The “/wr” and “/rd” signals, which can be used for other devices besides the SRAM, are also generated. Table 8 shows the memory map.

**Table 8. System Memory Map**

| <b>Device</b>   | <b>Memory Space</b> | <b>Memory Page</b> |
|-----------------|---------------------|--------------------|
| EPROM, Block 1  | 4000 – 7FFF         |                    |
| EPROM, Block 2  | 8000 – BFFF         |                    |
| EPROM, Block 3  | C000 – FFFF         |                    |
| SRAM (PSD)      | 1000 – 13FF         |                    |
| SRAM (External) | 2000 – 3FFF         |                    |
| I/O Devices     | 0000 – 00FF         |                    |



## Interfacing To The 68HC11 Family Of Microcontrollers (Cont.)

### Define The DPLD/Decoding Function In The ABEL File For External SRAM (Cont.)

```

module hc11
title 'Design example of 68hc11 DPLD source file to interface with external SRAM';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

"Output signals
as,rd_wr,e pin 37,29,41; "Motorola related ale and read/write signals
csiop, rs0, es0, es1, es2 node ; "DPLD output chip selects

"assign pins (port c) for latched address out
addr0, addr1,addr2,addr3,addr4,addr6, addr7 pin 17,16,15,14,13,12,11,10;

"External SRAM chip select and read/write signal generation
swr, srd, sram_ce pin;

"DEFINITIONS

X = .x ; " Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^h0000) & (Address <= ^h00FF); "CSIOP 256bytes block
rs0 = (Address <= ^h1000) & (Address >= ^h13FF); "SRAM 2KB block

es1 = (Address >= ^h4000) & (Address <= ^h7FFF); "2nd EPROM block cs
es2 = (Address >= ^h8000) & (Address <= ^hBFFF); "3rd EPROM block cs
es3 = (Address >= ^hC000) & (Address <= ^hFFFF); "4th EPROM block cs

"The first EPROM block is not used and it is not required to define es0

"Equations to select/read/write the 61128, external SRAM through PSD

swr = !(e & lrd_wr); "write signal
srd = !(e & rd_wr); "read signal
sram_ce = (Address >= ^h2000) & (Address <= ^h3FFF); "8K SRAM chip select

END

```

---

**Interfacing  
To The 68HC11  
Family Of  
Microcontrollers  
(Cont.)**

**The 68HC11K4 Bus**

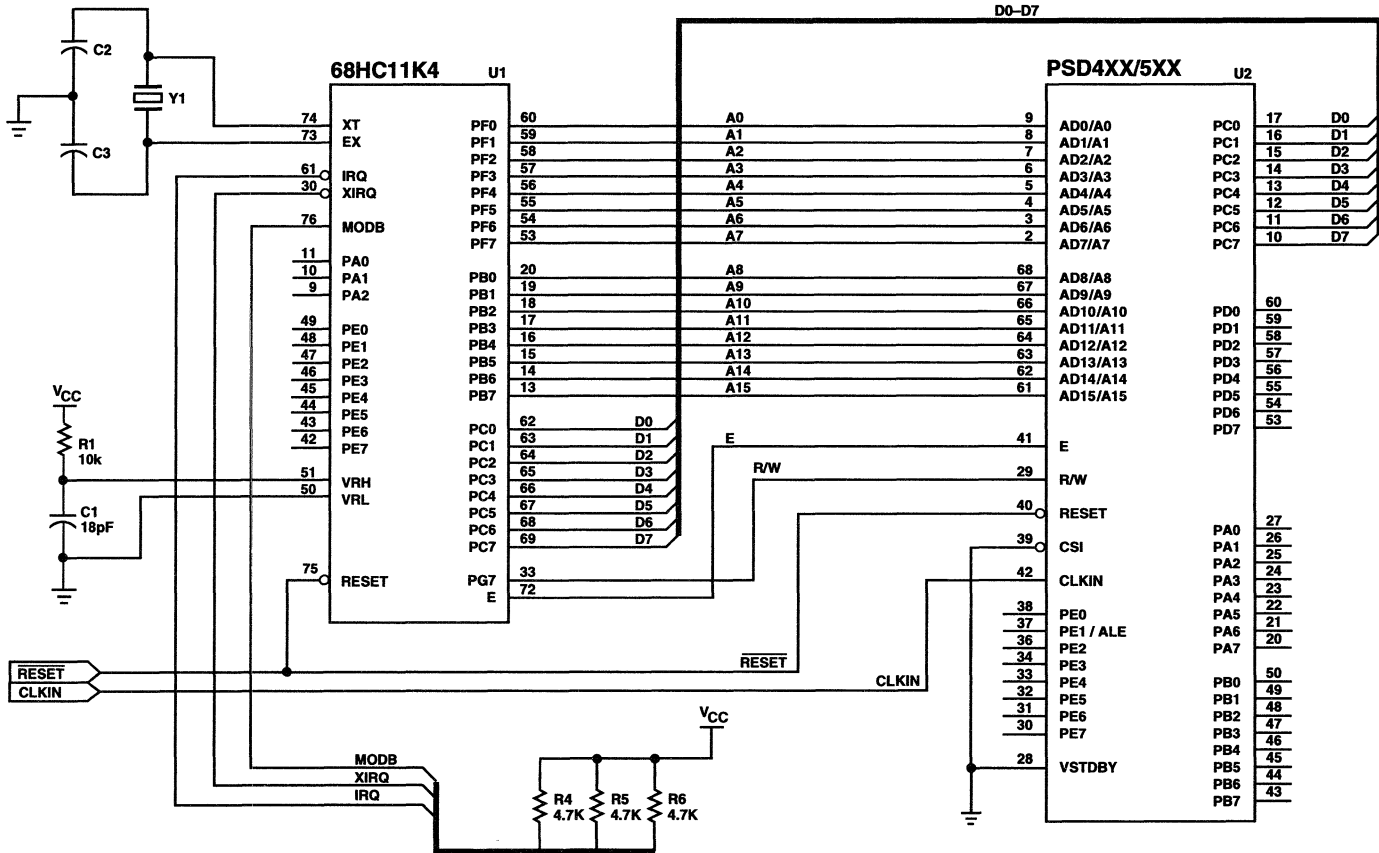
Motorola's 68HC11K4 has a non-multiplexed 16-bit address and an 8-bit data bus. The control signals used for accessing I/O devices or memory are the E clock and the R/W signal.

**The 68HC11K4 Interface to PSD4XX/5XX:**

The 68HC11K4 can interface directly to the PSD4XX/5XX without any additional glue logic. As shown in Figure 15, the E clock is connected to the "RD" pin, which is configured to act as the E clock input. The R/W signal is connected to the "WR" pin, which is configured to act as the R/W input.

The PSD4XX/5XX generates internal "write" and "read" signals based on the E clock and the R/W inputs. If E clock is high and R/W is high, then PSD4XX/5XX sees it as a read bus cycle and drive data onto the data bus through the Port C if any of its internal devices is selected.

Figure 15. 68HC11K4 and PSD4XX/5XX Interface

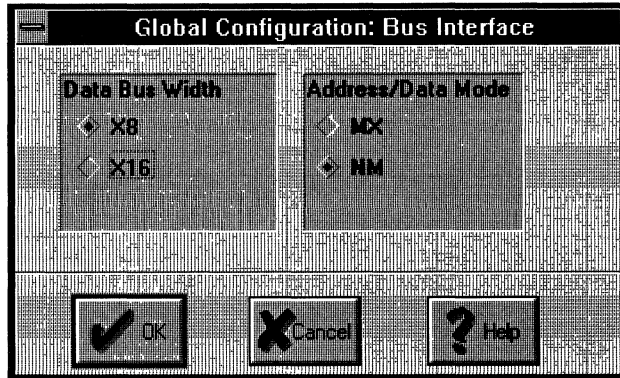


**Interfacing  
To The 68HC11  
Family Of  
Microcontrollers  
(Cont.)**

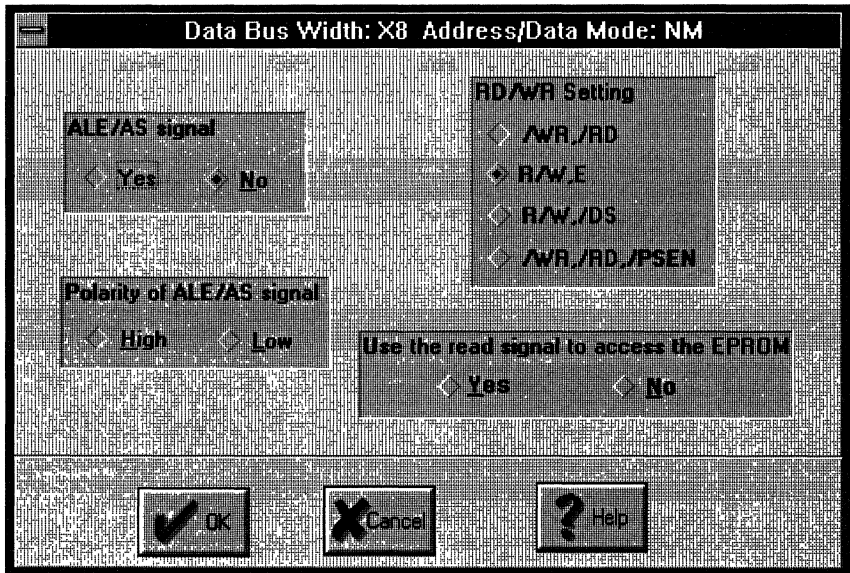
**Specify The 68HC11K4 Non-Multiplexed Bus Interface In PSDconfiguration**

As shown in the following windows which are captured from PSDconfiguration, the 68HC11 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: NM
- RD/WR Setting: R/W, E



3



**Interfacing  
To The 68HC11  
Family Of  
Microcontrollers  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL File**

The following is an example of defining the decoding function for the 68HC11K4 based application. Table 9 shows the memory map implemented by the DPLD.

**Table 9. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 1 | 4000 – 7FFF         |                    |
| EPROM, Block 2 | 8000 – BFFF         |                    |
| EPROM, Block 3 | C000 – FFFF         |                    |
| SRAM           | 1000 – 13FF         |                    |
| I/O Devices    | 0000 – 00FF         |                    |

```

module hc11k4
title 'Design example of 68hC11K4 DPLD source file';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

"Output signals
rd_wr,e pin 29,41; "Motorola related ale and read/write signals
csiop, rs0, es0, es1, es2, es3 node; "DPLD output chip selects

"DEFINITIONS

X = .x.; " Don't care
Address =
[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^h0000) & (Address <= ^h00FF); "CSIOP 256bytes block
rs0 = (Address >= ^h1000) & (Address <= ^h13FF); "SRAM 2k block

es1 = (Address >= ^h4000) & (Address <= ^h7FFF); "2nd EPROM block cs
es2 = (Address >= ^h8000) & (Address <= ^hBFFF); "3rd EPROM block cs
es3 = (Address >= ^hC000) & (Address <= ^hFFFF); "4th EPROM block cs

END

```

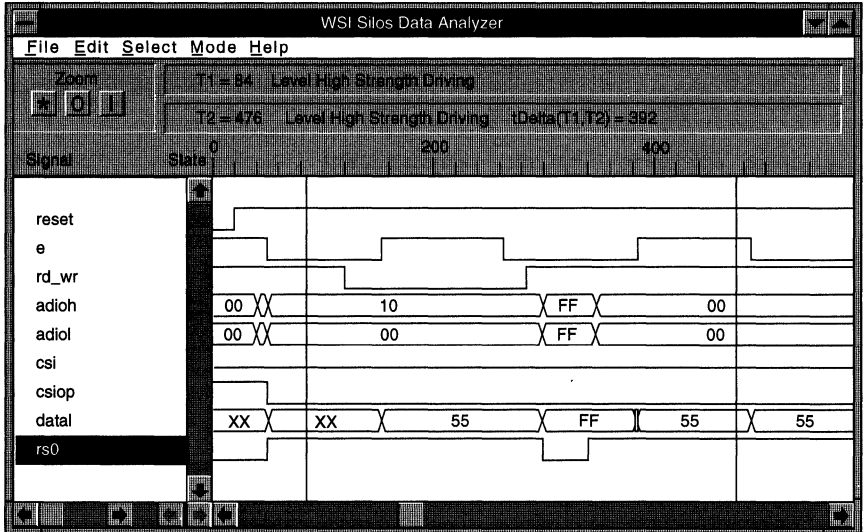


**Interfacing  
To The 68HC11  
Family Of  
Microcontrollers  
(Cont.)**

**Simulation Of 68HC11K4 Bus Cycles With The PSD4XX/5XX**

Figure 16 shows the output of the 68HC11K4 bus cycle simulation. Data byte 55h is written to loaction 1000h of the SRAM in a write bus cycle with the R/W signal low. In the next cycle, the R/W signal is high and the same data byte is being read back as shown in the DATAL bus.

**Figure 16.**



3

**Interfacing  
To The 80C196  
Family Of  
Microcontrollers**

**The 80C196 Bus**

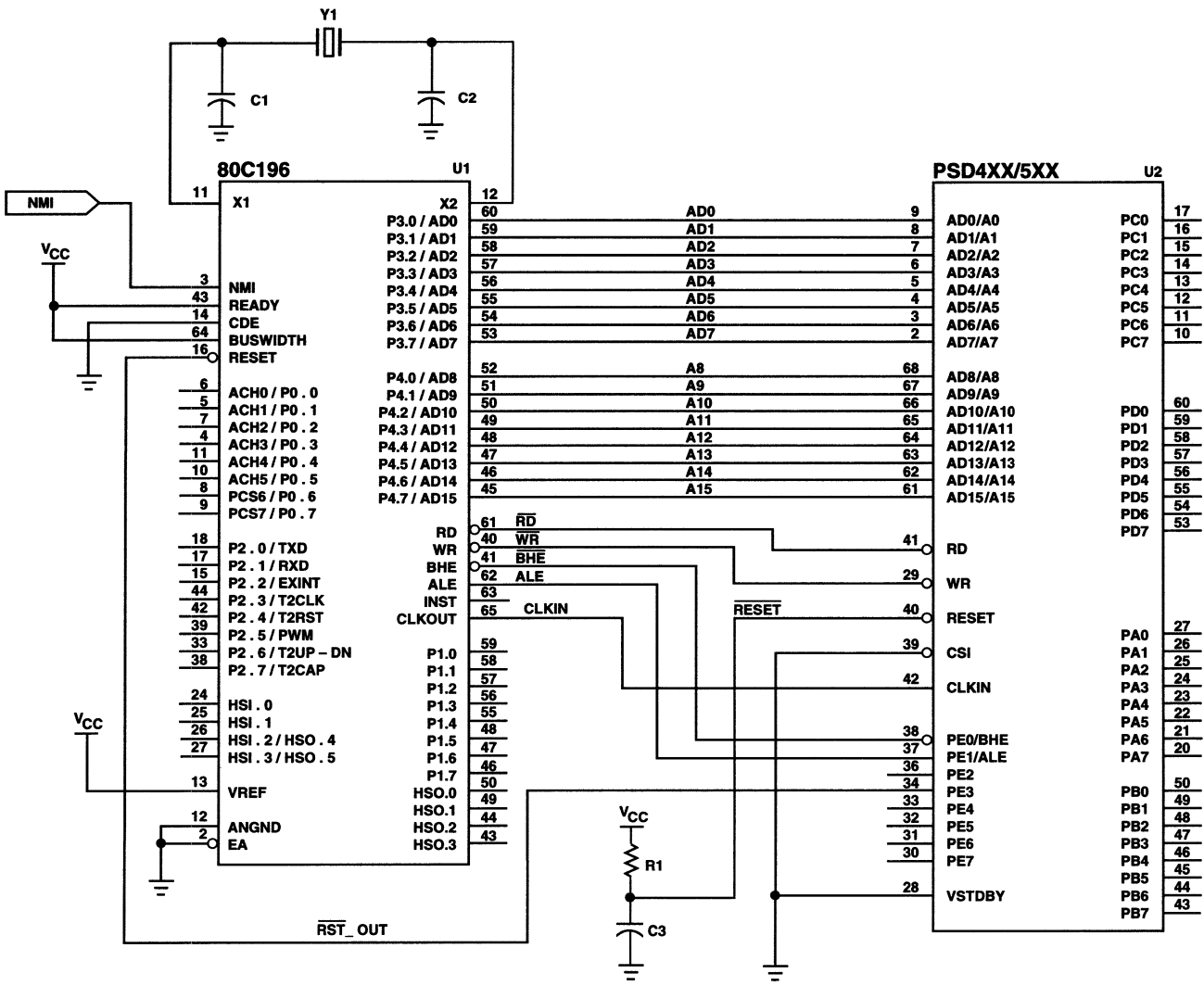
The 80C196 family of microcontrollers has a 16-bit multiplexed address/data bus. The processor has a dynamic data bus width. In a typical application, the EPROM has an 8-bit data bus while the SRAM has a 16-bit data bus. The PSD4XX/5XX is able to provide a 16-bit data bus interface to both the SRAM and EPROM, thus increase system performance and throughput.

The 80C196 bus control signals include the ALE, the  $\overline{RD}$ , the  $\overline{WR}$  and the  $\overline{BHE}$ . It also has a special mode, the Write Strobe Mode. In this mode, the  $\overline{WR}$  and  $\overline{BHE}$  signals are replaced by WRL and WRH. The PSD4XX/5XX supports both interfaces.

**The 80C196 and PSD4XX/5XX Interface Schematic**

Figure 17 shows the 80C196 and PSD4XX/5XX interface schematic. The address/data bus and the bus control signals such as ALE,  $\overline{RD}$ , ER, BHE etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic.

Figure 17. 80C196 and PSD4XX/5XX Interface

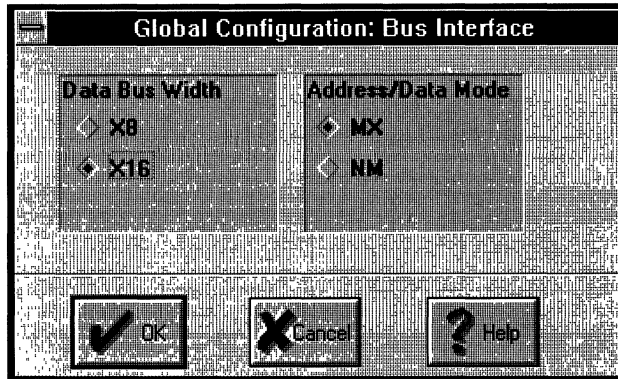


**Interfacing  
To The 80C196  
Family Of  
Microcontrollers  
(Cont.)**

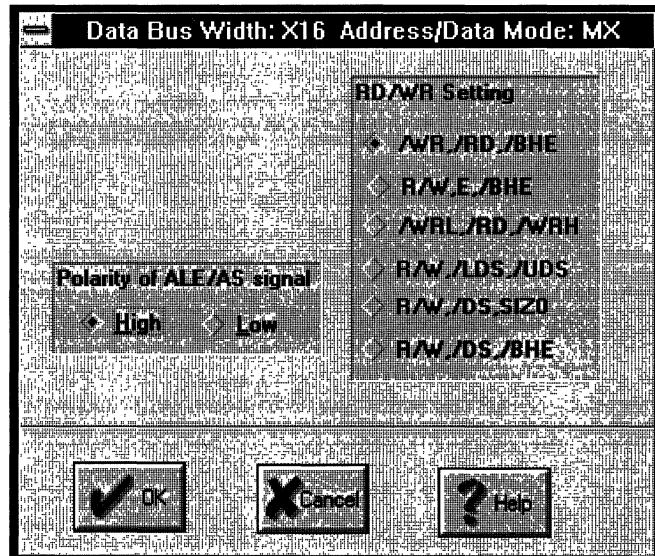
**Specify The 80C196 Bus Interface In PSDconfiguration**

As shown in the following windows captured from PSDconfiguration, specify the 80C196 interface bus by selecting:

- Data Bus Width: X16
- Address/Data Mode: MX
- Polarity of ALE: High
- RD/WR Setting: WR, RD, BHE  
(WRL, RD, WRH for Write Strobe Mode)



3





**Interfacing  
To The 80C196  
Family Of  
Microcontrollers  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL File**

The following is an example of defining the decoding function for the 80C196 based application. The codes are stored in three 32KB EPROM blocks and occupy the same address space from 0000h to 7FFFh. This requires the EPROM blocks to be assigned to 3 different pages. Table 10 illustrates the address map.

Depending on your application, you could also use the GPLD to generate the control signals for the 80C196 “Ready” and the “Buswidth” input.

**Table 10. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 0000 – 7FFF         | Page 0             |
| EPROM, Block 1 | 0000 – 7FFF         | Page 1             |
| EPROM, Block 2 | 0000 – 7FFF         | Page 2             |
| SRAM           | 8000 – 87FF         | All Pages          |
| I/O Devices    | C000 – C0FF         | All Pages          |

module 80C196

title 'Design example of 80C196 DPLD source file';

“Input signals

“Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

pgr3, pgr2, pgr1, pgr0 node; “Page Register outputs

reset pin ;

rst\_out pin 34;

“Output signals

csiop, rs0, es0, es1, es2 node ; “DPLD output chip selects

“DEFINITIONS

page = [pgr3,pgr2,pgr1,pgr0];

X = .x. ; “ Don't care

Address =[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,a1,a0];

equations

“DPLD EQUATIONS

csiop= (Address >= ^hC000) & (Address <= ^hC0FF) ; “Chip Select 256 block

rs0 = (Address >= ^h8000) & (Address <= ^h87FF) ; “ SRAM, 2KB

es0 = (Address >= ^h0000) & (Address <= ^h7FFF) & (page == 0); “EPROM 32KB, page 0

es1 = (Address >= ^h0000) & (Address <= ^h7FFF) & (page == 1); “EPROM 32KB, page 1

es2 = (Address >= ^h0000) & (Address <= ^h7FFF) & (page == 2); “EPROM 32KB, page 2

“GPLD EQUATIONS

rst\_out = reset;

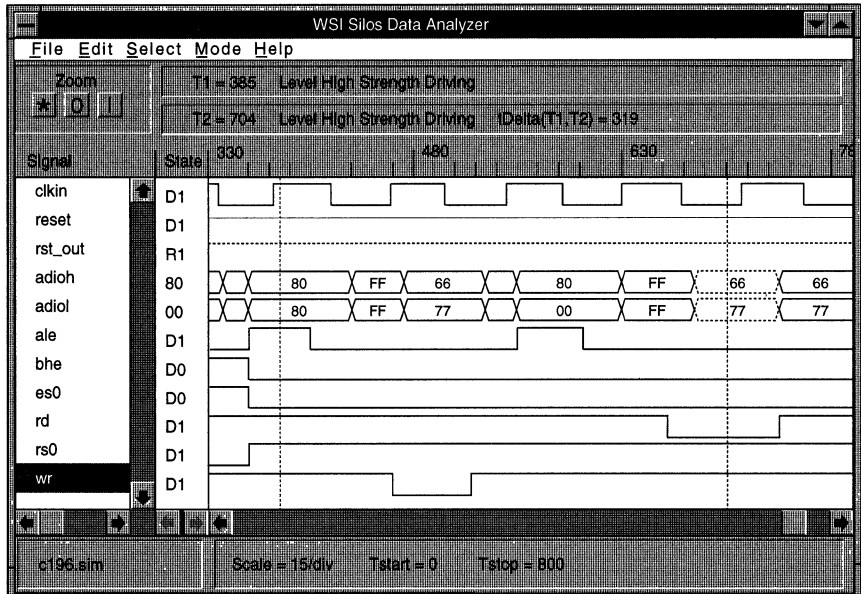
END

## Interfacing To The 80C196 Family Of Microcontrollers (Cont.)

### Simulation Of 80C196 Bus Cycle With The PSD4XX/5XX

Figure 18 shows the simulation output of the SILOS3 Simulator. The 80C196 is writing a word 6677h to SRAM location 8000h and reading back the same location in the next bus cycle.

**Figure 18.**



**Interfacing  
The PSD4XX/5XX  
To The 68302**

**The 68302 Bus**

The Motorola 68302 has a non-multiplexed bus with a 16-bit data bus. It has the following bus signals:

- Address Bus:** A23-A1
- Data Bus** D15-D0
- Address Strobe:** AS
- Control Signals:** R/W,  $\overline{UDS}$ ,  $\overline{LDS}$

The 68302 has no A0 in the address bus; therefore the A0 (ADIO0) pin on the PSD4XX/5XX is grounded. The signals  $\overline{UDS}$  and  $\overline{LDS}$  (Upper and Lower Data Strobe) are used to select whether the low byte, high byte or both bytes for the current bus cycle. See Table 11 for the byte enable assignment.

**Table 11. Byte Enable Assignment**

| $\overline{UDS}$ | $\overline{LDS}$ | D8 – D15 | D0 – D7  |
|------------------|------------------|----------|----------|
| Low              | Low              | Enabled  | Enabled  |
| Low              | High             | Enabled  | Disabled |
| High             | Low              | Disabled | Enabled  |
| High             | High             | Disabled | Disabled |

**The 68302 and PSD4XX/5XX Interface Schematic**

Figure 19 is the 68302 and PSD4XX/5XX interface schematic. The address bus, data bus and bus control signals such as  $\overline{LDS}$ ,  $\overline{UDS}$ , R/W, AS etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic. Please note A0 pin on the PSD4XX/5XX is grounded.

For Motorola 16-bit microcontrollers, the data byte D0 – D7 is considered as an odd byte and D8 – D15 as an even byte. This is just the reverse of Intel and other similar processors. If you select a Motorola 16-bit bus interface, the PSDcompiler automatically swaps these bytes such that D0-D7 is programmed to even byte locations and D8 – 15 is programmed to odd byte locations in the PSD EPROM. This swapping is transparent to the user. In the interface schematic, connect D0 – D7 from the 68302 to Port D (Data Port D0 – D7) and D8 – D15 from the 68302 to Port D (Data Port D8 – D15).

The  $\overline{CS0}$  signal from the 68302 can be connected to the CSI pin on the PSD4XX/5XX for power management. If the 68302 is not fetching code from the PSD,  $\overline{CS0}$  is high and thus puts the PSD into power saving Standby Mode.



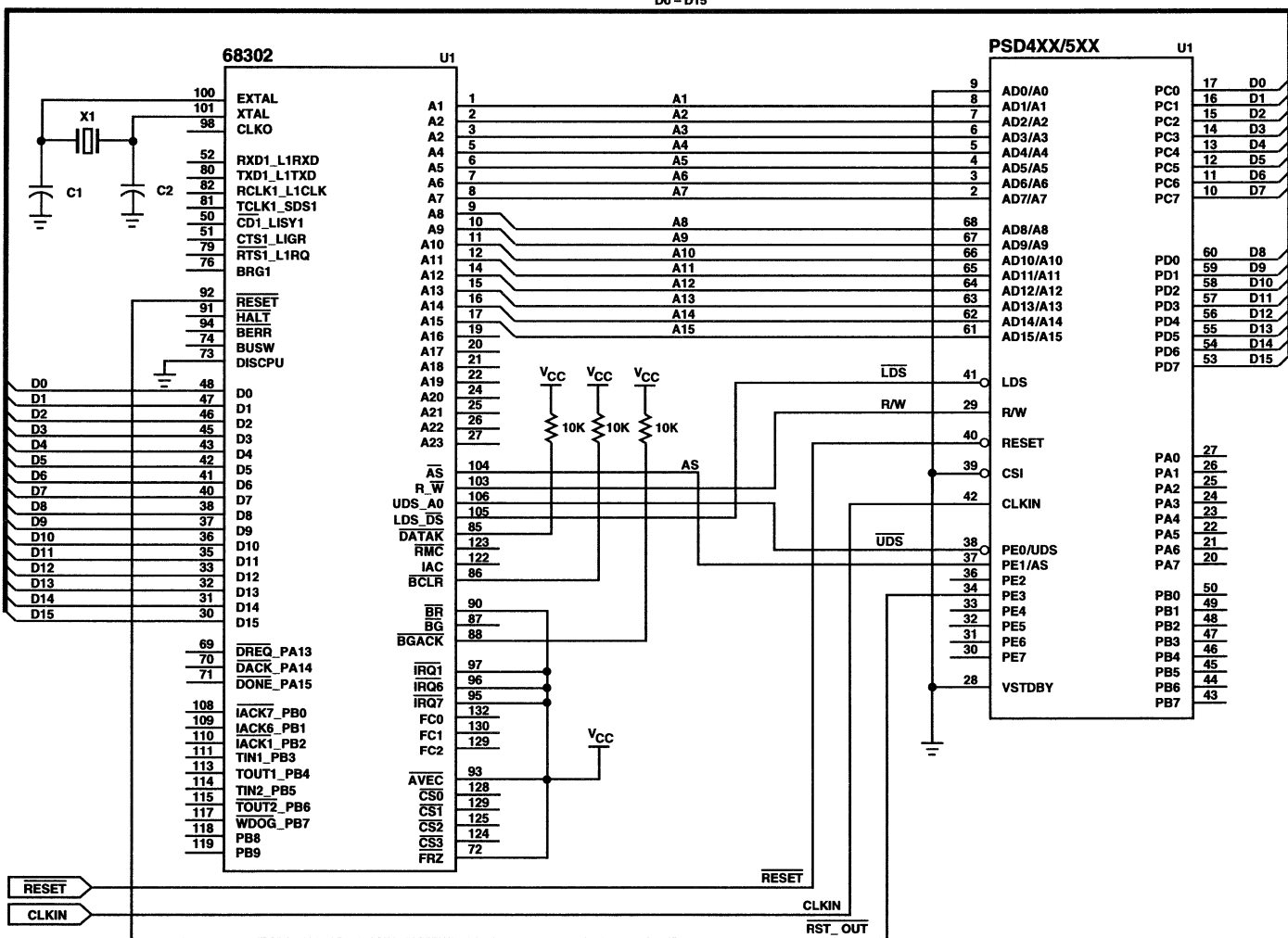


Figure 19. 68302 and PSD4XX/5XX Interface



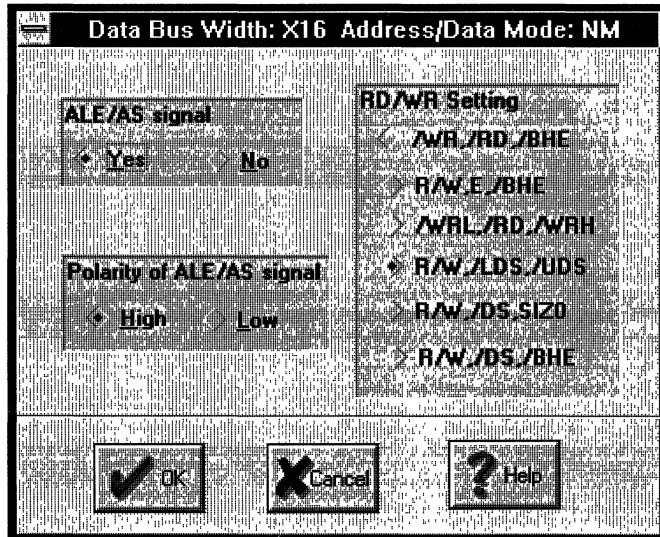
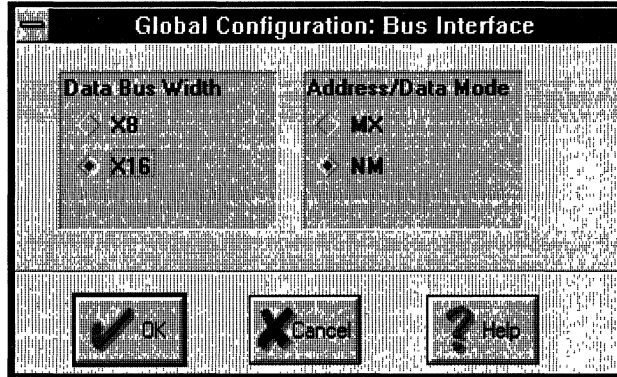
**Interfacing  
The PSD4XX/5XX  
To The 68302**

(Cont.)

**Specify The 68302 Bus Interface In PSDConfiguration**

As shown in the following windows captured from PSDconfiguration, specify the 68302 bus interface by selecting:

- Data Bus Width: X16
- Address/Data Mode: NM
- ALE/AS signal: Yes (No if you prefer not to use AS to latch address)
- Polarity of ALE: High (if Yes on ALE/AS)
- RD/WR Setting: R/W, LDS, UDS



**Interfacing  
The  
PSD4XX/5XX  
To The 68302  
(Cont.)**

**Define the DPLD/Decoding function in the ABEL file**

The following is an example of defining the decoding function for the 68302 based application. The codes are stored in three 32KB EPROM blocks and occupy the same address space from 0000h to 7FFFh. This requires the EPROM blocks to be assigned to 3 different pages. Table 12 illustrates the address map.

**Table 12. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 0000 – 3FFF         | All Pages          |
| EPROM, Block 1 | 4000 – 7FFF         | Page 1             |
| EPROM, Block 2 | 4000 – 7FFF         | Page 2             |
| SRAM           | 8000 – 87FF         | All Pages          |
| I/O Devices    | C000 – C0FF         | All Pages          |

module 68302

title 'example of 68302 DPLD source file ';

"Input signals

"Address lines, using reserved names.  
a15,a14,a13,a12,a11,a10,a9,a8,a1 pin;

"Use the reserved names to declare the following special functions  
reset pin;  
rst\_out pin 34;

" Output signals

" Internal PSD5XX PLD output signals.

"DPLD outputs using reserved names.  
csiop, rs0, es0, es1, es2, es3 node ;

" Definitions

" Don't care  
X = .x. ;

"Note in the Address definition that a7 - a2 are denoted by don't-cares  
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,a1,X];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hC000) & (Address <= ^hC0FF) ; "Chip Select 256 block

rs0 = (Address >= ^h8000) & (Address <= ^h87FF) ; "SRAM, 2KB

es0 = (Address >= ^h0000) & (Address <= ^h3FFF) & (page == X);

"EPROM 16KB, any page

es1 = (Address >= ^h4000) & (Address <= ^h7FFF) & (page == 1); "EPROM 16KB, page 1

es2 = (Address >= ^h4000) & (Address <= ^h7FFF) & (page == 2); "EPROM 16KB, page 2

"GPLD EQUATIONS

rst\_out = reset;

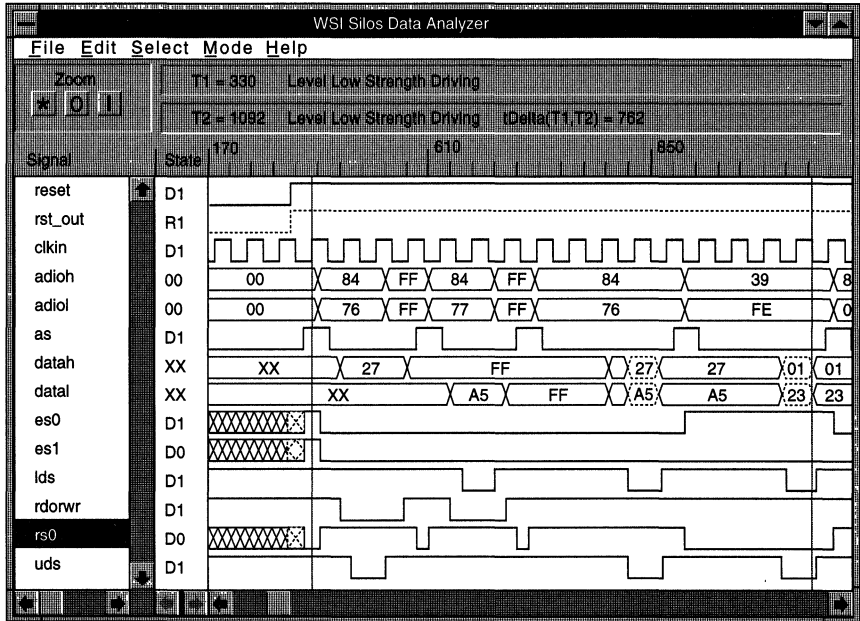
END

**Interfacing  
The  
PSD4XX/5XX  
To The 68302  
(Cont.)**

**Simulation Of 68302 Bus Cycle With The PSD4XX/5XX**

Figure 20 shows the simulation of 3 bus cycles of the 68302. The 68302 is writing a byte to SRAM location 8477h and location 8476h. The third bus cycle is a word read to the same locations.

**Figure 20.**



**Interfacing  
The  
PSD4XX/5XX  
To  
68HC16/68330/  
331/332/340**

**The 683XX Bus**

This group of Motorola 16-bits microcontrollers have similar bus structure and the bus interface to the PSD4XX/5XX are identical. The 68332 microcontroller is used here as an example. The bus is a non-multiplexed data and address bus and has the following signals:

- Address Bus:** A23-A0
- Data Bus:** D15-D0
- Address Strobe:** AS
- Control Signals:**  $\overline{DS}$ , R/W, SIZ0, SIZ1

The higher address pins A23-A19 can be configured either as address lines or as chip select outputs ( $\overline{CS6} - \overline{CS10}$ ) at reset time. Two of the signals, SIZ0 and A0 are used to determine whether the current cycle is a byte or a word operation. If SIZ0 is low, it is always a word operation. If SIZ0 is high, it is a byte operation and A0 determines which byte is enabled.

The PSD4XX/5XX generates internal write or read pulses based on the status of the R/W and  $\overline{DS}$  signal inputs.

**The 68332 And PSD4XX/5XX Interface Schematic**

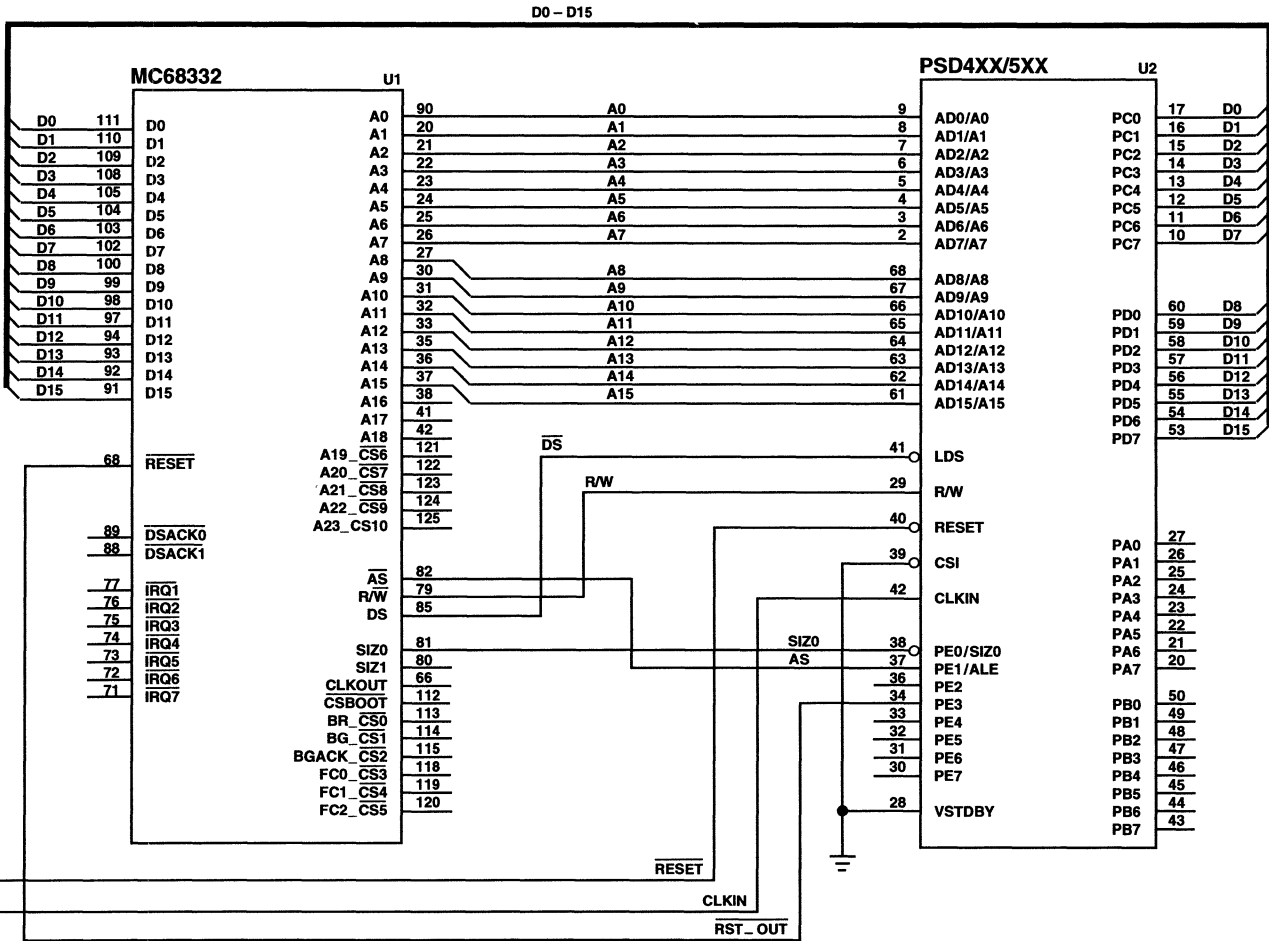
Figure 21 is the 68332 and PSD4XX/5XX interface schematic. The address bus, data bus and the bus control signals such as  $\overline{DS}$ , R/W, SIZ0, AS etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic.

For Motorola 16-bit microcontrollers the data byte D0 – D7 is considered as odd byte and D8 – D15 as even byte, which is the reverse of Intel and other similar processors. If you select a Motorola 16-bit bus interface, the PSDcompiler automatically swaps these bytes such that D0 – D7 is programmed to even byte locations and D8 – 15 is programmed to odd byte locations in the PSD EPROM. This swapping is transparent to the user. In the interface schematic, connect D0 – D7 from the 68332 to Port C (Data Port D0 – D7) and D8 – D15 to Port D (Data Port D8 – D15).

The  $\overline{CSBOOT}$  signal from the 68332 can be connected to the CSI pin on the PSD4XX/5XX to control the device power consumption. If the 68332 is not fetching code from the PSD, the  $\overline{CSBOOT}$  is high and puts PSD4XX/5XX into power saving Standby Mode. After system reset, the  $\overline{CSBOOT}$  has a default value of 1M byte memory space starting from address 000000h. This value can be re-programmed after system initialization to include the PSD EPROM, SRAM and I/O space.



Figure 21. 68332 and PSD4XX/5XX Interface

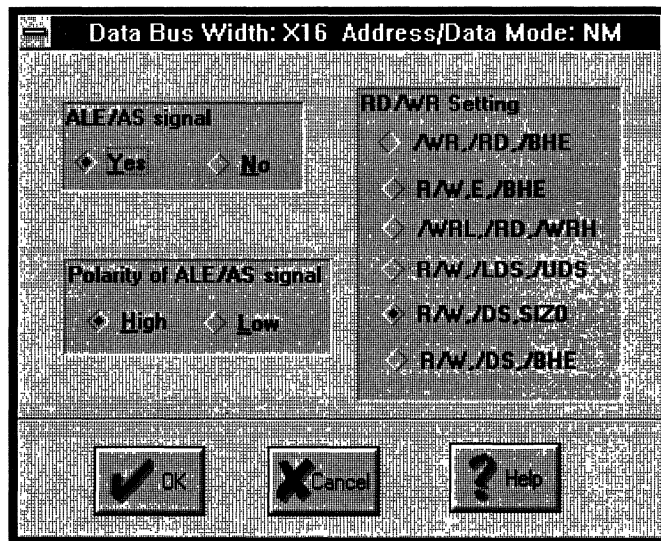
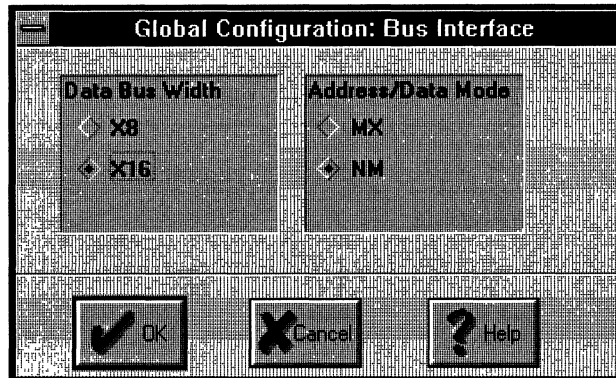


**Interfacing  
The  
PSD4XX/5XX  
To  
68HC16/68330/  
331/332/340  
(Cont.)**

**Specify the 68332 Bus Interface in PSDConfiguration**

As shown in the following windows which are captured from PSDconfiguration, the 68332 bus interface can be specified by selecting:

- Data Bus Width: X16
- Address/Data Mode: NM
- ALE/AS signal: Yes (No if you prefer not to use AS to latch address)
- Polarity of ALE: High (if Yes on ALE/AS)
- RD/WR Setting: R/W, DS, SIZ0



3

**Interfacing  
The  
PSD4XX/5XX  
To  
68HC16/68330/  
331/332/340  
(Cont.)**

**Define the DPLD/Decoding function in the ABEL file**

The following is an example of defining the decoding function for the 68332 based application. The codes are stored in three 32KB EPROM blocks and occupy the same address space from 0000h to 7FFFh. This requires the EPROM blocks to be assigned to 3 different pages. Table 13 illustrates the address map.

**Table 13. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 0000 – 3FFF         | All Pages          |
| EPROM, Block 1 | 4000 – 7FFF         | Page 1             |
| EPROM, Block 2 | 4000 – 7FFF         | Page 2             |
| SRAM           | 8000 – 87FF         | All Pages          |
| I/O Devices    | C000 – C0FF         | All Pages          |

```

module 68332
title 'example of 68332 DPLD source file';

"Input signals

"Address lines, using reserved names.
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

"Use the reserved names to declare the following special functions
reset pin;
rst_out pin 34;

" Output signals

" Internal PLD output signals.

"DPLD outputs using reserved names.
csiop, rs0, es0, es1, es2 node;

" Definitions

" Don't care
X = .x.;

"Note in the Address definition that a7 - a2 are denoted by don't-cares
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hC000) & (Address <= ^hC0FF); "Chip Select 256 block
rs0 = (Address >= ^h8000) & (Address <= ^h87FF); "SRAM, 2KB
es0 = (Address >= ^h0000) & (Address <= ^h3FFF) & (page == X);
 " EPROM 16KB , any page
es1 = (Address >= ^h4000) & (Address <= ^h7FFF) & (page == 1); "EPROM 16KB, page 1
es2 = (Address >= ^h4000) & (Address <= ^h7FFF) & (page == 2); "EPROM 16KB, page 2

"GPLD EQUATIONS

rst_out = reset;

END

```

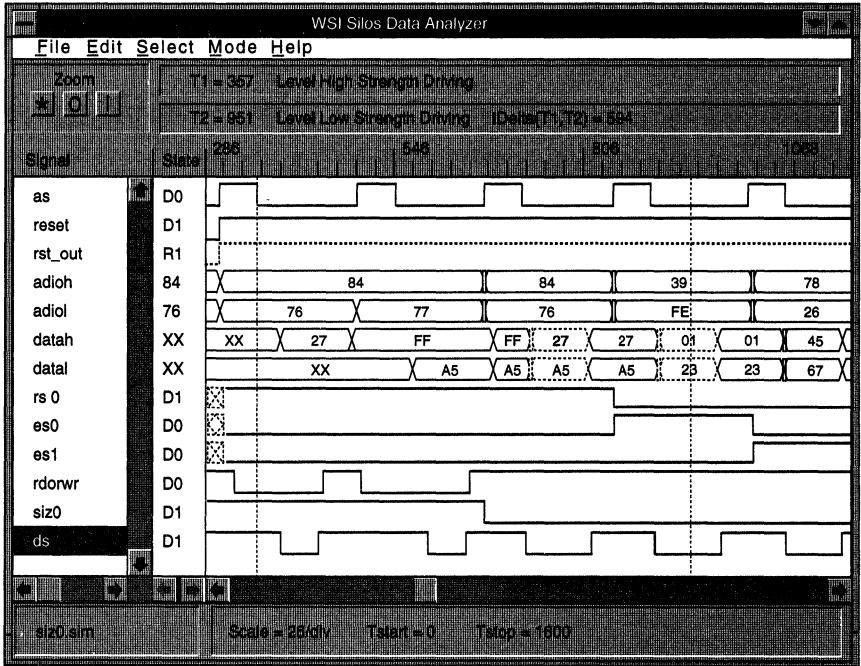


**Interfacing  
The  
PSD4XX/5XX  
To  
68HC16/68330/  
331/332/340  
(Cont.)**

**Simulation Of 68332 Bus Cycle With The PSD4XX/5XX**

Figure 22 shows the simulation of five 68332 bus cycles. The first two are byte write cycles to SRAM locations 8476 and 8477. The next cycle is a word read to the same location. The next cycle is reading an EPROM block.

**Figure 22.**



3

## **Interfacing The PSD4XX/5XX To Z8**

### **The Z8 Bus**

The Z8 has an 8-bit multiplexed external memory bus. Port 1 of the Z8 is used as the multiplexed bus port which provides the multiplexed lower address byte and data. Port 0 can be used as the output port for the non-multiplexed address lines A15-A8. The bus has the following signals:

- Address/Data Bus:** AD7 – A0
- Address Bus:** A15 – A8
- Address Strobe:**  $\overline{AS}$
- Control Signals:** DS, R/W, DM

The Z8 has 64KB of program memory space. It can also address another 64KB of data memory if the signal  $\overline{DM}$  (data memory) is enabled. The signal  $\overline{DM}$  can be programmed to appear on pin 4 of Port 3. If your application does not require separate data space, there is no need to connect the  $\overline{DM}$  as input to the PSD4X/5XX.

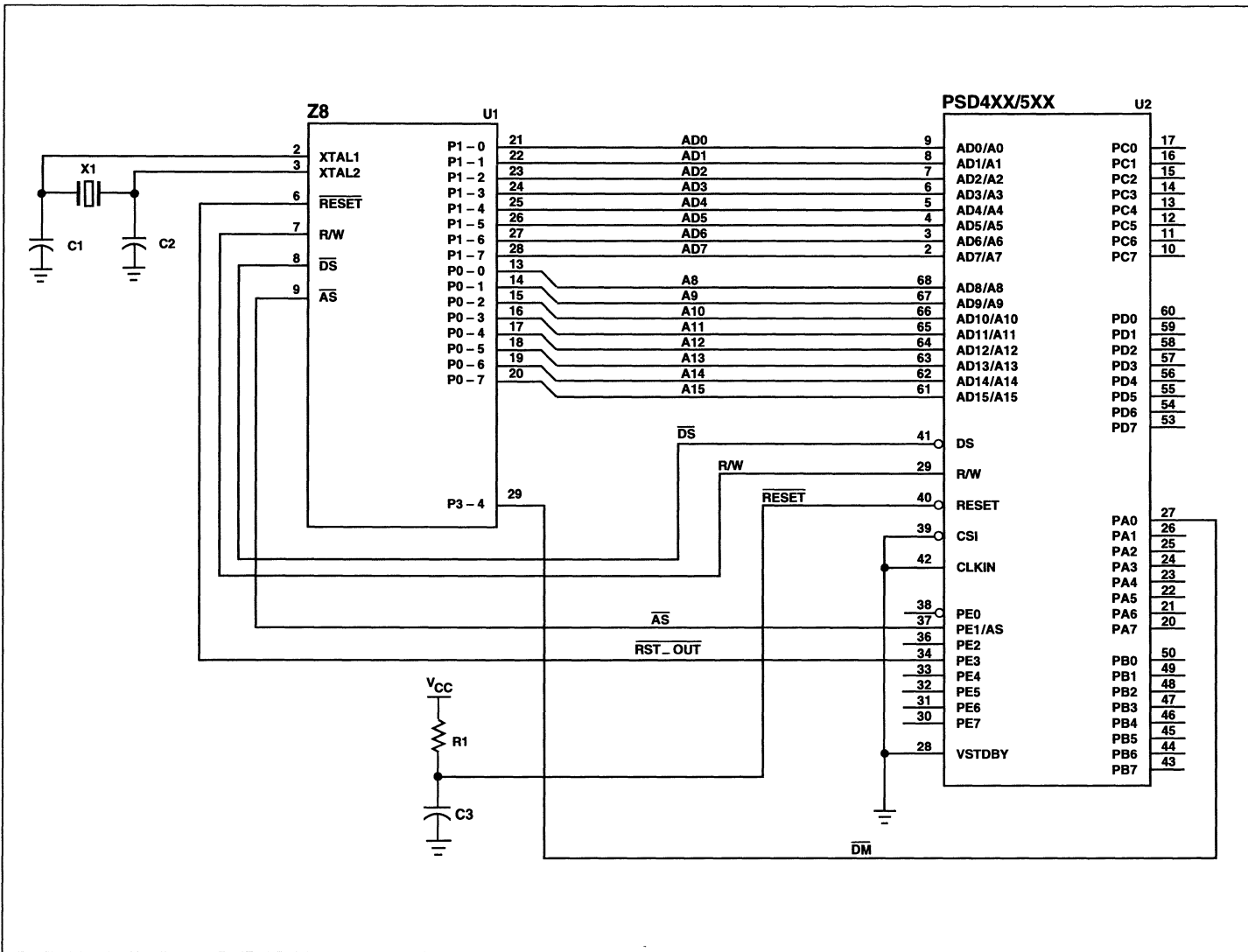
---

### **The Z8 And PSD4XX/5XX Interface Schematic**

Figure 23 is the Z8 and PSD4XX/5XX interface schematic. The address bus, data bus and the bus control signals such as  $\overline{DS}$ , R/W,  $\overline{AS}$  etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic.

In this example,  $\overline{DM}$  is used to separate the program space from the data space by including it in the DPLD chip select equations. Due to the PSD4XX/5XX architecture requirement that any input signals which are included in the EPROM chip select equations must come from Port A, the  $\overline{DM}$  signal is connected to pin PA0 in the schematic.

Figure 23. Z8 and PSD4XX/5XX Interface

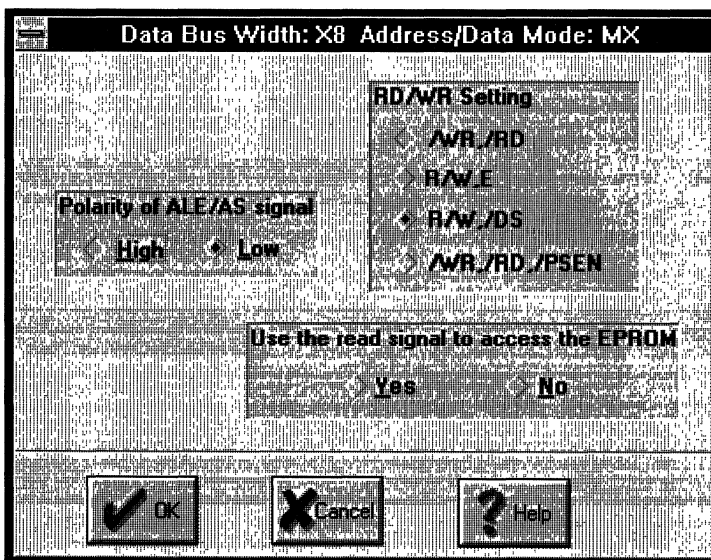
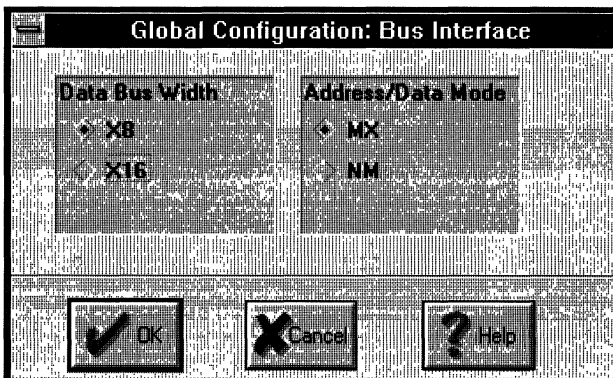


**Interfacing  
The  
PSD4XX/5XX  
To Z8  
(Cont.)**

**Specify The Z8 Bus Interface In PSDconfiguration**

As shown in the following windows which are captured from PSDconfiguration, the Z8 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: MX
- Polarity of ALE/AS: Low
- RD/WR Setting: R/W,  $\overline{DS}$



**Interfacing  
The  
PSD4XX/5XX  
To Z8  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL File**

The following is an example of defining the decoding function for the Z8 based application. 64KB of code is stored in EPROM blocks 0 and 1. The SRAM, I/O space and EPROM block 2 are assigned as data memory. Table 14 illustrates the address map.

**Table 14. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> |           |
|----------------|---------------------|-----------|
| EPROM, Block 0 | 0000 – 7FFF         | Code Area |
| EPROM, Block 1 | 8000 – FFFF         | Code Area |
| EPROM, Block 2 | 0000 – 7FFF         | Data Area |
| SRAM           | 8000 – 87FF         | Data Area |
| I/O Devices    | C000 – C0FF         | Data Area |

```

module Z8
title 'example of Z8 DPLD source file';

"Input signals

"Address lines, using reserved names.
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

"Use the reserved names to declare the following special functions
reset pin;
dm pin 27 "assign pin PA0 as input pin for DM
rst_out pin 34;

" Output signals

" Internal PLD output signals.

"DPLD outputs using reserved names.
csiop, rs0, es0, es1, es2, es3 node;

" Definitions

" Don't care
X = .x;

"Note in the Address definition that a7 - a2 are denoted by don't-cares
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hC000) & (Address <= ^hC0FF) & !DM; "Chip Select 256 block
rs0 = (Address >= ^h8000) & (Address <= ^h87FF) & !DM; " SRAM, 2KB
es0 = (Address >= ^h0000) & (Address <= ^h7FFF) & DM; " EPROM 32KB code
es1 = (Address >= ^h8000) & (Address <= ^hFFFF) & DM; " EPROM 32KB code
es2 = (Address >= ^h0000) & (Address <= ^h7FFF) & !DM; " EPROM 32KB data

"GPLD EQUATIONS

rst_out = reset;

END

```

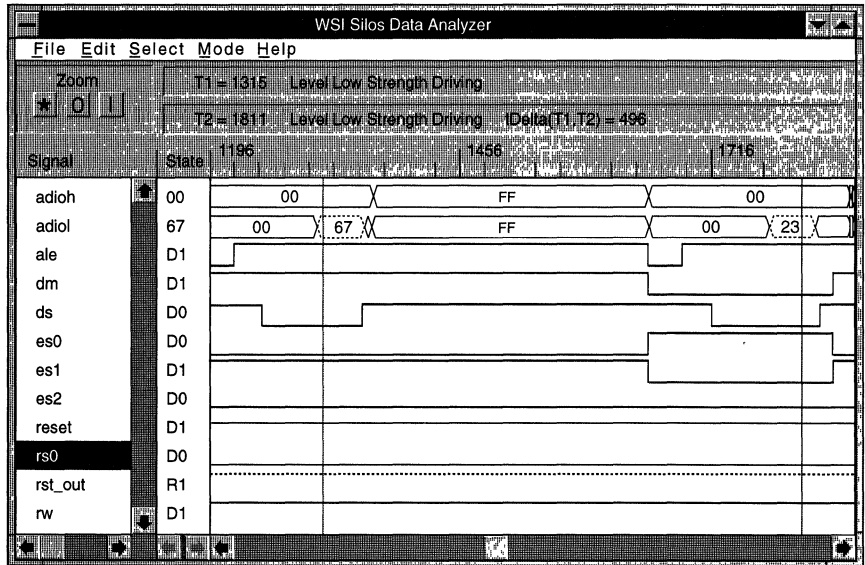


**Interfacing  
The  
PSD4XX/5XX  
To Z8  
(Cont.)**

**Simulation Of Z8 Bus Cycle With The PSD4XX/5XX**

Figure 24 shows the simulation of two Z8 bus cycles. The first is a code fetch at location 0000h from EPROM block 0, with /DM input high. The second cycle is a data read at location 0000h from EPROM block 1. The /DM signal is low since this is a data memory bus cycle.

**Figure 24.**



**Interfacing  
The  
PSD4XX/5XX  
To Z80  
(Cont.)**

**The Z80 Bus**

The Z80 has an 8-bit non-multiplexed bus. The following signals are used to interface to memory or I/O devices:

- **Address Bus:** A15 – A0
- **Data Bus:** D7 – D0
- **Address Strobe:** None
- **Control Signals:**  $\overline{M1}$ ,  $\overline{MREQ}$ ,  $\overline{IORQ}$ ,  $\overline{RD}$ ,  $\overline{WR}$

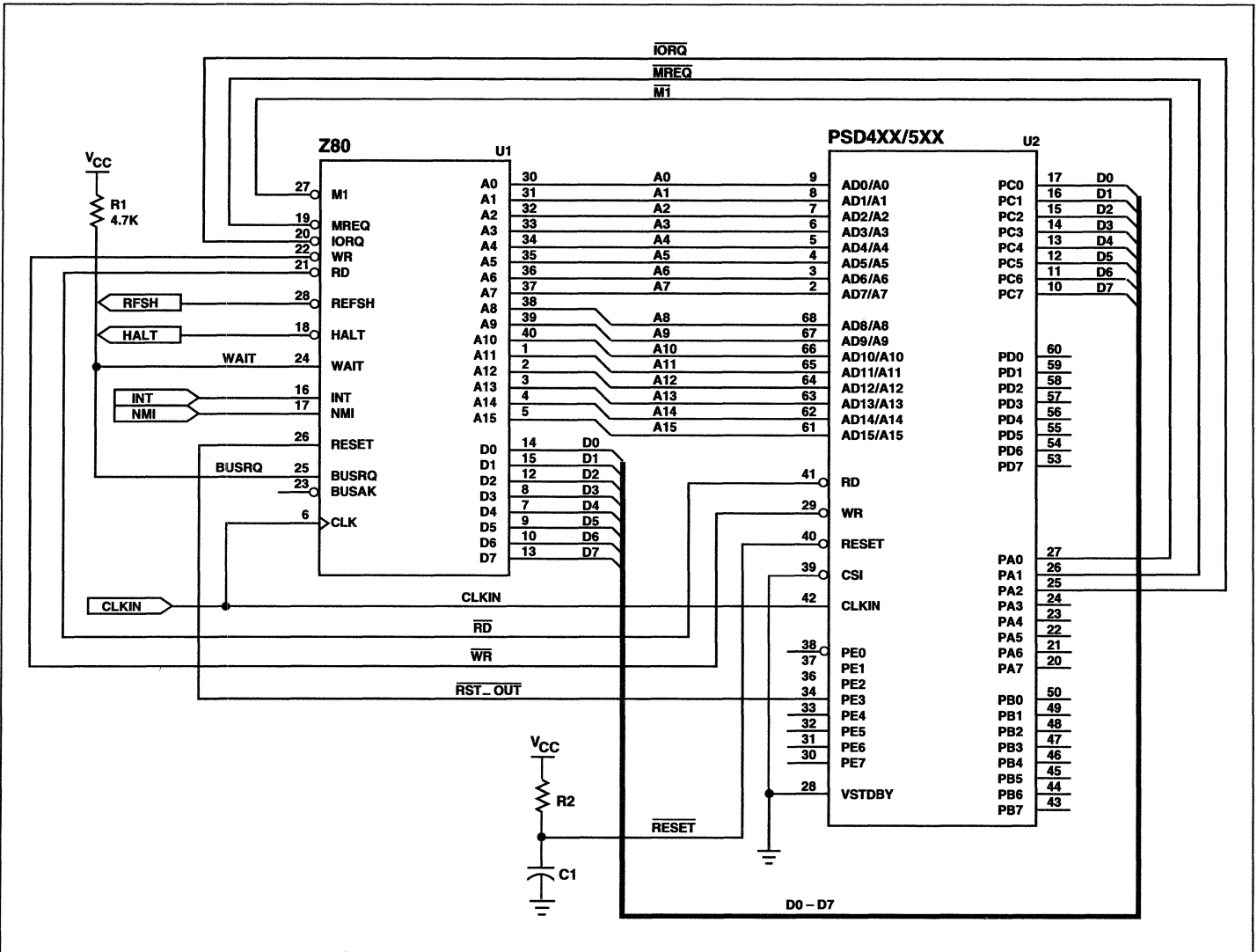
The Z80 has 64KB of program memory space and 256 bytes of I/O space. In a memory cycle both  $\overline{M1}$  and  $\overline{MREQ}$  are low. In an I/O bus cycle,  $\overline{M1}$  is high and  $\overline{IORQ}$  is low. Only A7-A0 are active and thus limit the I/O space to 256 bytes. If  $\overline{M1}$  is low and  $\overline{IORQ}$  is low, it is an interrupt acknowledge bus cycle.  $\overline{M1}$  can be ignored if interrupt is not used.

**The Z80 And PSD4XX/5XX Interface Schematic**

Figure 25 is the Z80 and PSD4XX/5XX interface schematic. The address lines A15 – A0 are connected to the ADIO Port and the data lines D7 – D0 are connected to Port C. Control signals  $\overline{RD}$  and  $\overline{WR}$  are directly connected to the corresponding pins of the PSD4XX/5XX without any additional glue logic.

The PSD4XX/5XX does not have specific pins assigned to  $\overline{MREQ}$ ,  $\overline{IORQ}$  and  $\overline{M1}$ . Since these signals are used in the EPROM chip select equations, you should assign them to Port A pins in the ABEL file.

Figure 25. Z80 and PSD4XX/5XX Interface

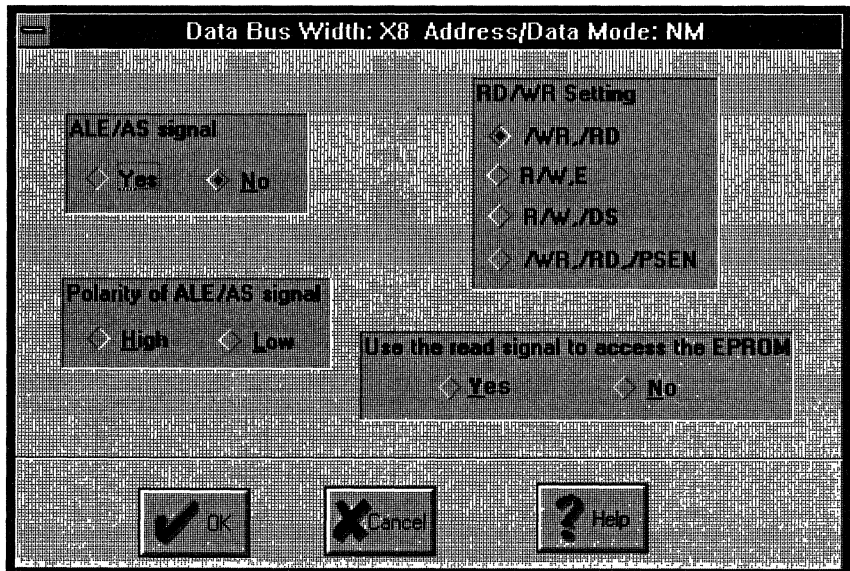
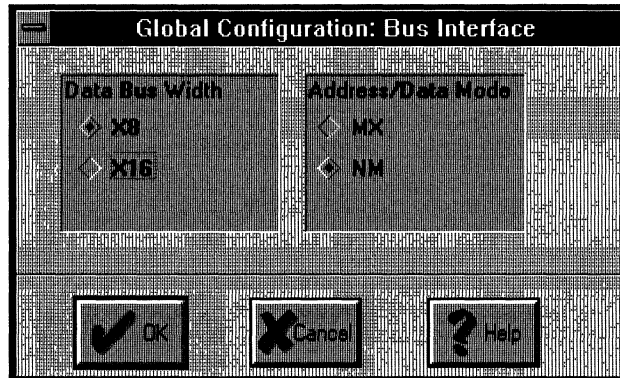


**Interfacing  
The  
PSD4XX/5XX  
To Z80  
(Cont.)**

**Specify The Z80 Bus Interface In PSDconfiguration**

As shown in the following windows which are captured from PSDconfiguration, the Z80 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: NM
- ALE/AS signal: No
- RD/WR Setting:  $\overline{RD}$ ,  $\overline{WR}$



**Interfacing  
The  
PSD4XX/5XX  
To Z80  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL File**

The following is an example of defining the decoding function for the Z80 based application. Please note the 256 bytes of I/O space are all taken by the PSD4XX/5XX internal I/O devices, and that the CSiOP signal becomes active whenever /IORQ is active.

You could define a 256 byte block of memory space to the CSiOP chip select. In this case, you have to use memory reference instructions to access the PSD4XX/5XX I/O devices. Table 15 illustrates the address map. The CSiOP completely takes up 256 bytes of I/O space and is enabled whenever IORQ is active.

**Table 15. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> |      |
|----------------|---------------------|------|
| EPROM, Block 0 | 0000 – 3FFF         | Code |
| EPROM, Block 1 | 4000 – 7FFF         | Code |
| EPROM, Block 2 | 8000 – BFFF         | Data |
| SRAM           | C000 – C7FF         | Data |

```

module z80
title ' example of z80 DPLD source file';

"Input signals

"Address lines, using reserved names.
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

reset pin;
rst_out pin 34;

mreq pin 27; " assign mreq input to Port A pin PA0
iorq pin 26; " assign ioreq input to Port A pin PA1
m1 pin 25; " assign m1 input to Port A pin PA2

"Output signals

csiop, rs0, es0, es1, es2 node ; "DPLD output chip selects

"DEFINITIONS
X = .x ; " Don't care
Address =
[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = !iorq & m1; "I/O Chip Select 256 bytes
rs0 = (Address >= ^hC000) & (Address <= ^hC7FF) & !mreq "SRAM, 2KB
es0 = (Address >= ^h0000) & (Address <= ^h3FFF) & !mreq; "EPROM 16KB code
es1 = (Address >= ^h4000) & (Address <= ^h7FFF) & !mreq; "EPROM 16KB code
es2 = (Address >= ^h8000) & (Address <= ^hBFFF) & !mreq; "EPROM 16KB data

rst_out = reset;

END

```

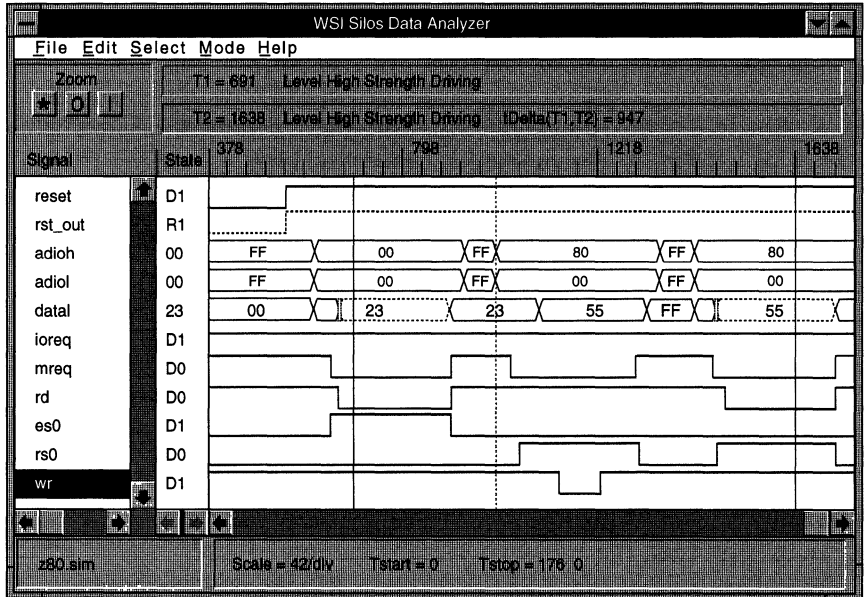


**Interfacing  
The  
PSD4XX/5XX  
To Z80  
(Cont.)**

**Simulation Of Z80 Bus Cycle With The PSD4XX/5XX**

Figure 26 shows the simulation of three Z80 bus cycles. The first is a code fetch at location 0000h from EPROM block 0. The second cycle is a data write to SRAM at location 8000h, and a read to the same location in the next cycle.

**Figure 26.**



3

**Interfacing  
The  
PSD4XX/5XX  
To ST90R26**

**The ST90R26 Bus**

The ST90R26 is the ROMless member of the ST9 family of microcontrollers from SGS-Thomson. The ST9 has an 8-bit multiplexed bus and the following are the bus signals used to interface to memory or I/O devices.

- Address/Data Bus:** AD7-AD0
- Address Bus:** A15-A8
- Address Strobe:**  $\overline{AS}$
- Control Signals:**  $\overline{DS}$ , R/W, P/D

The higher address lines A15-A8 are not multiplexed and are driven from Port P1. The ST9 has two memory spaces: the program and data memory. Each space has 64KB and is selected by the P/D signals. A high on the P/D signal indicates program space.

The PSD4XX/5XX generates internal write or read pulses based on the status of the R/W and  $\overline{DS}$  signal inputs.

---

**The ST90R26 And PSD4XX/5XX Interface Schematic**

Figure 27 is the ST90R26 and PSD4XX/5XX interface schematic. The address bus, data bus and the bus control signals such as  $\overline{DS}$ , R/W,  $\overline{AS}$  etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic. The P/D signal is connected to one of the pins in Port A as input to the DPLD.

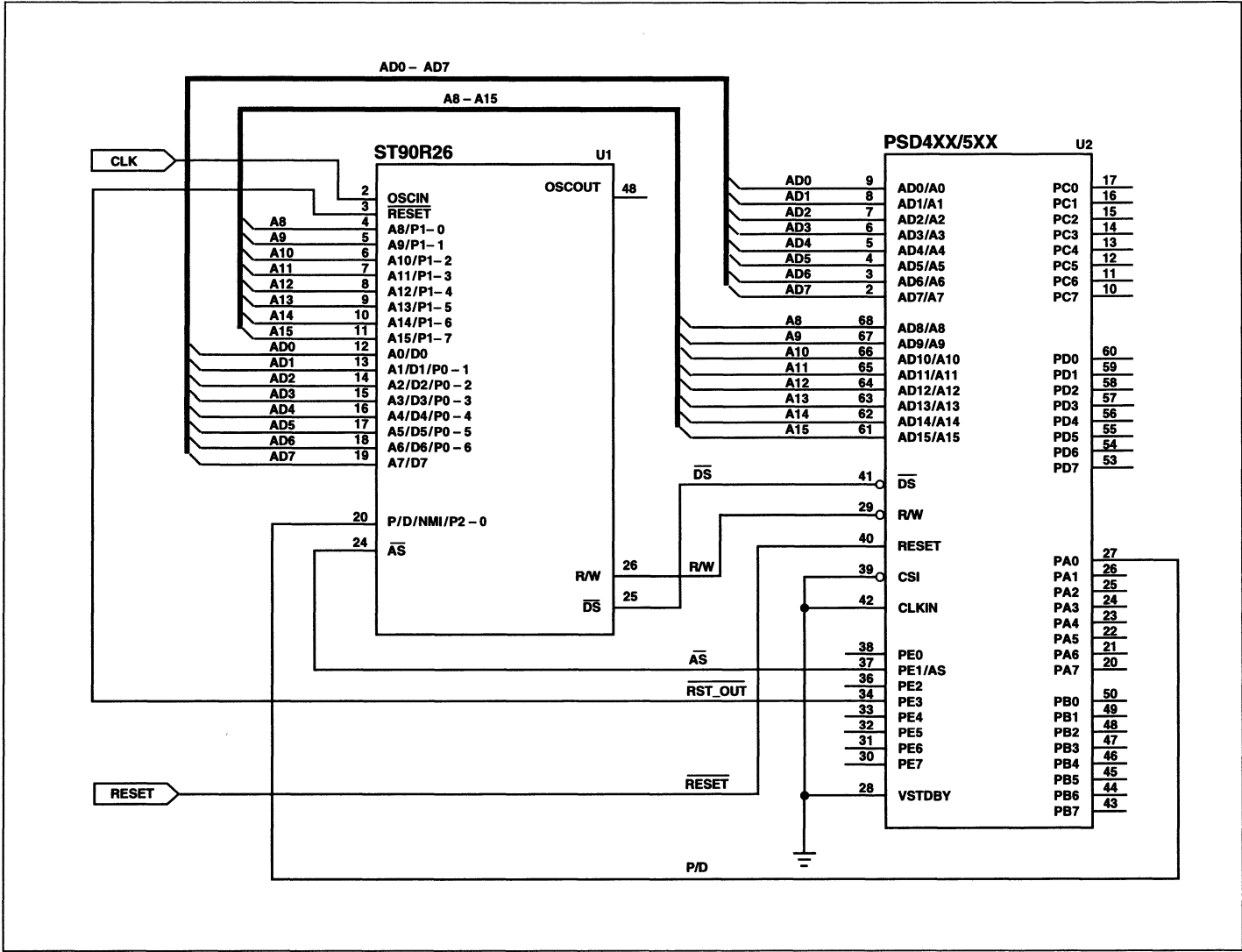


Figure 27. ST90R26 and PSD4XX/5XX Interface

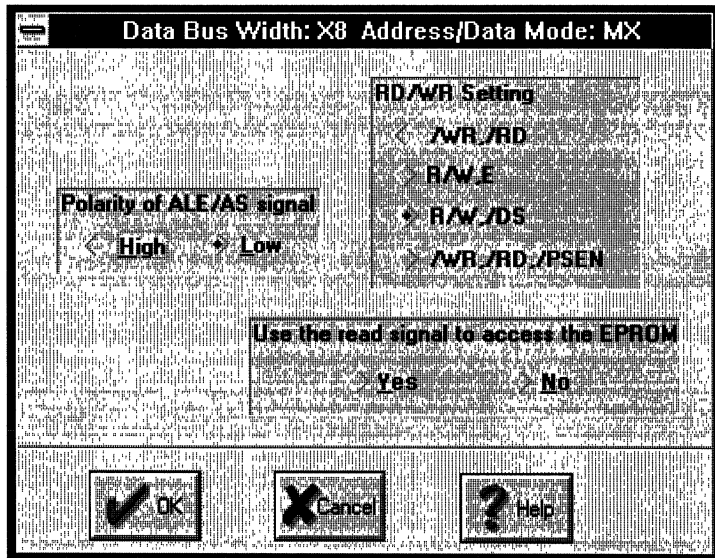
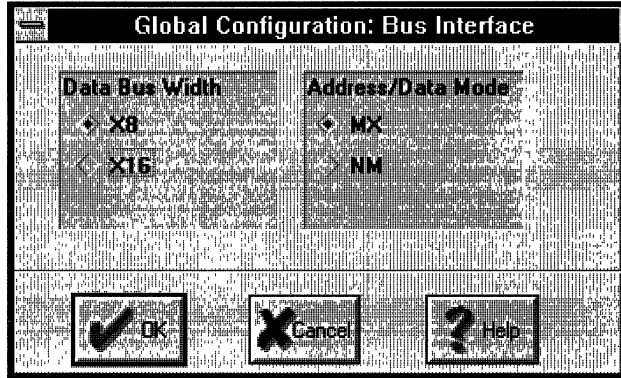


**Interfacing  
The  
PSD4XX/5XX  
To ST90R26  
(Cont.)**

**Specify The ST90R26 Bus Interface In PSDconfiguration**

As shown in the following windows which are captured from PSDconfiguration, the ST90R26 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: MX
- ALE/AS signal: Yes
- Polarity of ALE: Low
- RD/WR Setting: R/W,  $\overline{DS}$



**Interfacing  
The  
PSD4XX/5XX  
To ST90R26  
(Cont.)**

**Define The DPLD/Decoding Function In the ABEL File**

The following is an example of defining the decoding function for the ST9 based application. The codes are stored in three 16KB EPROM blocks and occupy address space from 0000h to BFFFh. The SRAM space is from 8000h to 87FFh. The P/D input is used to separate the EPROM (program) space to SRAM and I/O (data) space.

**Table 16. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> |      |
|----------------|---------------------|------|
| EPROM, Block 0 | 0000 – 3FFF         | Code |
| EPROM, Block 1 | 4000 – 7FFF         | Code |
| EPROM, Block 2 | 8000 – BFFF         | Code |
| SRAM           | 8000 – 87FF         | Data |
| I/O Devices    | A000 – A0FF         | Data |

```

module st9
title 'example of st9 DPLD source file';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

reset pin; "using the right pin #s
pd pin 27; "port A pin-0 for P/D input

"Output signals

csiop, rs0, es0, es1, es2 node ; "DPLD output chip selects
rst_out pin 34;

"DEFINITIONS

X = .x. ; " Don't care
Address =
[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hA000) & (Address <= ^hA0FF) & !pd ; "Chip Select 256 block
rs0 = (Address >= ^h8000) & (Address <= ^h87FF) & !pd; "SRAM, 2KB
es0 = (Address >= ^h0000) & (Address <= ^h3FFF) & pd; "EPROM 16KB
es1 = (Address >= ^h4000) & (Address <= ^h7FFF) & pd; "EPROM 16KB
es2 = (Address >= ^h8000) & (Address <= ^hBFFF) & pd; "EPROM 16KB

"GPLD EQUATIONS

rst_out = reset;

END

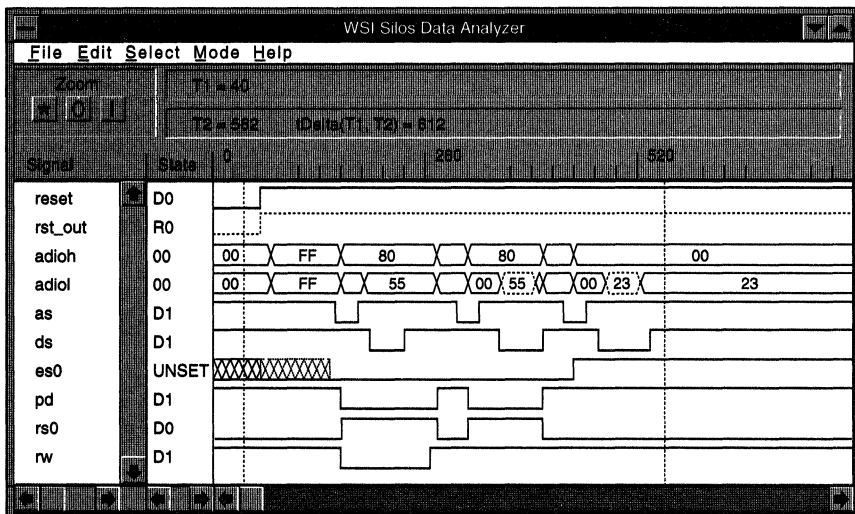
```

**Interfacing  
The  
PSD4XX/5XX  
To ST90R26  
(Cont.)**

**Simulation Of The ST90R26 Bus Cycle With The PSD4XX/5XX**

Figure 28 shows the simulation of three ST90R26 bus cycles. The first two cycles are byte write (55h) and read to SRAM location 8000h, and the third is a code fetch cycle to EPROM location 0000h. Please note that the P/D separates the data and program space.

**Figure 28.**



## **Interfacing The PSD4XX/5XX To 80C166**

### **The 80C166 Bus**

The Siemens' 80C166 is a very flexible microcontroller which can be operated in multiplexed or non-multiplexed bus mode. The bus configuration and data bus width (8 or 16) are determined at reset by sampling the EBC0-1 input pins. The multiplexed 16-bit data/16 bit address bus mode is selected here for PSD4XX/5XX implementation since it provides the best performance with the least pin count.

The 16-bit multiplexed bus consists of the following signals:

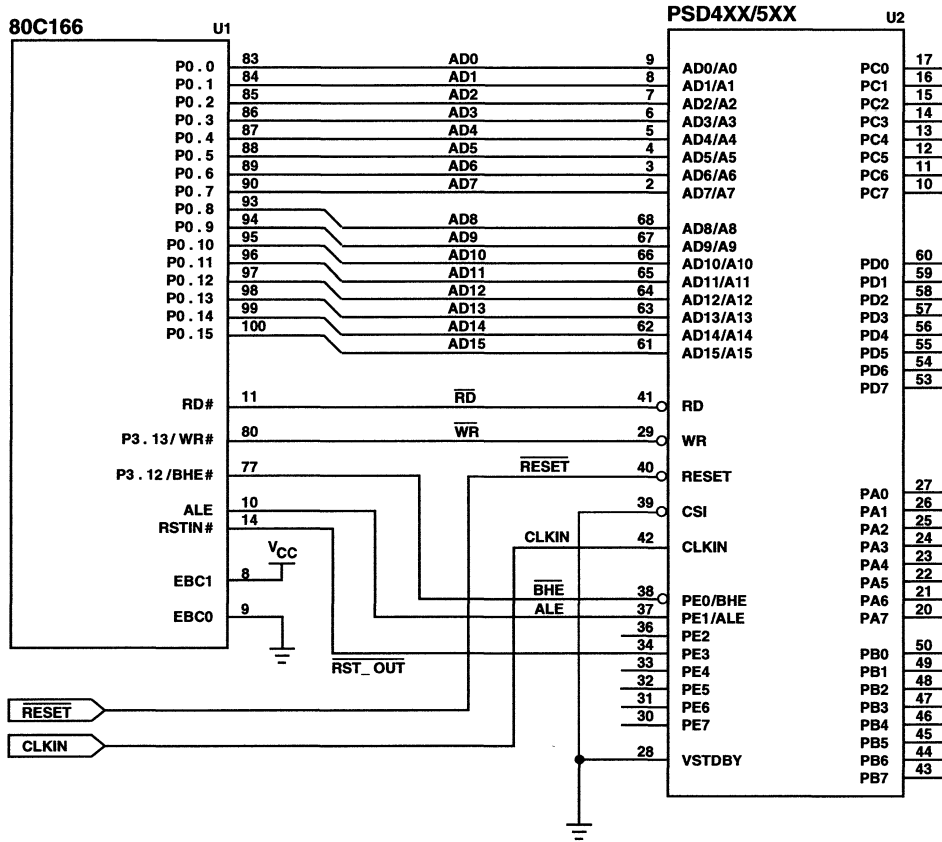
- Address/Data:** AD15-AD0
- Address Latch:** ALE
- Control Signals:**  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{BHE}$

The 80C166 also provides higher address lines A16-17 (segment address) if required.

### **The 80C166 And PSD4XX/5XX Interface Schematic**

Figure 29 shows the 80C166 and PSD4XX/5XX interface schematic. The address bus, data bus and bus control signals such as  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{BHE}$  etc., are directly connected to the corresponding pins of PSD4XX/5XX without any additional glue logic. Note that EBC0 is connected to ground and EBC1 is connected to  $V_{CC}$  to select the 16-bit address and 16-bit data multiplexed bus mode.

Figure 29. Siemens' 80C166 and PSD4XX/5XX Interface

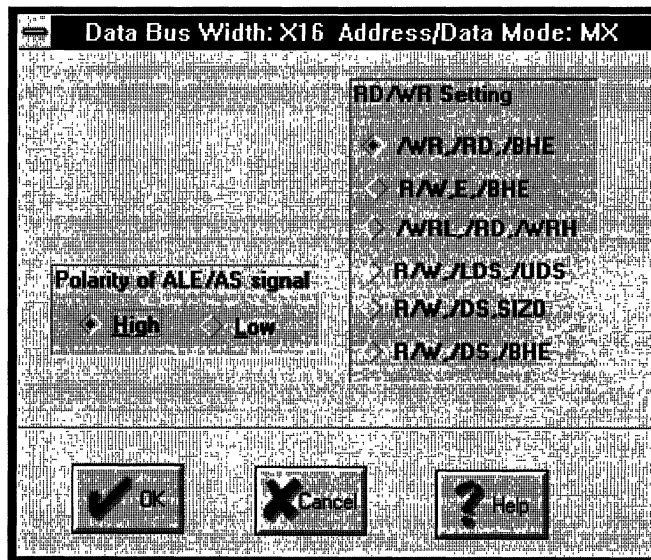
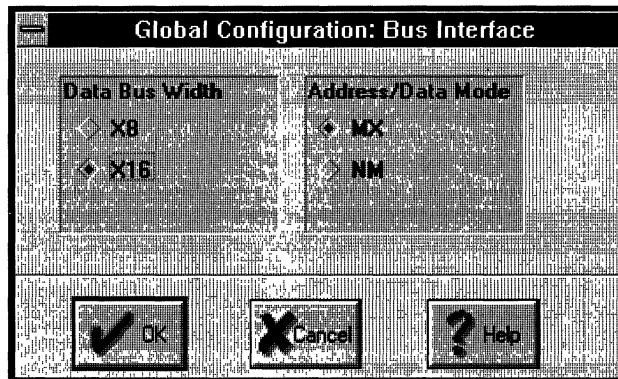


**Interfacing  
The  
PSD4XX/5XX  
To 80C166  
(Cont.)**

**Specify The 80C166 Bus Interface In PSDconfiguration**

As shown in the following windows which are captured from PSDconfiguration, the 80C166 bus interface can be specified by selecting:

- Data Bus Width: X16
- Address/Data Mode: MX
- ALE/AS signal: Yes
- Polarity of ALE: High
- RD/WR Setting:  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{BHE}$



**Interfacing  
The  
PSD4XX/5XX  
To 80C166  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL file**

The following is an example of defining the decoding function for the 80C166 application. The code is stored in two 16KB EPROM blocks and occupies address space 0000h to 7FFFh. The SRAM space is from 8000h to 87FFh. Table 17 illustrates the address map.

**Table 17. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> |      |
|----------------|---------------------|------|
| EPROM, Block 0 | 0000 – 3FFF         | Code |
| EPROM, Block 1 | 4000 – 7FFF         | Code |
| SRAM           | 8000 – 87FF         | Data |
| I/O Devices    | C000 – C0FF         | Data |

```

module 80c166
title 'example of 80c166 DPLD source file';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
reset pin ;
rst_out pin 34;

"Output signals

csiop, rs0, es0, es1 node ; "DPLD output chip selects

"DEFINITIONS

X = .x ; " Don't care
Address =
[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hC000) & (Address <= ^hC0FF) ; "Chip Select 256 block
rs0 = (Address >= ^h8000) & (Address <= ^h87FF) ; " SRAM, 2KB
es0 = (Address >= ^h0000) & (Address <= ^h3FFF) ; " EPROM 16KB
es1 = (Address >= ^h4000) & (Address <= ^h7FFF) ; " EPROM 16KB

"GPLD EQUATIONS

rst_out = reset;

END

```



## Interfacing The PSD4XX/5XX To 80C166 (Cont.)

### Simulation Of 80C166 Bus Cycle With The PSD4XX/5XX

Figure 30 shows the simulation of three 80C166 bus cycles. The first two cycles are byte write (55h) and read to SRAM location 8000h, the second is a code fetch cycle to EPROM location 0000h.

**Figure 30.**

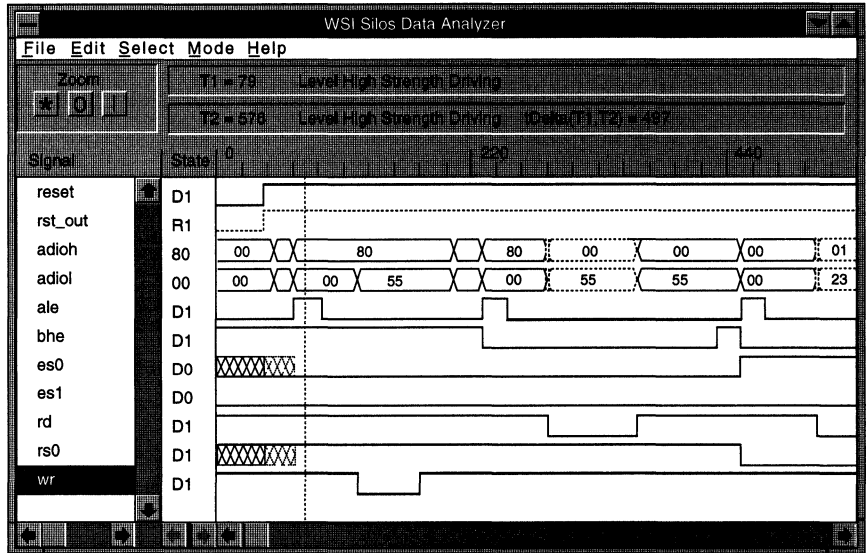


Figure 30 depicts a WORD read at location 0000hex when bhe=0 and A0=0.

## Interfacing The PSD4XX/5XX To Echelon NEURON® 3150™ Chip

### The 3150 Bus

The 3150 has an 8-bit non-multiplexed bus. The following signals are used to interface to memory or I/O devices:

- Address/Data:** A15-A0
- Data bus:** D7-D0
- Address Strobe:** None
- Control Signals:** R/W,  $\bar{E}$  (Enable Clock)

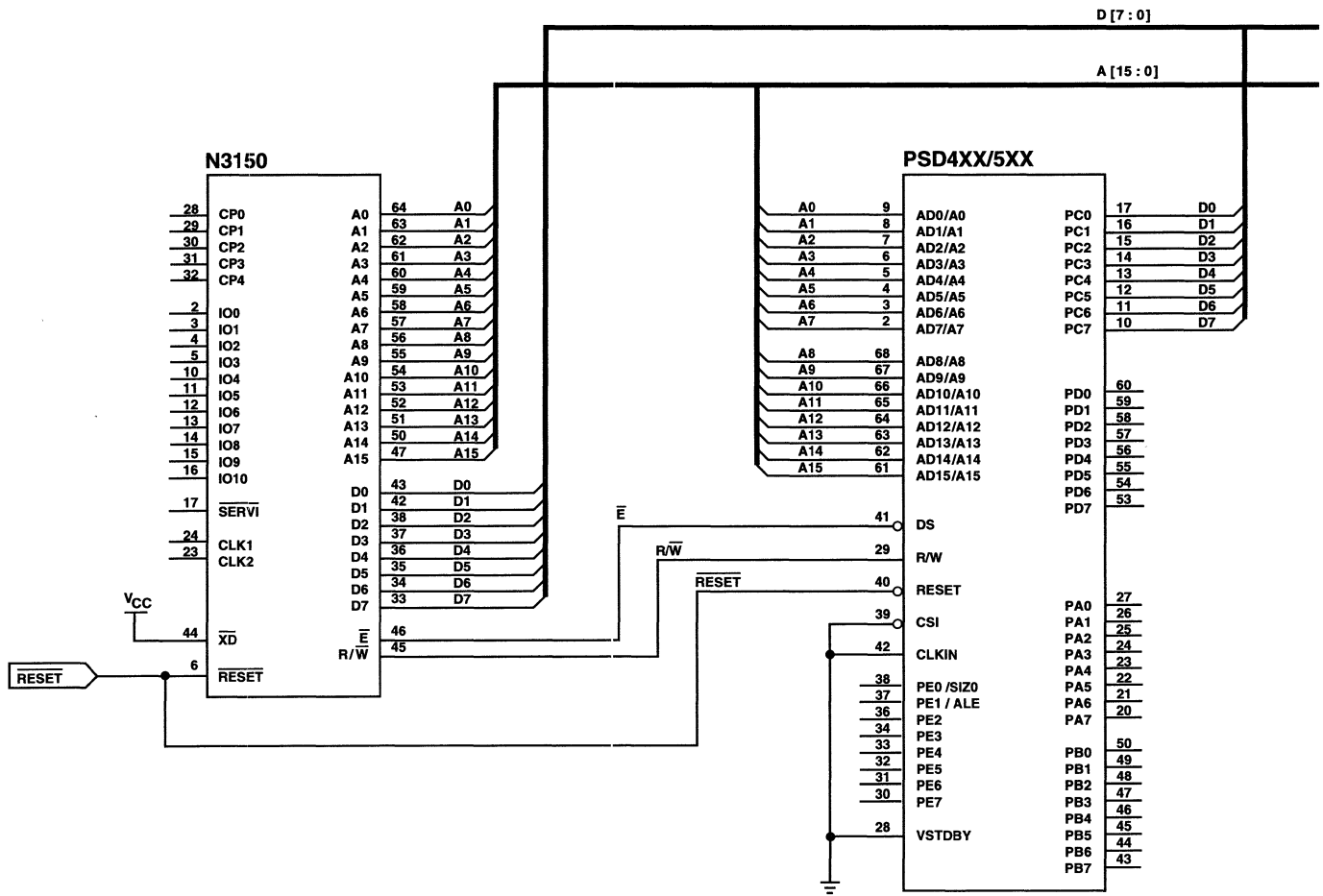
The 3150 has 64KB of program memory space. The E signal frequency is half that of the input clock. It is low during the second half of the bus cycle when read or write operation is taking place. A low R/W signal indicates it is a write bus cycle.

### The 3150 and PSD4XX/5XX Interface Schematic

Figure 31 shows the 3150 and PSD4XX/5XX interface schematic. The address lines A15-A0 are connected to the ADIO Port and the data lines D7-D0 are connected to Port C. Control signals R/W and  $\bar{E}$  are directly connected to the corresponding pins of the PSD4XX/5XX without any additional glue logic.



Figure 31. NEURON® 3150™ Chip and PSD4XX/5XX Interface

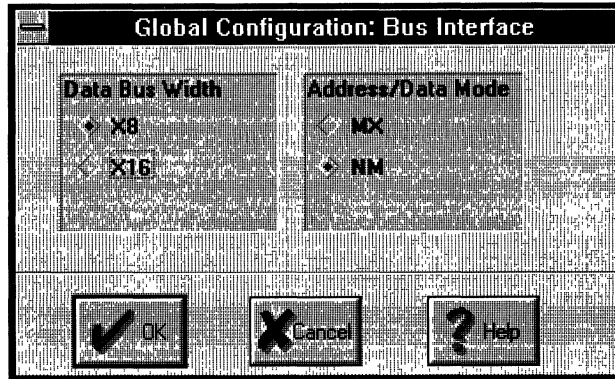


**Interfacing  
The  
PSD4XX/5XX  
To Echelon  
NEURON®  
3150™ Chip  
(Cont.)**

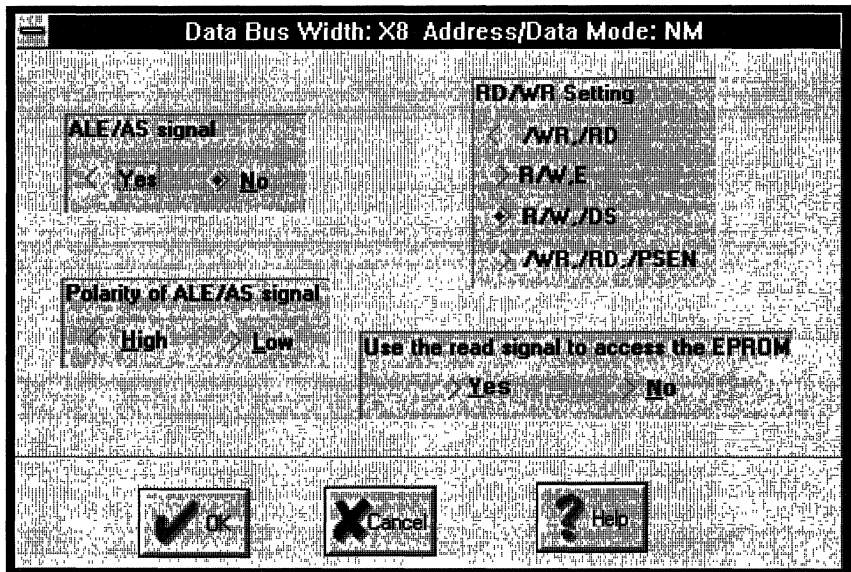
**Specify The 3150 Bus Interface In PSDconfiguration**

As shown in the following windows which are captured from PSDconfiguration, the 3150 bus interface can be specified by selecting:

- Data Bus Width: X8
- Address/Data Mode: NM
- ALE/AS signal: No
- RD/WR Setting:  $\overline{RD}$ ,  $\overline{DS}$ , (E acts as an active low data strobe signal)



3



**Interfacing  
The  
PSD4XX/5XX  
To Echelon  
NEURON®  
3150™ Chip  
(Cont.)**

**Define The DPLD/Decoding Function In The ABEL file**

The following is an example of defining the decoding function for the 3150 based application. The code is stored in three 16KB EPROM blocks and occupies address space 0000h to BFFFh. The SRAM space is from F000h to F7FFh. Table 18 illustrates the address map.

**Table 18. System Memory Map**

| <b>Device</b>  | <b>Memory Space</b> | <b>Memory Page</b> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 0000 – 3FFF         |                    |
| EPROM, Block 1 | 4000 – 7FFF         |                    |
| EPROM, Block 2 | 8000 – BFFF         |                    |
| SRAM           | C000 – C7FF         |                    |
| I/O Devices    | C800 – C8FF         |                    |

module 3150

title 'example of 3150 DPLD source file';

"Input signals

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

e pin 41; "ds in the configuration file has been aliased to e

"Output signals

csiop, rs0, es0, es1, es2 node ; "DPLD output chip selects

"DEFINITIONS

X = .x ; " Don't care

Address =

[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

"DPLD EQUATIONS

csiop = (Address >= ^hC800) & (Address <= ^hC8FF) ; " I/O Chip Select 256 bytes

rs0 = (Address >= ^hC000) & (Address <= ^hC7FF) ; " SRAM, 2KB

es0 = (Address >= ^h0000) & (Address <= ^h3FFF) ; " EPROM 16KB code

es1 = (Address >= ^h4000) & (Address <= ^h7FFF) ; " EPROM 16KB code

es2 = (Address >= ^h8000) & (Address <= ^hBFFF) ; " EPROM 16KB data

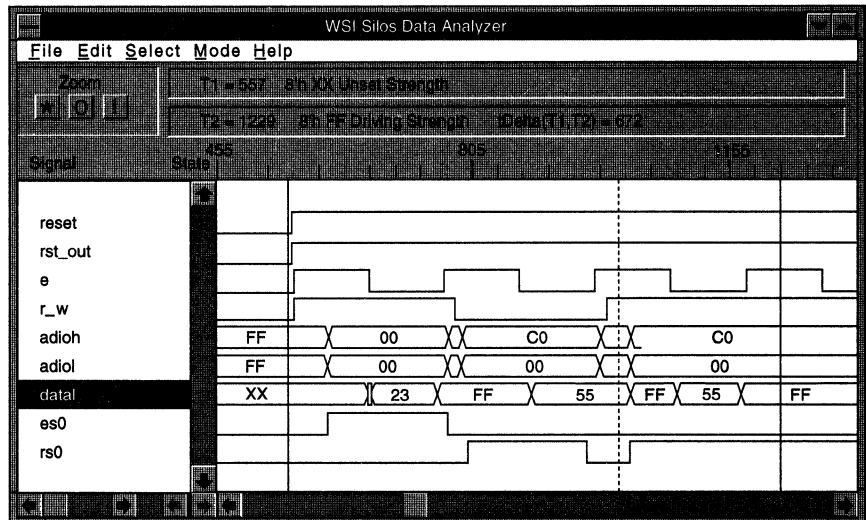
END

**Interfacing  
The  
PSD4XX/5XX  
To Echelon  
NEURON®  
3150™ Chip  
(Cont.)**

**Simulation Of The Echelon NEURON 3150 Bus Cycle With The PSD4XX/5XX**

Figure 32 shows the simulation of three 3150 bus cycles. The first cycle is a code fetch cycle to EPROM location 0000h and the following two cycles are write (55h) and read to SRAM location F000h.

**Figure 32.**



**3**

**Conclusion**

Using the PSD4XX/5XX with microcontrollers in embedded applications provides the following benefits over designs implemented with discrete components:

- Two chip solution (MCU & PSD) – smaller board size with fewer layers.
- ZPLD allows quick logic fixes and updates.
- Short development cycle
- Increase in system performance
- Reprogrammability.
- Lower power consumption
- Lower manufacturing cost
- Lower system cost.
- Security of design (security bit)
- Increase in system reliability.
- Reduced inventory cost.





# **Programmable Peripheral Application Note 030 PSD4XX and PSD5XX Power Calculations and Reduction**

---

## **Introduction**

The PSD4XX and PSD5XX families of programmable microcontroller peripherals integrate many functional blocks such as multiple ZPLD (Zero Power PLD) arrays, EPROM, SRAM, I/O Ports, Counter/Timers and an Interrupt Controller unit. The PSD family is being used extensively in microcontroller applications around the world by virtue of its flexibility and high level of integration, configurability and ease of use. This integration makes possible the design of very compact systems enabling the user to squeeze a great deal of functionality into a very small space. Thus, PSDs have found their way into small hand-held and battery operated applications such as cellular phones, medical instrumentation, and notebook computers that usually require, in addition to small space, very low power consumption. In many cases the PSDs are the lowest power design alternative possible!

The PSD4XX/5XX families are based on a patented high-performance CMOS technology and, like other CMOS devices, consume very little power even without the advanced power management features. However, the architecture of the family provides additional power management control via configuration bits, automatic power down circuitry, power switches and sleep mode making the PSD device even more valuable in power-sensitive applications.

This application note will describe the methods of optimizing and reducing the power consumption of the PSD device during system operation, standby and sleep mode. It makes sense to use these techniques even when low power is not a design requirement since they are easy to implement.

**3**

---

## **Power Use In The PSD4XX and PSD5XX**

The PSD4XX and PSD5XX contain several modules internally, each of which can be considered power consuming. These modules include the following:

- ZPLD (Zero Power PLD)
- EPROM
- SRAM
- I/O Ports
- Counter/Timer (only in PSD5XX)
- Interrupt Controller (only in PSD5XX)

The key to reducing the power used by the PSD4XX and PSD5XX is to reduce the power used by each individual module. There are three groups of power consuming functions that can work independently of each other and they are:

- ZPLD
- EPROM, SRAM and I/O Ports
- Counter/Timer and Interrupt Logic

For example, the ZPLD could be operating as a state machine while one of the MCU peripherals (EPROM, SRAM, I/O Ports) is being accessed by the MCU and the Counter/Timer is operating in the PWM mode. Obviously in this operation all modules operate and consume power. To derive the equations for power consumption it is necessary to understand the operating modes of each of the PSD modules and how to control them using the two Power Management Mode Registers (PMMR0 and PMMR1).

**Power Management Mode Registers**

The Power Management Mode Registers enable the user to have in-system control of the power consumption of each PSD module.

**PSD4XX Power Management Mode Register 0 (PMMR0)**

| Bit 7   | Bit 6     | Bit 5     | Bit 4      | Bit 3        | Bit 2      | Bit 1           | Bit 0   |
|---------|-----------|-----------|------------|--------------|------------|-----------------|---------|
| *       | ZPLD RCLK | ZPLD ACLK | ZPLD Turbo | EPROM CMiser | APD Enable | ALE PD Polarity | *       |
| 1 = OFF | 1 = OFF   | 1 = OFF   | 1 = OFF    | 1 = ON       | 1 = ON     | 1 = HIGH        | 1 = OFF |

**PSD5XX Power Management Mode Register 0 (PMMR0)**

| Bit 7   | Bit 6     | Bit 5     | Bit 4      | Bit 3        | Bit 2      | Bit 1           | Bit 0   |
|---------|-----------|-----------|------------|--------------|------------|-----------------|---------|
| TMR CLK | ZPLD RCLK | ZPLD ACLK | ZPLD Turbo | EPROM CMiser | APD Enable | ALE PD Polarity | *       |
| 1 = OFF | 1 = OFF   | 1 = OFF   | 1 = OFF    | 1 = ON       | 1 = ON     | 1 = HIGH        | 1 = OFF |

Bit 0 – \* = Should be set to High (1) to operate the APD.

Bit 1 – 0 = ALE Power Down (PD) Polarity Low.  
 1 = ALE Power Down (PD) Polarity High.

Bit 2 – 0 = Automatic Power Down (APD) Disable.  
 1 = Automatic Power Down (APD) Enable.

Bit 3 – 0 = EPROM/SRAM CMiser is OFF. (See EPROM/SRAM section for explanation)  
 1 = EPROM/SRAM CMiser is ON. (See EPROM/SRAM section for explanation)

Bit 4 – 0 = ZPLD Turbo is ON. ZPLD is always ON.  
 1 = ZPLD Turbo is OFF. ZPLD will Power Down when inputs are not changing.

Bit 5 – 0 = ZPLD Clock Input into the Array is connected. Every Clock change will Power Up the ZPLD when the Turbo bit is OFF.  
 1 = ZPLD Clock Input into the Array is disconnected.

Bit 6 – 0 = ZPLD Clock Input into the MacroCell registers is connected to the direct Clock input.  
 1 = ZPLD Clock Input into the MacroCell registers is disconnected from the direct Clock input.

Bit 7 – \* = In the PSD4XX should be set to High (1).  
 0 = In the PSD5XX Clock Input is connected to the Timer.  
 1 = In the PSD5XX Clock Input is disconnected from the Timer.

## Power Management Mode Registers (Cont.)

### PSD4XX/5XX Power Management Mode Register 1 (PMMR1)

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1      | Bit 0     |
|-------|-------|-------|-------|-------|-------|------------|-----------|
| *     | *     | *     | *     | *     | *     | Sleep Mode | APD CLK   |
|       |       |       |       |       |       | 1 = ON     | 1 = CLKIN |

NOTE: \* = Reserved for future use, should be set to zero.

Bit 0 – 0 = Automatic Power Down Unit Clock is connected to Port E7 (PE7) alternate function input.

1 = Automatic Power Down Unit Clock is connected to the PSD Clock input.

Bit 1 – 0 = Sleep Mode Disabled.

1 = Sleep Mode Enabled.

## ZPLD

The Zero Power PLD (ZPLD) has two modes of operation:

Non-Turbo

Turbo

These modes are in-system user programmable on the fly by configuring the ZPLD Turbo bit (bit 4) in the PMMR0. The difference between the two modes is shown in Figure 1 for PSD5XXB1 (the PSD4XXA1 and PSD4XXA2 power figures are in their respective data sheets). When the ZPLD Turbo bit is OFF (Logic 1), the ZPLD will be in power down if no inputs are changing for a time of 66 ns. When one or more inputs change the ZPLD automatically powers up and generates and latches the new outputs. It will retain the output values as long as the inputs do not change. This is also true in power down and sleep mode. The inputs that cause the ZPLD to power up are described in Table 1. It is important to note that those inputs affect the ZPLD only when they are used as inputs to the ZPLD.

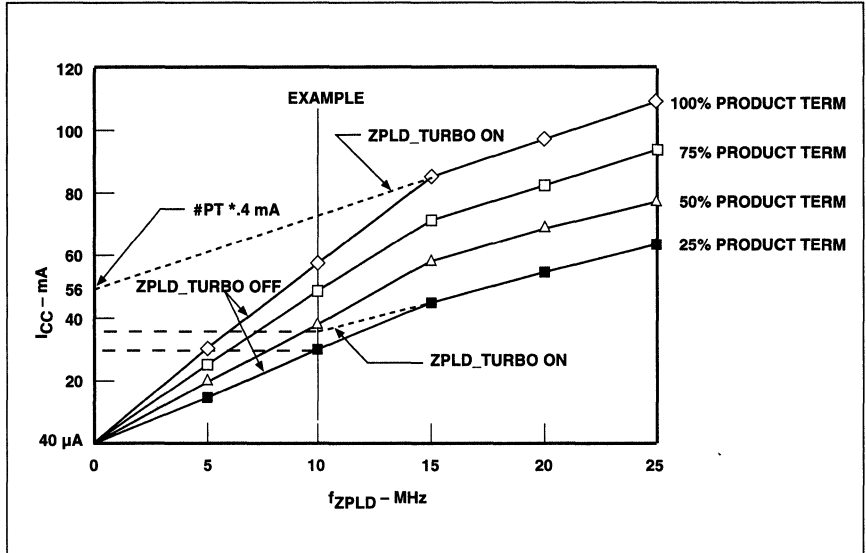
In the non-turbo mode there is an additional delay of 10 ns for some timing parameters ( $t_{PD}$ ,  $t_{RPD}$ ,  $t_{EA}$ ,  $t_{ER}$ ,  $t_{ARP}$ ,  $t_S$ ,  $t_{SA}$ ,  $t_{COA}$ ). It is important to note that if inputs are changing at a higher frequency than 15 MHz there is no need to add 10 ns to those timing parameters. Above 15 MHz the ZPLD will stay powered up all the time independent of the mode of operation.

The power down specification for the ZPLD is shown in Table 2 under the ZPLD only section (40  $\mu$ A). As the frequency of the inputs to the ZPLD increases, the ICC drain also increases. For the same frequency the power consumption is proportional to the percentage of product terms used in the ZPLD in that application. For example, if an application uses a PSD5XX that has a 140 product term ZPLD but only 35 product terms participate in generating the user defined equations (the other 105 product terms are automatically turned off by the PSDsoft) then the 25% (35/140) product term graph should be used to calculate the ICC consumption. At 10 MHz the ICC equals 29 mA.



**ZPLD**  
(Cont.)

**Figure 1. PSD5XXB1 ZPLD  $I_{CC}$  vs. Frequency Consumption**



Equations Representation of the ZPLD Power Graphs:

1. If ZPLD\_TURBO Bit = OFF and  $f_{ZPLD} \leq 15$  MHz then

$$I_{CC_{ZPLD}} = \left( \frac{2 \text{ (mA/MHz)} * 15 \text{ (MHz)} + \# \text{ PT} * 0.4 \text{ (mA/PT)}}{15} \right) * f_{ZPLD}$$

2. If ZPLD\_TURBO Bit = ON or  $f_{ZPLD} > 15$  MHz then

$$I_{CC_{ZPLD}} = 2 \text{ (mA/MHz)} * f_{ZPLD} + \# \text{ PT} * 0.4 \text{ (mA/PT)}$$

If the ZPLD Turbo bit is ON (Logic 0) the ZPLD will not enter standby mode and it will always consume power even when inputs are not changing. In this mode the ICC power usage of the ZPLD is also based on the percent of product terms being used in the application. At 10 MHz, for the same example above, the ICC equals 34 mA.

Above 15 MHz both modes operate with the same ICC power curves. The reason is that a non-Turbo ZPLD at frequencies below 15 MHz is capable of powering down before the next input changes. For customers that use the ZPLD at frequencies above 15 MHz but still have modes that require powering down the PSD4XX/5XX, the non-turbo mode should be used. If Sleep mode is enabled (see Sleep Mode Section) and executed, the ZPLD will enter into Sleep overriding the condition of the ZPLD Turbo bit. The ZPLD will return to the previous mode of operation when exiting Sleep Mode.

**ZPLD**  
(Cont.)**Table 1. PSD4XX/5XX ZPLD Inputs**

| <b>Function Name</b>    | <b>ZPLD Input Condition</b>                                                                                                                                                                               |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A8 – AD15               | Always ZPLD Input                                                                                                                                                                                         |
| A0 – A1                 | Always ZPLD Input                                                                                                                                                                                         |
| $\overline{RD}$         | Always ZPLD Input                                                                                                                                                                                         |
| $\overline{WR}$         | Always ZPLD Input                                                                                                                                                                                         |
| $\overline{CSi}$        | Always ZPLD Input                                                                                                                                                                                         |
| $\overline{RESET}$      | Always ZPLD Input                                                                                                                                                                                         |
| CLKIN                   | Upon reset CLKIN is an input to the ZPLD array. CLKIN can be masked from the ZPLD array by the user, if it is not used as part of logic equation or to reduce power in the system by setting PMMR0 bit 5. |
| PA0 – PA7               | Only when used as ZPLD inputs                                                                                                                                                                             |
| PB0 – PB7               | Only when used as ZPLD inputs                                                                                                                                                                             |
| PC0 – PC7               | Only when used as ZPLD inputs<br>(Not available in PSD4XXA1)                                                                                                                                              |
| PD0 – PD7               | Only when used as ZPLD inputs<br>(Not available in PSD4XXA1)                                                                                                                                              |
| PE0 – PE7               | Only when used as ZPLD inputs or alternate functions<br>(BHE, PSEN, WRH, UDS, SIZE, ALE, APD CLK)                                                                                                         |
| PGR0 – PGR3             | Always ZPLD Input                                                                                                                                                                                         |
| INT2PLD                 | Always ZPLD Input (Only in PSD5XX)                                                                                                                                                                        |
| WDOG2PLD                | Always ZPLD Input (Only in PSD5XX)                                                                                                                                                                        |
| ZPLD MacroCell Feedback | If ZPLD MacroCell is used as a buried feedback<br>(Combinatorial or Registered) accounts as a ZPLD input.                                                                                                 |

3

**EPROM**

The EPROM power consumption in the PSD is controlled by bit 3 in the PMMR0 – EPROM CMiser. Upon reset the CMiser bit is OFF. This will cause the EPROM to be ON at all times as long as  $\overline{CSi}$  is enabled (low). The reason this mode is provided is to reduce the access time of the EPROM by 10 ns relative to the low power condition when CMiser is ON. If  $\overline{CSi}$  is disabled (high) the EPROM will be deselected and will enter standby mode (OFF) overriding the state of the CMiser.

If CMiser is set (ON), the EPROM will enter the standby mode when not selected. This condition can take place when  $\overline{CSi}$  is high or when  $\overline{CSi}$  is low and the EPROM is not accessed. For example, if the MCU is accessing the SRAM, the EPROM will be deselected and will be in low power mode.

An additional advantage of the CMiser is achieved when the PSD is configured in the by 8 mode (8-bit data bus). In this case an additional power savings is achieved in the EPROM (and also in the SRAM) by turning off 1/2 of the array even when the EPROM is accessed (the array is divided internally into odd and even arrays).

Power consumption for the different EPROM modes is given in Table 2 under ICC (DC) EPROM Adder.

## SRAM

The SRAM in the PSD will always be in the Standby mode when not selected.

An additional advantage of the CMiser is when the PSD is configured in the by 8 mode (8-bit data bus). In this case an additional power saving is achieved in the SRAM by turning off 1/2 of the array even when the SRAM is accessed (the array is divided internally to odd and even arrays).

The SRAM also has a dedicated supply voltage  $V_{SBY}$  that can be used to connect a battery. When  $V_{CC}$  becomes lower than  $V_{SBY} - 0.6$ , the PSD will automatically connect the  $V_{SBY}$  as a power source to the SRAM. The SRAM Standby Current ( $I_{SBY}$ ) is typically 0.5  $\mu$ A.

The SRAM data retention voltage  $V_{DF}$  is 2 V minimum.

---

## Standby Modes

There are two standby modes in the PSD4XX/5XX:

- Power Down**
- Sleep**

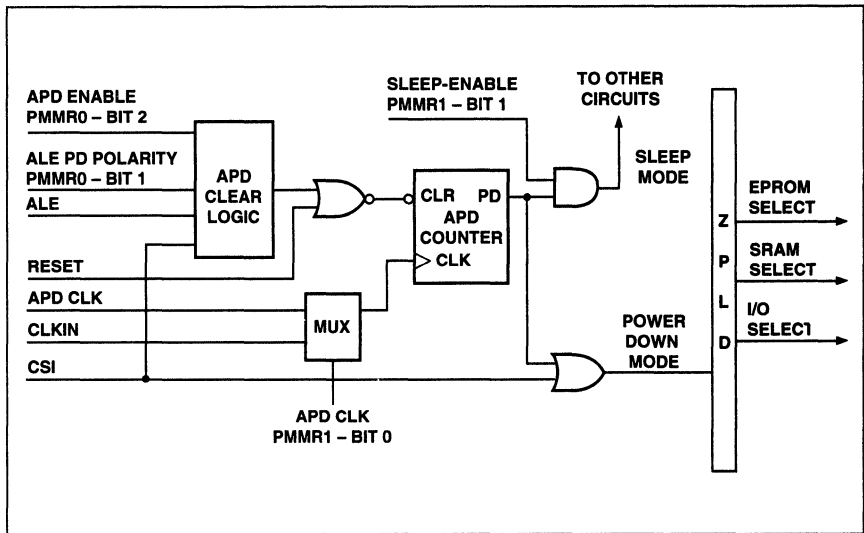
### Power Down Mode

Power Down mode causes all the memory blocks (EPROM, SRAM) that are connected to the MCU to enter their low power modes. Traditionally the power down mode is controlled by the  $\overline{CS}$  pin on peripherals and memories. In addition to the  $\overline{CS}$  pin causing power down in the PSD4XX/5XX, there is also an Automatic Power Down Unit (APD) which will be described later. When  $\overline{CS}$  is high or APD reaches an overflow condition the EPROM and SRAM will power down. In addition, all the MCU interface buffers will be disabled to reduce power consumption. The MCU interface includes signals  $\overline{ADIO0} - 15$ ,  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{ALE}$  (disabled only by  $\overline{CS}$  and not by APD),  $\overline{PSEN}$ ,  $\overline{UDS}$ ,  $\overline{LDS}$  and other alternate functions. The PSD4XX/5XX non-memory internal blocks such as Counter/Timers, Interrupt Controller, I/O Ports and ZPLD (in non-turbo mode) continue to function independently of the power down mode. It is important however to note that if no inputs are changing these peripherals do not consume any power.

The PSD4XX/5XX also includes an APD unit that enables the user to enter a power down mode independent of controlling the  $\overline{CS}$  input. This feature eliminates the need for external logic (decoders and latches) to power down the PSD. The APD unit concept is based on tracking the activity on the  $\overline{ALE}$  pin. If the APD unit is enabled and  $\overline{ALE}$  is not active, the 4-bit APD counter starts counting and will overflow after 15 clocks, generating a PD signal powering down the PSD. If sleep mode is enabled, then the PD signal will also activate the sleep mode. Immediately after  $\overline{ALE}$  starts pulsing the PSD will exit the power down or sleep mode.

## Standby Modes (Cont.)

**Figure 2. PSD4XX/5XX Automatic Power Down Unit (APD) Block Diagram**



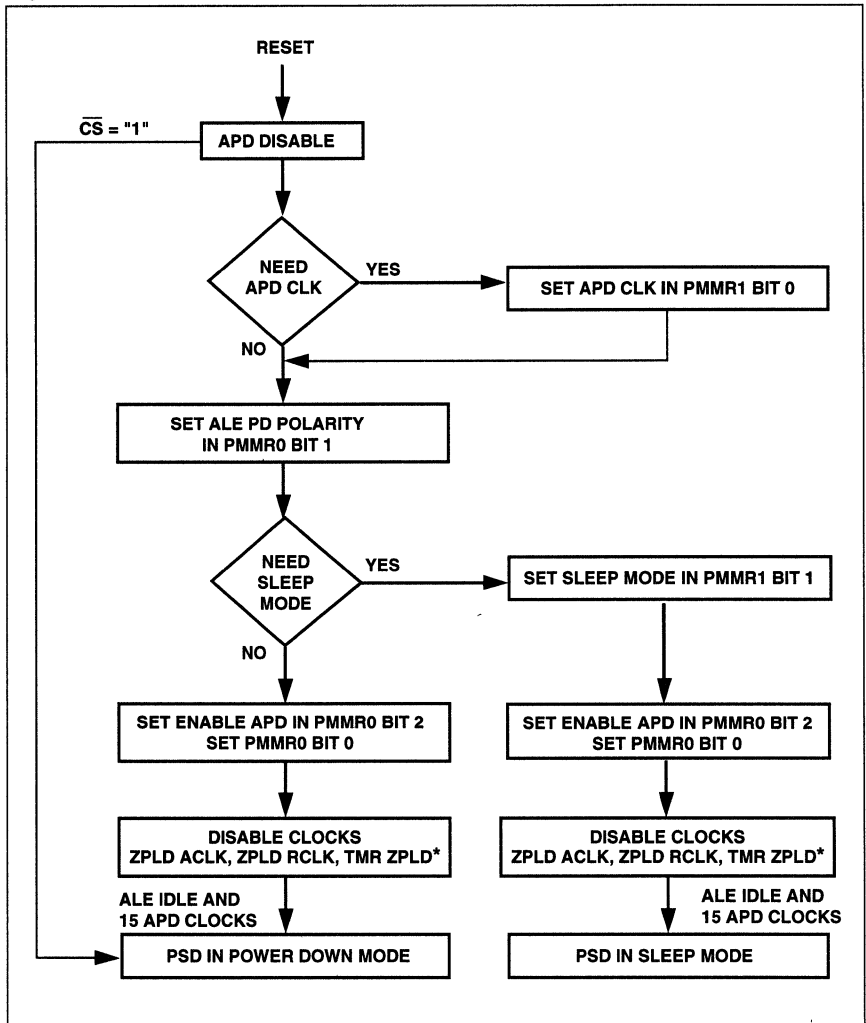
### Power Down Mode (Cont.)

The operation of the APD is controlled by the PMMR (see Figure 3). PMMR1 bit 0 selects the source of the APD counter clock. After reset the APD counter clock is connected to the CLKIN pin of the PSD. In order to guarantee that the APD will not overflow there should be less than 15 APD clocks between two ALE pulses. If the CLKIN frequency is not adequate, then a different clock can be connected to PE7 which is used as an Alternate Function In – APD CLK.

The next step is to select the ALE power down polarity. Usually, MCUs entering power down will freeze their ALE at logic high or low. By programming bit 1 of PMMR0 the power down polarity can be defined for the APD. If the APD detects that the ALE is in the power down polarity for 15 APD counter clocks, the PSD will enter a power down mode. To enable the APD, operation bit 2 in the PMMR0 should be set high.

**Standby Modes**  
(Cont.)

**Figure 3. Automatic Power Down Unit (APD) Flow Chart**



\*TMR ZPLD is only on PSD5XX.

## Standby Modes (Cont.)

### Sleep Mode

Sleep Mode provides capability to reduce the power consumption of the PSD4XX/5XX to 5  $\mu$ A. In addition to the Power Down mode state, in Sleep Mode also all reference voltages are turned off.

The Sleep Mode is enabled by executing the same operations required for automatic power down. In addition PMMR1 bit 1 (Sleep Mode) should be set to high. When the APD counter overflows the PSD will enter Sleep Mode.

Two conditions can cause the PSD to exit the Sleep Mode: either the ALE starts pulsing or the  $\overline{\text{CSI}}$  pin changes its state from high to low. The PSD access time from Sleep Mode is specified by the  $t_{\text{LVDV1}}$  parameter. In the Sleep Mode the ZPLD still monitors the inputs and responds to them with a delay time of  $t_{\text{LVDV2}}$ . See Table 2 for a summary of timing during Power Down and Sleep Mode.

## Input Clock

The PSD4XX/5XX clock input (CLKIN) is used as a source for driving the following modules:

- ZPLD Array Clock Input
- ZPLD MacroCell Clock Flip-Flop
- APD Counter Clock
- Counter/Timers Clock

During power down or if any of the modules are not being used the clock to these modules should be disabled. To reduce AC power consumption, it is especially important to disable the clock input to the ZPLD array if it is not used as part of a logic equation.

The ZPLD Array Clock can be disabled by setting PMMR0 bit 5 (ZPLD ACLK). The ZPLD MacroCell Clock Input can be disabled by setting PMMR0 bit 6 (ZPLD RCLK). The Timer Clock can be disabled by setting PMMR0 bit 7 (TMR CLK). The APD Counter Clock will be disabled automatically if Power Down or Sleep Mode is entered through the APD unit. The input buffer of the CLKIN input will be disabled if bits 5 – 7 PMMR0 are set and the APD has overflowed.

The Counter/Timers can operate in Sleep Mode if the TMR CLK bit is low, but the power consumption will be based on the frequency of operation (CLKIN frequency).

3

**Table 2. Summary of PSD4XX/5XX Timing and Standby Current During Power Down and Sleep Modes**

|            | <b>PLD Propagation Delay</b>       | <b>PLD Recovery Time To Normal Operation</b> | <b>Access Time</b> | <b>Access Recovery Time To Normal Access</b> | <b>Typical Standby Current Consumed</b> |
|------------|------------------------------------|----------------------------------------------|--------------------|----------------------------------------------|-----------------------------------------|
| Power Down | Normal $t_{\text{PD}}$<br>(Note 1) | 0                                            | No Access          | $t_{\text{LVDV}}$                            | 40 $\mu$ A<br>(Note 4)                  |
| Sleep      | $t_{\text{LVDV2}}$<br>(Note 2)     | $t_{\text{LVDV3}}$<br>(Note 3)               | No Access          | $t_{\text{LVDV1}}$                           | 5 $\mu$ A<br>(Note 5)                   |

- NOTES**
1. Power Down does not affect the operation of the ZPLD. The ZPLD operation in this mode is based only on the ZPLD\_Turbo Bit.
  2. In Sleep Mode any input to the ZPLD will have a propagation delay of  $t_{\text{LVDV2}}$ .
  3. PLD recovery time to normal operation after exiting Sleep Mode. An input to the ZPLD during the transition will have a propagation delay time of  $t_{\text{LVDV3}}$ .
  4. Typical current consumption assuming all clocks are disabled and ZPLD is in non-turbo mode.
  5. Typical current consumption assuming all clocks are disabled.

## PSD4XX/5XX Power Consumption Equations

To calculate the PSD4XX/5XX power consumption the following assumptions are made:

- In Sleep Mode none of the internal blocks are operating.
- The ZPLD can operate at times when the MCU is idle. Operating frequency of the ZPLD is based on the highest frequency input signal connected to it.
- The total power consumption is based on the percentage of PSD operation in each mode of operation.

The PSD4XX/5XX power consumption equation is given by the following:

$$[1] \text{ ICC}_{\text{TOTAL}} = \text{ISB}_{\text{SLEEP}} + \text{ICC}_{\text{ZPLD}} + \text{ICC}_{\text{MCU\_ACCESS}} + \text{ICC}_{\text{TIMER}}$$

$$[2] \text{ ICC}_{\text{MCU\_ACCESS}} = \text{ISB}_{\text{PD}} + \text{ICC}_{\text{EPROM}} + \text{ICC}_{\text{SRAM}} + \text{ICC}_{\text{OTHER}}$$

### NOTATION:

- $\text{ISB}_{\text{SLEEP}}$  – Standby current in sleep mode
- $\text{ICC}_{\text{ZPLD}}$  – ICC current when ZPLD is operating
- $\text{ICC}_{\text{MCU\_ACCESS}}$  – ICC current when MCU is accessing the PSD
- $\text{ICC}_{\text{TIMER}}$  – ICC current when Timer is operating. This current is only AC and is based on the CLOCK in frequency.
- $\text{ISB}$  – Standby current in power down mode
- $\text{ICC}_{\text{EPROM}}$  – ICC current when EPROM is operating
- $\text{ICC}_{\text{SRAM}}$  – ICC current when SRAM is operating
- $\text{ICC}_{\text{OTHER}}$  – ICC current when other peripherals are being accessed such as the internal Counter/Timers, I/O Ports or external peripherals to the PSD4XX/5XX. In this case only the ZPLD is used and the power is calculated based on the ZPLD Only section in Table 3.

Equation [1] describes the total ICC consumed by the PSD in the system while equation [2] is the current consumed by the PSD blocks that are accessed by the MCU. The sum of the currents is proportional to the time that the PSD is in each mode.

Table 3 includes the power specifications required to calculate the PSD4XX or PSD5XX power consumption (see data sheet for most recent ICC values). All parameters are specified for  $V_{\text{CC}} = 5\text{V} \pm 10\%$ . The standby current (ISB) is specified with the assumption that all internal blocks are idle. The ICC (DC) is specified for 3 blocks: ZPLD, EPROM and SRAM. If the ZPLD is active (all other modules idle) and ZPLD\_Turbo bit is OFF then the PSD will be in one of the standby modes (based on the PMMR configuration). If ZPLD\_Turbo is on, the DC power consumption has to be calculated based on the number of product terms used. When the EPROM or SRAM are accessed, power is added to the power consumed by the ZPLD. The AC parameters are also specified and should be added based on the percentage of activity of each module.

**Table 3. PSD4XX/5XX AC/DC Power Consumption Parameters**

| <b>Symbol</b>         | <b>Parameter</b>         |                                 | <b>Conditions</b>                         | <b>Min</b> | <b>Typ</b> | <b>Max</b> | <b>Units</b>     |
|-----------------------|--------------------------|---------------------------------|-------------------------------------------|------------|------------|------------|------------------|
| $V_{CC}$              | Operating Supply Voltage |                                 |                                           | 4.5        | 5          | 5.5        | V                |
| $I_{SB}$              | Standby Supply Current   | Power Down Mode                 | $\overline{CS1} > V_{CC} - 0.3 \text{ V}$ |            | 40         | 100        | $\mu\text{A}$    |
|                       |                          | Sleep Mode                      | $\overline{CS1} > V_{CC} - 0.3 \text{ V}$ |            | 5          | 10         | $\mu\text{A}$    |
| $I_{CC} \text{ (DC)}$ | Operating Supply Current | ZPLD Only                       | ZPLD_TURBO = OFF, f = 0 MHz               |            | 40         |            | $\mu\text{A}$    |
|                       |                          |                                 | ZPLD_TURBO = ON, f = 0 MHz                |            | 400        | 700        | $\mu\text{A/PT}$ |
|                       |                          | EPROM Adder                     | CMiser = ON and Not Selected              |            | 0          |            | $\mu\text{A}$    |
|                       |                          |                                 | CMiser = ON and Selected (x8 Data Mode)   |            | 10         | 15         | mA               |
|                       |                          |                                 | CMiser = ON and Selected (x16 Data Mode)  |            | 15         | 20         | mA               |
|                       |                          |                                 | CMiser = OFF Selected or Not Selected     |            | 15         | 20         | mA               |
|                       |                          | SRAM Adder                      | SRAM Not Selected                         |            | 0          |            | $\mu\text{A}$    |
|                       |                          |                                 | CMiser = ON and Selected (x8 Data Mode)   |            | 25         | 40         | mA               |
|                       |                          |                                 | CMiser = ON and Selected (x16 Data Mode)  |            | 30         | 45         | mA               |
| $I_{CC} \text{ (AC)}$ | ZPLD                     | ZPLD_TURBO = OFF (See Figure 1) |                                           |            |            |            |                  |
|                       |                          | ZPLD_TURBO = ON                 |                                           |            | 2          |            | mA/MHz           |
|                       | EPROM or SRAM            |                                 |                                           |            | 2          |            | mA/MHz           |
|                       | TIMER                    |                                 |                                           |            | 1          |            | mA/MHz           |

NOTE: See data sheet for the most recent ICC values.



### Examples

Here are three examples of power calculations for an application that has a high percentage of time in Sleep mode. It is important to note the measured ICC in the system could be lower because the parameters provided in Table 3 are conservative. Following is the PSD5XX configuration in the system used to calculate those examples:

|                              |   |                              |
|------------------------------|---|------------------------------|
| Data Bus Width               | = | 8-Bit                        |
| PSD MCU Access Frequency     | = | 2 MHz (ALE Frequency)        |
| % of Time in Sleep Mode      | = | 90%                          |
| % of Time MCU Access the PSD | = | 10%                          |
| % of MCU Access in PD        | = | 30%                          |
| % of MCU Access to EPROM     | = | 50%                          |
| % of MCU Access to SRAM      | = | 10%                          |
| % of MCU Access to Other     | = | 10%                          |
| % of Time ZPLD Operating     | = | 10%                          |
| ZPLD Operational Frequency   | = | 12 MHz                       |
| ZPLD Product Terms Active    | = | 40 = $(40/140) * 100 = 28\%$ |

Table 4 shows calculation of the power consumption assuming CMiser = ON and ZPLD\_Turbo = OFF.

Table 5 shows calculation of the power consumption assuming CMiser = OFF and ZPLD\_Turbo = OFF.

Table 6 shows calculation of the power consumption assuming CMiser = ON and ZPLD\_Turbo = ON.

**Table 4. CMiser = ON, ZPLD\_Turbo = OFF**

| Mode         | %   | I <sub>CC</sub> (DC)<br>D/S | I <sub>CC</sub> (DC)<br>Equations | I <sub>CC</sub> (DC)<br>Total | I <sub>CC</sub> (AC)<br>D/S | I <sub>CC</sub> (AC)<br>Equations | I <sub>CC</sub> (AC)<br>Total | I <sub>CC</sub><br>(DC + AC) |
|--------------|-----|-----------------------------|-----------------------------------|-------------------------------|-----------------------------|-----------------------------------|-------------------------------|------------------------------|
| Sleep        | 90% | ISB (Sleep)<br>5 μA         | .9*5 μA                           | 4.5 μA                        | 0                           | 0                                 | 0                             | 4.5 μA                       |
| MCU_Access   | 10% |                             |                                   |                               |                             |                                   |                               |                              |
| PD           | 30% | 40 μA                       | .1*.3*40 μA                       | 1.2 μA                        | 0                           | 0                                 | 0                             | 1.2 μA                       |
| EPROM        | 50% | 10 mA                       | .1*.5*10 mA                       | .5 mA                         | 2 mA/MHz                    | .1*.5*2 mA/MHz*2 MHz              | .2 mA                         | 0.7 mA                       |
| SRAM         | 10% | 25 mA                       | .1*.1*25 mA                       | .25 mA                        | 2 mA/MHz                    | .1*.1*2 mA/MHz*2 MHz              | .04 mA                        | 0.39 mA                      |
| Other        | 10% | 0                           | 0                                 |                               | 1 mA/MHz                    | .1*.1*1 mA/MHz*2 MHz              | .02 mA                        | 0.12 mA                      |
| ZPLD         | 10% | 0                           | 0                                 | 0                             | (Figure 1)<br>36 mA         | .1*36 mA                          | 3.6 mA                        | 3.6 mA                       |
| <b>TOTAL</b> |     |                             |                                   | <b>.75 mA</b>                 |                             |                                   | <b>3.86 mA</b>                | <b>4.63 mA</b>               |

The ZPLD does not consume DC power and the EPROM consumes power only when selected.



**Table 5. CMiser = OFF, ZPLD\_Turbo = OFF**

| <b>Mode</b>  | <b>%</b> | <b><math>I_{CC}</math> (DC)<br/>D/S</b> | <b><math>I_{CC}</math> (DC)<br/>Equations</b> | <b><math>I_{CC}</math> (DC)<br/>Total</b> | <b><math>I_{CC}</math> (AC)<br/>D/S</b> | <b><math>I_{CC}</math> (AC)<br/>Equations</b> | <b><math>I_{CC}</math> (AC)<br/>Total</b> | <b><math>I_{CC}</math><br/>(DC + AC)</b> |
|--------------|----------|-----------------------------------------|-----------------------------------------------|-------------------------------------------|-----------------------------------------|-----------------------------------------------|-------------------------------------------|------------------------------------------|
| Sleep        | 90%      | ISB (Sleep)<br>5 $\mu$ A                | .9*5 $\mu$ A                                  | 4.5 $\mu$ A                               | 0                                       | 0                                             | 0                                         | 4.5 $\mu$ A                              |
| MCU_Access   | 10%      |                                         |                                               |                                           |                                         |                                               |                                           |                                          |
| PD           | 30%      | 40 $\mu$ A                              | .1*.3*40 $\mu$ A                              | 1.2 $\mu$ A                               | 0                                       | 0                                             | 0                                         | 1.2 $\mu$ A                              |
| EPROM        | 50%      | 15 mA                                   | .1*.5*15 mA                                   | .75 mA                                    | 2 mA/MHz                                | .1*.5*2 mA/MHz*2 MHz                          | .2 mA                                     | 0.95 mA                                  |
| SRAM         | 10%      | 45 mA                                   | .1*.1*45 mA                                   | .45 mA                                    | 2 mA/MHz                                | .1*.1*2 mA/MHz*2 MHz                          | .04 mA                                    | 0.49 mA                                  |
| Other        | 10%      | 15 mA                                   | .1*.1*15 mA                                   | .15 mA                                    | 1 mA/MHz                                | .1*.1*1 mA/MHz*2 MHz                          | .02 mA                                    | 0.17 mA                                  |
| ZPLD         | 10%      | 0                                       | 0                                             | 0                                         | (Figure 1)<br>36 mA                     | .1 * 36 mA                                    | 3.6 mA                                    | 3.6 mA                                   |
| <b>TOTAL</b> |          |                                         |                                               | <b>1.35 mA</b>                            |                                         |                                               | <b>3.86 mA</b>                            | <b>5.21 mA</b>                           |

The ZPLD does not consume DC power. The EPROM is on when the PSD is not in PD and the by 8 configuration does not provide the advantage of turning off 1/2 of the arrays in the SRAM and EPROM. This is the reason that the EPROM, SRAM and other power consumption increases.

**Table 6. CMiser = OFF, ZPLD\_Turbo = ON**

| <b>Mode</b>  | <b>%</b> | <b>I<sub>CC</sub> (DC)<br/>D/S</b> | <b>I<sub>CC</sub> (DC)<br/>Equations</b> | <b>I<sub>CC</sub> (DC)<br/>Total</b> | <b>I<sub>CC</sub> (AC)<br/>D/S</b> | <b>I<sub>CC</sub> (AC)<br/>Equations</b> | <b>I<sub>CC</sub> (AC)<br/>Total</b> | <b>I<sub>CC</sub><br/>(DC + AC)</b> |
|--------------|----------|------------------------------------|------------------------------------------|--------------------------------------|------------------------------------|------------------------------------------|--------------------------------------|-------------------------------------|
| Sleep        | 90%      | ISB (Sleep)<br>5 μA                | .9*5 μA                                  | 4.5 μA                               | 0                                  | 0                                        | 0                                    | 4.5 μA                              |
| MCU_Access   | 10%      |                                    |                                          |                                      |                                    |                                          |                                      |                                     |
| PD           | 30%      | 40 μA                              | .1*.3*40 μA                              | 1.2 μA                               | 0                                  | 0                                        | 0                                    | 1.2 μA                              |
| EPROM        | 50%      | 15 mA                              | .1*.5*15 mA                              | .75 mA                               | 2 mA/MHz                           | .1*.5*2 mA/MHz*2 MHz                     | .2 mA                                | 0.95 mA                             |
| SRAM         | 10%      | 45 mA                              | .1*.1*45 mA                              | .45 mA                               | 2 mA/MHz                           | .1*.1*2 mA/MHz*2 MHz                     | .04 mA                               | 0.49 mA                             |
| Other        | 10%      | 15 mA                              | .1*.1*15 mA                              | .15 mA                               | 1 mA/MHz                           | .1*.1*1 mA/MHz*2 MHz                     | .02 mA                               | 0.17 mA                             |
| ZPLD         | 10%      | .4 mA/PT                           | .1*.4*40 PT                              | 1.6 mA                               | (Figure 1)<br>40 mA                | .1 * 40 mA                               | 4 mA                                 | 5.6 mA                              |
| <b>TOTAL</b> |          |                                    |                                          | <b>2.95 mA</b>                       |                                    |                                          | <b>4.26 mA</b>                       | <b>7.21 mA</b>                      |

The ZPLD is on 10% of the time also when inputs are not changing.

**Conclusion**

The PSD4XX/5XX provides an extremely low power programmable solution to any system. The capability of the user to configure the power consumption in-system using the power management registers enables speed/power optimization. If the designer uses all of the power saving features in the PSD, the result will be a lower power consumption than that of any alternative.



# Programmable Peripheral Application Note 031 PSD4XX/5XX Design Tutorial

## Introduction

This tutorial takes you step by step through the development cycle of a PSD4XX/5XX based design, from design entry to programming the device. A simple design example is used to demonstrate how some of the key functions in the PSD4XX/5XX are utilized.

At the end of this chapter, the following materials are included:

- Files generated during the design cycle
- Files generated for applications with various bus types
- Chip architecture overview

The following information is covered in the tutorial:

- Design Example
- PSDsoft Development Tools
- Using the Design Example

## Design Example

A typical yet simple design is used as an example to illustrate the development cycle. This design example can be a part of a larger system where it communicates to other peripherals or to a host through I/O ports. Although a PSD5XX is selected for the design, this tutorial is applicable to both PSD4XX and PSD5XX based designs.

The PSD5XX family includes the following three PLDs (Programmable Logic Devices):

- GPLD (General Purpose PLD)
- DPLD (Decoding PLD)
- PPLD (Peripheral PLD)

## Functional Specifications

The main functional specifications of this design are shown in Table 1.

**Table 1. Functional Specification**

|                              |                                                                                                                                                                                        |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Processor</b>             | 16 MHz microcontroller with 16-bit multiplexed address/data bus; non-multiplexed address A16 – A19. With $\overline{RD}$ , $\overline{WR}$ , ALE and $\overline{BHE}$ control signals. |
| <b>Memory</b>                | 128KB EPROM (64K x 16), 2KB SRAM (1K x 16), with paging support.                                                                                                                       |
| <b>Counter/Timers</b>        | Event Counter, Watchdog Timer.                                                                                                                                                         |
| <b>Loadable Down Counter</b> | PLD to implement 5-bit down counter.                                                                                                                                                   |
| <b>I/O Ports</b>             | 1. One 6-bit output port.<br>2. One 4-bit input port.                                                                                                                                  |
| <b>Address Decoder</b>       | PLD generates all chip select signals                                                                                                                                                  |

**Design Example**

**Functional Specifications (Cont.)**

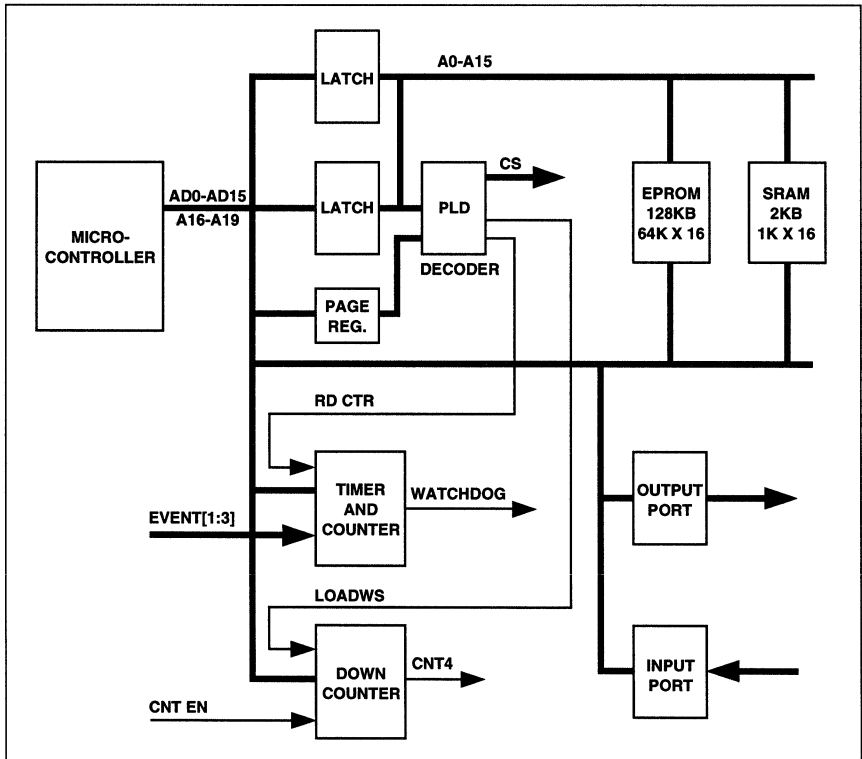
The system memory map is shown in Table 2. The EPROM/code memory consists of three 32KB blocks, where two of the blocks share the same address space (0-07FFF) and are in different memory pages.

Figure 1 is the functional block diagram of the tutorial design. PLDs are used wherever possible to reduce board space. Power consumption is also a major concern of this design.

**Table 2. System Memory Map**

| <i>Device</i>  | <i>Memory Space</i> | <i>Memory Page</i> |
|----------------|---------------------|--------------------|
| EPROM, Block 0 | 00000 – 07FFF       | Page 0             |
| EPROM, Block 1 | 00000 – 07FFF       | Page 1             |
| EPROM, Block 2 | 48000 – 4FFFF       | All Pages          |
| SRAM           | 08000 – 087FF       | All Pages          |
| I/O Devices    | 0C000 – 0CFFF       | All Pages          |

**Figure 1. Tutorial Design Example Block Diagram**



## Design Example

### Functional Partition

With the tutorial design defined, we can investigate the block diagram to see how much of the logic can be implemented in the PSD5XX. Some of the functions, such as the Event Counter, are available in the microcontroller but might require additional discrete logic to support specific applications. This imposes no problem in the PSD5XX because the Peripheral PLD (PPLD) can be programmed to implement any logic function.

The partitioning of the logic between the PSD5XX and the rest of the design example can be viewed as a top-level fitting process. First, we must go through the design functional specifications and block diagram to identify functions that can be implemented in the PSD5XX. The PSDsoft Development Tool performs the final fitting process.

From Table 3, it is obvious that the PSD5XX is more than able to meet all the required functional specifications of the design example.

A microcontroller running at 16MHZ has a T<sub>avdv</sub> (address valid to data valid time) of 138 ns. In order for the processor to run with zero wait states, it requires a PSD5XX-12 (120 ns part), which has sufficient speed to meet the T<sub>avdv</sub> requirement.

**Table 3. Functional Partition**

|                              | <b>System Functional Block</b>                                             | <b>Matching PSD5XX Functional Blocks</b>                                                                   |
|------------------------------|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| <b>Processor</b>             | 16-bit multiplexed address/data bus with $\overline{RD}$ , $\overline{WR}$ | Bus Interface accepts processor bus. No glue logic required.                                               |
| <b>Memory</b>                | 128KB EPROM with memory paging support and 2KB SRAM                        | Meets memory access time requirement; provides x16 configuration. A page register provides paging support. |
| <b>Timers</b>                | Event Counter, Watchdog Timer                                              | Event Counter: Use CTU 0<br>Watchdog: Use CTU 2                                                            |
| <b>Loadable Down Counter</b> | PLD: State machine to implement down counter                               | Use GPLD to implement state machine, Port B used as I/O                                                    |
| <b>I/O Ports</b>             | 1. One 6-bit output port<br>2. One 4-bit input port                        | 1. Port C or D<br>2. Port C or D                                                                           |



## **Design Example**

### **Functional Blocks**

The PSD5XX provides multiple system-level functional blocks and allows you to define and configure the blocks to meet the design specifications. There are three main blocks that you need to define and configure the PSD5XX.

- Bus Interface**
- Zero Power PLD (ZPLD) Block**
- I/O Ports**

**Bus Interface** – The PSD5XX Bus Interface allows communication to a microcontroller with no glue logic.

**ZPLD Block** – The DPLD defines the decoding function of the DPLD. The decoding function defines the memory address map and generates chip selects to internal PSD blocks, including the EPROM, SRAM, and I/O ports.

The GPLD defines the operation of the state machines and general-purpose logic.

The PPLD defines the Counter/Timer and Interrupt Controller control conditions.

**I/O Ports** – The I/O ports assign the functions of the forty I/O pins, including the MCU I/O function, ZPLD I/O function, Counter/Timer I/O function and other I/O functions.

---

## **PSDsoft Development Tools**

The PSD5XX functional blocks just described are supported by PSDsoft, an integrated system development software tool from WSI, which runs on a PC in the Microsoft Windows® environment. The PSDsoft tool consists of the following major modules:

- PSDabel**
- PSDconfiguration**
- PSDcompiler**
- PSDsimulator**
- PSDprogrammer**

### **PSDabel**

PSDabel is the WSI Windows version of the Data I/O ABEL design software. The .abl file, which defines the logic functions of the ZPLDs, can be compiled, optimized, and simulated in PSDabel. The PSDabel output is the .t2 file, which is the optimized PLA file.

### **PSDconfiguration**

The PSDconfiguration software tool allows you to specify the PSD5XX bus interface type and I/O port pin assignments. The output is the .glc configuration file.

## **PSDsoft Development Tools (Cont.)**

### **PSDcompiler**

The PSDcompiler software consists of two portions: the Fitter and Address Translator.

The Fitter, based on .tt2 and .glc input files, fits the logic and I/O functions you have specified into the PSD5XX.

The Address Translator performs address translation on your code (HEX) file. PSDcompiler also generates the object output file (.obj) for the programmer. The .obj file includes on-chip configuration data, the ZPLD fusemap, and user program codes.

The PSDcompiler also provides a decompilation function. Based on the .obj file, the Decompiler generates ZPLD and EPROM fuse map files for chip-level simulation.

### **PSDsimulator**

PSDsimulator is the WSI version of SIMUCAD PSDsilosIII Simulation Software. PSDsimulator provides PSD5XX chip-level simulation.

### **PSDprogrammer**

The PSDprogrammer software is the programming interface to the WSI PSD MagicPro™ Programmer. PSDprogrammer is used for downloading, uploading, and programming the PSD device.

### **PSDsoft Program Flow**

Figure 2 shows the PSDsoft program flow in configuring, defining and programming the PSD5XX. Each PSDsoft submodule is enclosed by dashed lines. The figure illustrates the normal steps you follow in creating a design. These steps are enumerated below. The files generated during the process are named using the project name you specify.

1. Create or open a project after entry into PSDsoft.
2. Use an editor in Windows or PSDsoft to generate the project.abl file.
3. Use PSDabel to compile and optimize the project.abl file. Perform simulation if needed. Generate an optimized PLA file (project.tt2) for the Fitter.
4. Configure the Bus Interface in PSDconfiguration. Generate the project.glc file for the Fitter.
5. Compile the design using PSDcompiler. Compilation consists of two steps:  
Fitting and Address Translation.

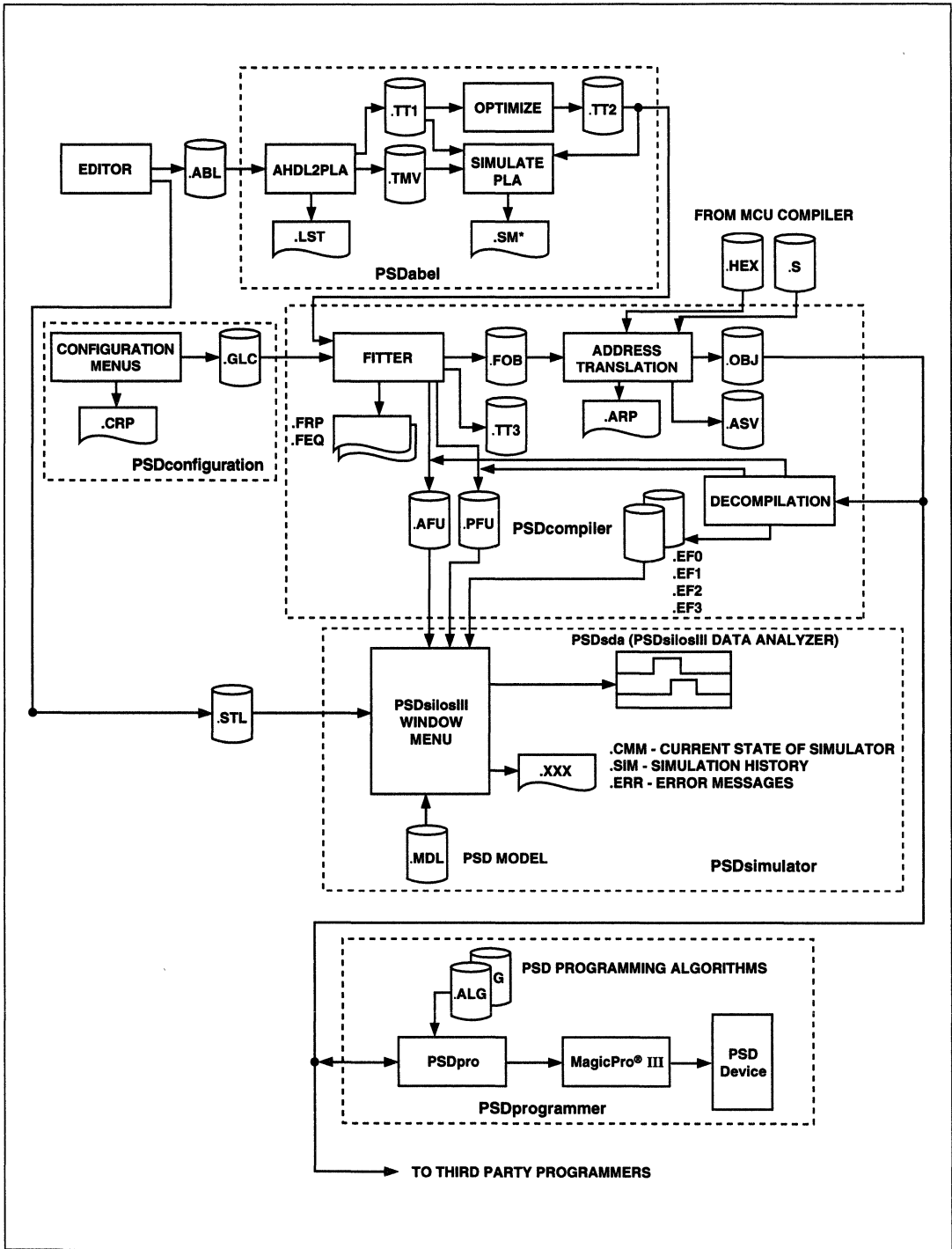
The Fitter generates the project.fob and fuse map based on the PSDabel and PSDconfiguration output files.

Address Translation combines the code file and the project.fob file into a project.obj file. This .obj file includes the program code, the PSD5XX fuse map, and the configuration bits.

6. Verify the design using PSDsimulator. Chip level simulation is based on the stimulus file (project.stl) and fusemap files from the Fitter.
7. Download the project.obj file to the PSD MagicPro programmer and use the PSDpro software to program the chip. A compatible third-party EPROM programmer can also be used.

For a description of all the files generated by PSDsoft, please refer to the appendixes of the PSDsoft manual.

Figure 2. PSD5XX PSDsoft Program Flow



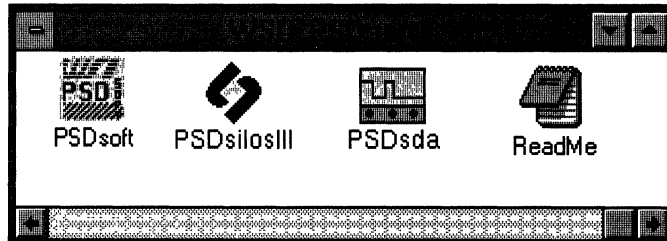
## Using the Design Example

This section uses the tutorial design example to illustrate the steps to invoke the software and create a design of your own. The files required, which are generated for the tutorial design, can be found in the \TUTORIAL directory after the PSDsoft software is installed. If changes are made during this tutorial, the corresponding file in the TUTORIAL directory will be changed as well. If you are unsure as to the status of the tutorial files when you are finished going through the tutorial design example, you may reinstall the software to restore the files to their original state.

To enter the PSDsoft program

1. Install the PSDsoft software.

The WSI-PSDsoft window with four distinct icons appears.



2. Double-click the ReadMe icon.

Important information you should know about PSDsoft is presented. Read this information before proceeding. A list of the PSDcontrol error messages is included in the ReadMe file. The PSDsilosIII™ icon invokes the PSDsilosIII simulator for chip-level simulation. The PSDsda (PSDsilosIII Data Analyzer) icon, which is also available under PSDsilosIII, allows you to display multiple simulation results.

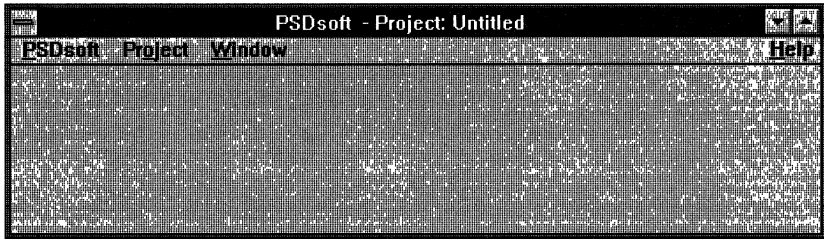
**3**

**Using the Design Example (Cont.)**

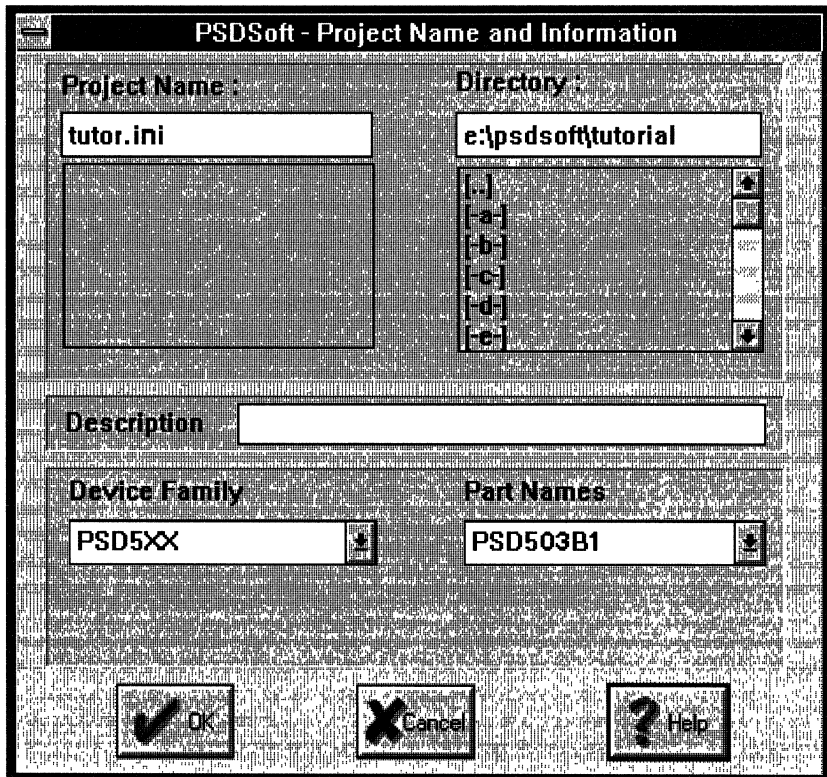
**Managing the Project**

Each new project may have its own working directory where all the files generated by PSDsoft reside. Once you specify the new project name, the PSDsoft Project Management passes the working directory and pertinent information to other functional modules. In the following sections, key windows are displayed to explain the step-by-step operation of PSDsoft.

1. Double-click the PSDsoft icon in the WSI-PSDsoft window.  
The Main PSDsoft window is displayed.



2. Pull down the Project menu and select **New**.  
The project window appears.



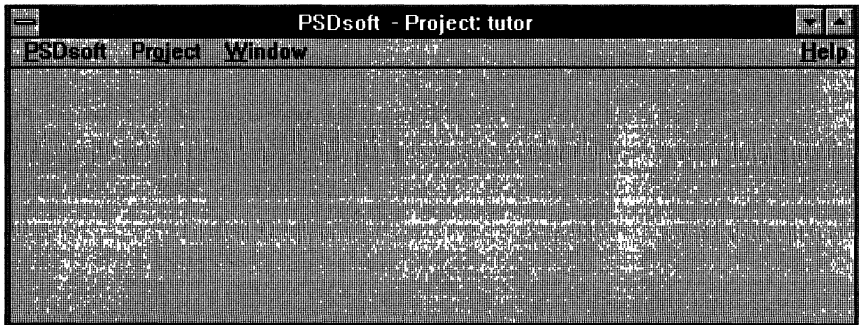
**Using the  
Design  
Example  
(Cont.)****Managing the Project (Cont.)**

3. Enter the name of the directory in which you want the new project to reside in the Directory window.
4. Enter the project name in the Project Name window. The examples in this tutorial are based on the name "tutor."

The project name you enter will be used as the file name in all the files generated, including the .abl file.

5. Specify the device family and part name (PSD503B1 for Design Tutorial).
6. Click OK after all the parameters are entered to your satisfaction.

PSDsoft creates a new project named TUTOR, generates a tutor.ini file for the TUTOR project, and presents the PSDsoft Menu, which now reflects the name of the project.

**3**

**Using the  
Design  
Example  
(Cont.)**

**Entering the Design Source File**

PSDabel is the design entry tool used to define the ZPLD and some I/O constructs. Because the tutorial design example has already been created for you, you do not need to create a design from scratch. However, if you were creating a new design, you would pull down the PSDsoft menu and choose PSD ABEL Design Entry. A window would open to allow you to enter the design. The material that follows is presented for you to understand the components of a source file. Following the source file explanation, we will continue by compiling the tutorial source file.

PSDabel is WSI's version of Data I/O's ABEL-HDL Design Software, and includes all the ABEL functions required to compile, optimize, and simulate the PLD source file written in ABEL Hardware Description Language (PSDabel-HDL).

The logic functions of the PSD5XX's ZPLDs can be defined entirely by PSDabel-HDL. PSDabel takes the PLD input source file and generates an output file (.tt2) after compilation and optimization. A source file consists of one or more modules, and each module has its own beginning and end. The source file that describes the PSD5XX's ZPLDs can consist of one large module, or it can be implemented in three modules, one each for the DPLD, GPLD, and PPLD.

A module usually consists of five sections:

1. **Header.** A header consists of a module name and/or title. The module name must be the same as the file name of the .abl file.
2. **Declarations.** Declarations define signals, constants, and macros. No device declaration is required.
3. **Logic Description.** The logic description defines the PLD functions in terms of equations, truth tables, and state diagrams.
4. **Test\_Vectors.** The Test\_Vectors are used in logic simulation (only in the ABEL Simulator).
5. **End.** A module must include the End statement.

## Using the Design Example (Cont.)

### Entering the Design Source File (Cont.)

The listing of a typical source file is shown below. Source statements end with semicolons; comments begin with a double quotation mark. Keywords are indicated in bold letters.

```

module DPLD

title 'Decoding equations for internal PSD5XX devices'

Declarations

"Input Signals, external or internal PSD5XX signals

a18,a17,a16,a15,a14,a13 pin;"address lines,
a12,a11,a10,a9,a8,a1,a0 pin;"using reserved names
pgr3,pgr2,pgr1,pgr0 pin;"page register embedded inputs

"Output signals, internal PSD5XX DPLD output signals.
csiop,rs0,es0,es1,es2,es3 node ; "using reserved names.

"definitions
X = .x. ;
page = [pgr3,pgr2,pgr1,pgr0];
Address = [a18,a17,a16,a15,a14,a13,a12,a11,
 a10,a9,a8,X,X,X,X,X,X,a1,a0];

Equations

csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF);
rs0 = (Address <= ^h087FF) & (Address >= ^h08000);
es0 = (Address <= ^h07FFF) & (page == 0);
es1 = (Address <= ^h07FFF) & (page == 1);
es2 = (Address <= ^h4FFFF) & (Address >= ^h48000);

Test_Vectors
([page, Address] > [es0, es1, es2])
[0 , ^h07020] > [1 , 0 , 0];
[1 , ^h07020] > [0 , 1 , 0];
[0 , ^h4A000] > [0 , 0 , 1];
[0 , ^h50000] > [0 , 0 , 0];

End DPLD

```



**Using the  
Design  
Example  
(Cont.)****Entering the Design Source File (Cont.)****Writing the Source (.abl) File**

Keep the following things in mind when you write the .abl file:

- Nodes and Pins**
- Embedded Nodes**
- Reserved Names**
- Pin Assignments**
- Node Assignments**

**Nodes and Pins**

In PSDabel-HDL, the keyword PIN refers to input and output signals that are available on the device's pins. For the PSD5XX family, signals such as RD, WR, and A8–A15 are defined as pin inputs to the ZPLD. The NODE keyword refers to signals that are buried or embedded inside the device. However, the Fitter does have the option to assign a NODE to a pin. The ZPLDs in the PSD5XX are embedded inside the chip and some of its input and output signals are actually internal nodes. For example, the outputs from the Page Register are internal nodes.

**Embedded Nodes**

Not all state outputs of a state machine need to be routed to an output pin. For example, in a state machine (8-bit counter) that takes up all eight PA macrocells (macrocells connected to Port A), only the MSB of the counter is needed for external logic. In this case, pin PA0 is fitted to provide the MSB, while PA1-PA7 are available for other Port functions such as I/O ports for the microcontroller.

**Reserved Names**

There are 61 input signals to each of the ZPLDs. The number of ZPLD outputs are variable and depend on user application. Some of these signals have reserved names (dedicated names), as they represent a hardwired function. The reserved names are required by the Fitter (PSDcompiler), which has to recognize the functions of these signals in order to perform the proper fitting and generate the correct fusemap. For example, the reserved names for the address lines are A0-A23. Wherever the address lines are involved, you must use the reserved names A0-A23 or aliases of these signals.

Some of the pins on the PSD5XX have multiple functions. Pin 41 is the “read” pin with the reserved name “RD”. The RD pin can also be configured to accept other bus control signals (E, DS, or LDS/). If you prefer to use the name “DS/” (for 68332-based design) instead of RD, the .abl file should contain the declaration:

```
ds pin 41;
```

The ZPLD input signals are listed in Table 4 with their corresponding reserved names.

Table 5 shows the output signals of the ZPLD that have a reserved name.

**Using the  
Design  
Example  
(Cont.)**

**Table 4. ZPLD Input Signals**

| <b>Signals/Reserved Names</b>      | <b>Signal Source</b>   |
|------------------------------------|------------------------|
| pa0 – pa7 (a16 – a23) <sup>1</sup> | Port A                 |
| pb0 – pb7                          | Port B                 |
| pc0 – pc7 (a16 – a23) <sup>2</sup> | Port C                 |
| pd0 – pd7 (a16 – a23) <sup>3</sup> | Port D                 |
| pe0 – pe7                          | Port E                 |
| pgr0 – pgr3                        | Page Mode Registers    |
| wdog2pld                           | Counter/Timer          |
| intr2pld                           | Interrupt Controller   |
| a0 – a15, a0, a1                   | MCU Address Bus        |
| rd/                                | MCU Bus Control Signal |
| wr/                                | MCU Bus Control Signal |
| clkin                              | Input Clock            |
| reset                              | Reset Input            |
| csi/                               | CSI Pin                |
| timerout0 – 3 <sup>4</sup>         | Input Clock            |

**NOTES:**

1. Port A can be assigned by the Fitter or by the user as high address line inputs to the DPLD for decoding.
2. If A16 – A23 are not used as inputs to the DPLD, the Fitter can assign A16 – A23 to Port C or D.
3. If A16 – A23 are not used as inputs to the DPLD, the Fitter can assign A16 – A23 to Port C or D.
4. Available only if Timer Output is selected.

**Table 5. ZPLD Output Signals**

| <b>Signals/Reserved Names</b> | <b>Signal Source/Destination</b> |
|-------------------------------|----------------------------------|
| es0 – es3                     | DPLD/EPROM Block Chip Selects    |
| rs0                           | DPLD/SRAM Chip Select            |
| csiop                         | DPLD/PSD I/O Port Chip Select    |
| pse10 – pse11                 | DPLD/Port Peripheral I/O         |
| mc2tmr0 – mc2tmr3             | PPLD/Counter/Timer               |
| pt2int4 – pt2int5             | PPLD/Interrupt Controller        |
| mc2int6 – mc2int7             | PPLD/Interrupt Controller        |

## Using the Design Example (Cont.)

### Entering the Design Source File (Cont.)

#### Pin Assignments

The GPLD has a maximum of 24 macrocells. The inputs or outputs of the macrocells are connected to Ports A, B and E. Unless you specify otherwise, the Fitter assumes all the port pins are available for the GPLDs. If the ports are used for other functions, such as to provide latched address out or as I/O ports for the microcontroller, you must specify in the .abl file that these port pins are not available for fitting. For example, if Port A pins PA0-PA3 are used as latched address pins out or as MCU I/O ports, the .abl file should include the statement:

```
pa0, pa1, pa2, pa3 pin;
```

#### Node Assignments

The macrocells of Port A, B and E all have dedicated node numbers. If you have an embedded function and wish to assign it to a specific macrocell, you need to assign the function (signal) to the macrocell node number. Table 6 shows the node number of the macrocells.

**Table 6. Macrocells Node Number**

| <b>Macrocell</b> | <b>Node Number</b> | <b>Macrocell</b> | <b>Node Number</b> | <b>Macrocell</b> | <b>Node Number</b> |
|------------------|--------------------|------------------|--------------------|------------------|--------------------|
| PA0              | 27                 | PB0              | 50                 | PE0              | 38                 |
| PA1              | 26                 | PB1              | 49                 | PE1              | 37                 |
| PA2              | 25                 | PB2              | 48                 | PE2              | 36                 |
| PA3              | 24                 | PB3              | 47                 | PE3              | 34                 |
| PA4              | 23                 | PB4              | 46                 | PE4              | 33                 |
| PA5              | 22                 | PB5              | 45                 | PE5              | 32                 |
| PA6              | 21                 | PB6              | 44                 | PE6              | 31                 |
| PA7              | 20                 | PB7              | 43                 | PE7              | 30                 |

## Using the Design Example

(Cont.)

### The tutor.abl Source File

The ZPLD source file for the Tutor design (tutor.abl) is shown in the listing that follows. Source statements end with semicolons; comments begin with a double quotation mark.

```

module tutor
title 'tutor design example ZPLD source file';
"Input signals
"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
a18,a17,a16 pin ; "high order address
 "input for fitting

pgr3,pgr2,pgr1,pgr0 pin; "page register embedded inputs
bhe pin 38; "reserving pe0 for bhe
clkln, reset pin 42, 40; "using the right pin numbers,
 "inames are modified.

"General inputs for fitting

event1,event2,event3 pin;
cntout_en pin; "Enable down counter output
load pin; "Load and enable down counter
d4, d3, d2, d1, d0 pin; "Number of counts to load,
 "connect to processor data bus

wdog2pld node; "watch dog output from Counter/Timer

```

**Using the  
Design  
Example  
(Cont.)**

**Entering the Design Source File (Cont.)**

**The tutor.abl Source File (Cont.)**

```
"Output signals
csiop, rs0, es0, es1, es2 node ;
 "DPLD output chip selects
mc2tmr0 node ; "PPLD output to Event Counter

"General outputs

wstc pin; "down counter terminal count
cnt3, cnt2, cnt1, cnt0 node istype 'reg';
cnt4 pin istype 'reg';

"Cnt4-cnt0 implement a down counter with a parallel load.
"Only Cnt4 can drive out, cnt3-cnt0 are embedded in the chip.

out_p0,out_p1,out_p2 pin 60,59,58;
"assign 3-bit output port to PD0-PD2

wdout pin; "watch dog output

"DEFINITIONS

din = [d4,d3,d2,d1,d0];
cnt = [cnt4,cnt3,cnt2,cnt1,cnt0];
page = [pgr3,pgr2,pgr1,pgr0];
CK = .c. ; "Clock pulse definition
X = .x. ; "Don't care
Address =
[a18,a17,a16,a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];
event_in = [event3,event2,event1];
```

## Using the Design Example (Cont.)

### Entering the Design Source File (Cont.)

#### The tutor.abl Source File (Cont.)

equations

"DPLD EQUATIONS

```

csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF) ;
 "256 block
rs0 = (Address <= ^h087FF) & (Address >= ^h08000);
 "2k block
es0 = (Address <= ^h07FFF) & (page == 0);
 "32k block only at page 0
es1 = (Address <= ^h07FFF) & (page == 9);
 "32k block only at page 9
es2 = (Address <= ^h4FFFF) & (Address >= ^h48000);
 "32k block, always visible

```

"GPLD Equations

```

cnt.clk = clkln; " The global clock is the counter clock
cnt.re = !reset; " The global Reset is the counter reset
cnt4.oe = !cntout_en;
wstc = (cnt.fb == 0); "wstc is true when the counter outputs are zeroes

WHEN (load) THEN cnt := din;
 "Load cnt with din if load is true
ELSE WHEN (wstc) THEN cnt := 0;
 "Wait for a load pulse
ELSE cnt := cnt.fb - 1; "Count-down

```

"PPLD Equations

" PLD-driven Event Counter event inputs.

```
mc2tmr0 = event1 & !event2 # !event3;
```

```
wdout = !wdog2pld & (cnt.fb == 2);
```

**Using the  
Design  
Example  
(Cont.)**

**Entering the Design Source File (Cont.)**

**The tutor.abl Source File (Cont.)**

```

" TEST VECTORS

test_vectors([clkln,reset,load,din]->[wstc,cnt])

[CK,1,X,X]->[1,0]; "Reset is on, cnt reg. is at zero
[CK,0,0,X]->[1,0]; "No output change, wait for a load pulse
[CK,0,1,17]->[0,17]; "Load 17 to down-counter
[CK,0,0,X]->[0,16]; "Count down
[CK,0,0,X]->[0,15]; "Count down
[CK,0,0,X]->[0,14]; "Count down
[CK,0,0,X]->[0,13]; "Count down
[CK,0,0,X]->[0,12]; "Count down
[CK,0,0,X]->[0,11]; "Count down
[CK,0,0,X]->[0,10]; "Count down
[CK,0,0,X]->[0,9]; "Count down
[CK,0,0,X]->[0,8]; "Count down
[CK,0,0,X]->[0,7]; "Count down
[CK,0,0,X]->[0,6]; "Count down
[CK,0,0,X]->[0,5]; "Count down
[CK,0,0,X]->[0,4]; "Count down
[CK,0,0,X]->[0,3]; "Count down
[CK,0,0,X]->[0,2]; "Count down
[CK,0,0,X]->[0,1]; "Count down
[CK,0,0,X]->[1,0]; "Wait-State Terminal Count is set
[CK,0,0,21]->[1,0]; "Maintain same state
[CK,0,1,21]->[0,21]; "Load a new value
[CK,0,0,X]->[0,20]; "Count down
[CK,0,0,X]->[0,19]; "Count down
[CK,0,0,X]->[0,18]; "Count down

END tutor

```

## Using the Design Example (Cont.)

### Entering the Design Source File (Cont.)

Equations for the DPLD, GPLD and PPLD are included in one file. Some of the logic implementation and signal names are described in the following paragraphs.

#### Down Counter (5-bit)

|           |                                   |
|-----------|-----------------------------------|
| load      | input, load input                 |
| cntout_en | input, OE/ to counter output cnt4 |
| d[0:4]    | input, data input for loading     |
| cnt[0:4]  | counter outputs                   |

The Down Counter is implemented in the GPLD. Signals cnt0-cnt3 are embedded internal nodes. Only the cnt4 signal is driven out to a pin.

#### Event Counter

|            |                                                                                            |
|------------|--------------------------------------------------------------------------------------------|
| event[1-3] | event inputs                                                                               |
| mc2tmr0    | count enable input to Counter, function of event[1-3]. Count if (event1&!event2 # !event3) |

The Event Counter is implemented by Counter/Timer Unit 0. The output of the counter can be read by the microcontroller.

#### I/O Ports

|            |                               |
|------------|-------------------------------|
| in_p[0:3]  | input port, assign to Port C  |
| out_p[0:5] | output port, assign to Port D |

3

#### Watchdog Timer

|       |                                |
|-------|--------------------------------|
| wdout | output to pin, wdout= wdog2pld |
|-------|--------------------------------|

The Watchdog Timer is implemented in the Counter/Timer Unit 2. Output of timer, wdog2pld, is routed to output pin wdout.

#### Checking for Errors

Now that the tutor.abl file is complete, the next step is to check the syntax of the file. If there are no errors, the file is compiled and/or optimized to generate the tutor.t2 file for the Fitter.

In the previous section, PSDsoft was invoked and the project Tutor was specified.

1. Pull down the PSDsoft menu and select PSDlabel.

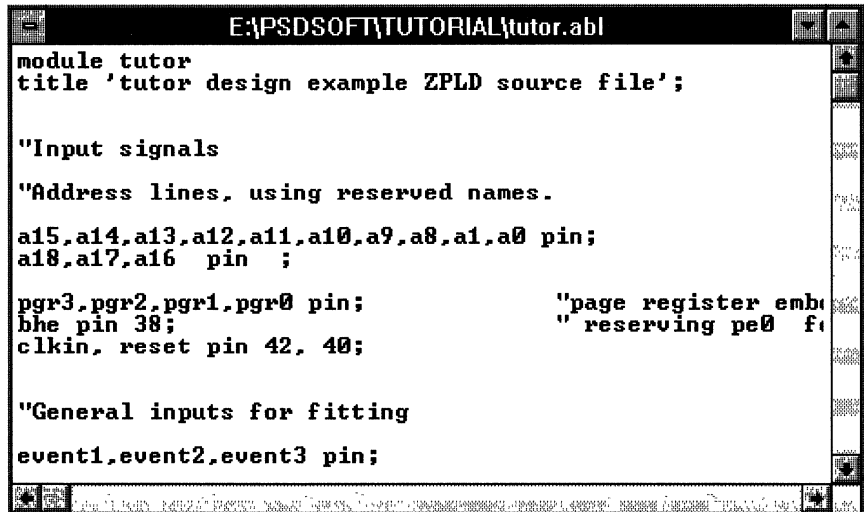
The tutor.abl file is displayed.



## Using the Design Example (Cont.)

### Entering the Design Source File (Cont.)

#### Checking for Errors



```

E:\PSDSOFT\TUTORIAL\tutor.abl
module tutor
title 'tutor design example ZPLD source file';

"Input signals

"Address lines, using reserved names.
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
a18,a17,a16 pin ;

pgr3,pgr2,pgr1,pgr0 pin; "page register emb
bhe pin 38; "reserving pe0 f
clkln, reset pin 42, 40;

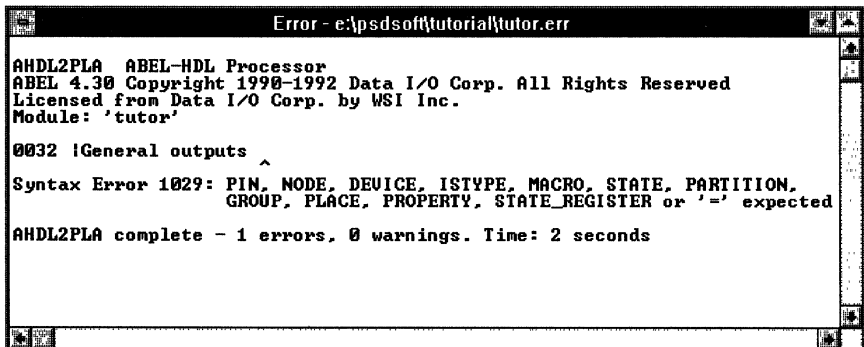
"General inputs for fitting
event1,event2,event3 pin;

```

PSDsoft automatically opens the window displaying the tutor.abl file, which already exists. If you were creating a new design, the window would be empty and you would need to enter a design.

2. Pull down the Compile menu and select Error Check.

If any errors are present, a window is created to display the error file, TUTOR.err.



```

Error - e:\psdsoft\tutorial\tutor.err

AHD2PLA ABEL-HDL Processor
ABEL 4.30 Copyright 1990-1992 Data I/O Corp. All Rights Reserved
Licensed from Data I/O Corp. by WSI Inc.
Module: 'tutor'

0032 !General outputs
Syntax Error 1029: PIN, NODE, DEVICE, ISTYPE, MACRO, STATE, PARTITION,
GROUP, PLACE, PROPERTY, STATE_REGISTER or '=' expected

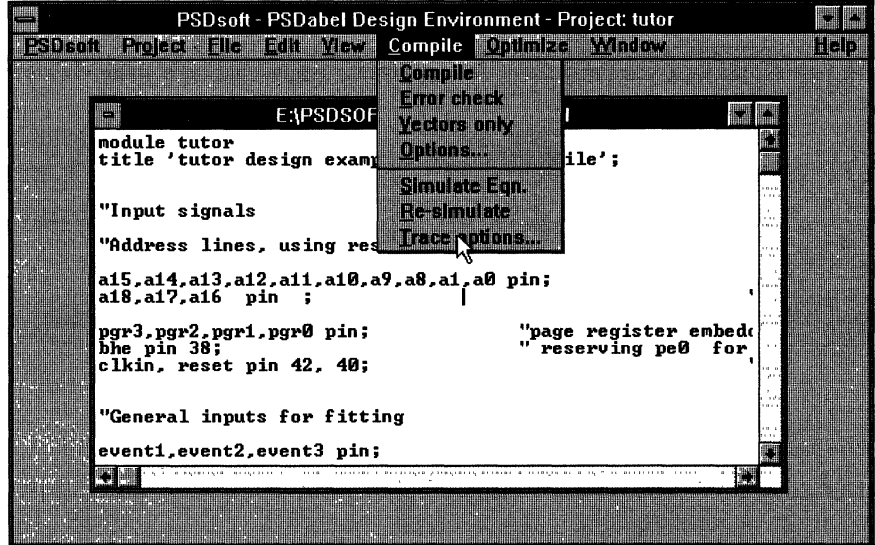
AHD2PLA complete - 1 errors, 0 warnings. Time: 2 seconds

```

## Using the Design Example (Cont.)

### Compiling the Source File

With the tutor.abl window still displayed, pull down the Compile menu and select **Compile**.



3

Compile generates two PLA output files: tutor.tt1 and tutor.tt2. The tutor.tt2 file is the optimized PLA file based on the reduction algorithm specified in the Optimize menu. A default optimize setting is used during the tutorial.

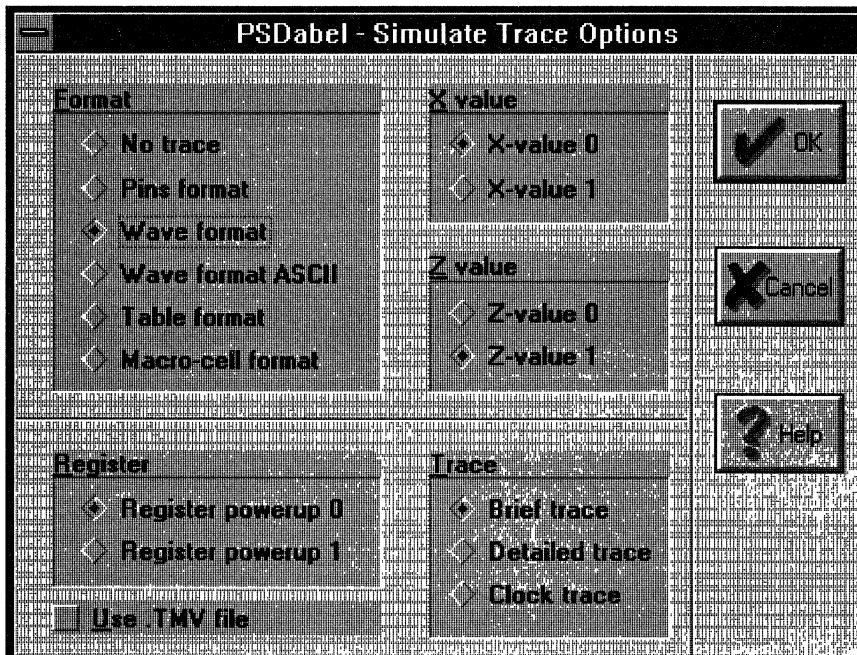
**Using the Design Example (Cont.)**

**Choosing Trace Options for the Source File**

The Compile Menu includes items that will be used by the PSDabel simulator, which simulates the equations generated in the tutor.tt1 file.

1. Pull down the Compile menu and select Trace Options.

The Simulate Trace Option window appears, allowing you to select the simulation output format and other options.



2. Click **Wave format**.

Table format is the default option. It generates a file during simulation that produces a waveform for each of the signals simulated (test vectors).

3. Click **OK**.

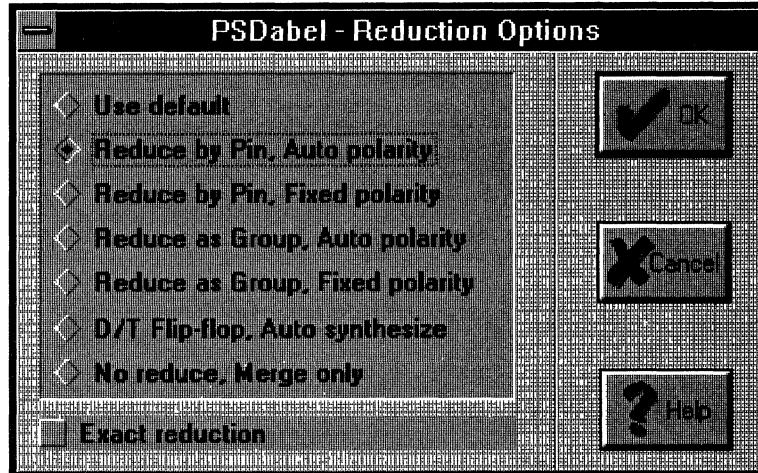
The PSDabel main window reappears.

**Using the  
Design  
Example  
(Cont.)**

**Choosing Optimize Options for the Source File**

You can choose different reduction algorithms to optimize the tutor.abl file.

1. Pull down the Optimize menu and select **Options**.  
The Reduction Options window appears.



3

The default option is the **Reduce by Pin, Auto Polarity** reduction. This is the option that is used for the tutorial design example. The purpose of optimization is to provide the optimal PLA file to the Fitter.

2. Click **OK**.  
The main PSDabel menu reappears.

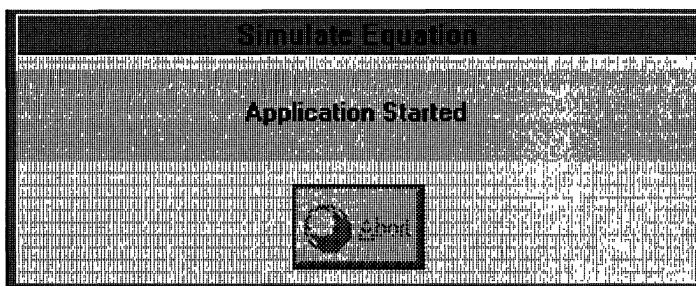
**Using the  
Design  
Example  
(Cont.)**

**Simulation Within PSDabel**

A presimulation capability is available within PSDabel. Simulation at this level is device- and pin-independent. The advantage of this is that you can generate waveforms based on the logic design or logic equations without any concern about a package for the design.

To perform a presimulation and view the results, pull down the Compile menu from the PSDsoft main menu and select **Simulate Eqn.**

The Simulate Equation progress window appears.



This window indicates that PSDabel is performing a simulation of the logic equations for the tutorial example and generating a waveform file for viewing.

## Using the Design Example (Cont.)

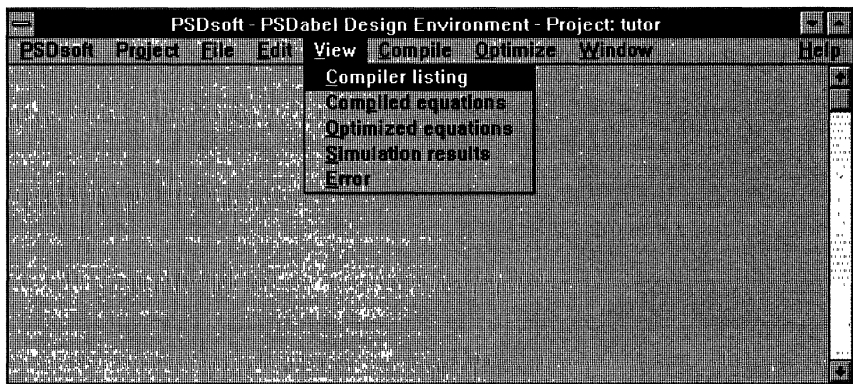
### Viewing Source File Components

The View Menu allows you to view the following:

- Compiler Listing
- Simulation Results
- Compiled Equations
- Optimized Equations
- Errors

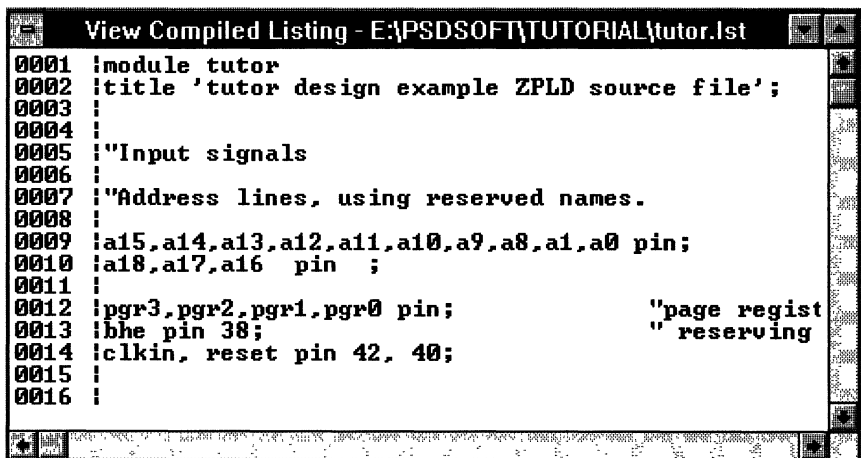
The Compiler Listing and Simulation Results are given here as examples of what might be displayed.

1. Pull down the View menu to display all the items that are available for display.  
The items available are shown below.



2. Choose the Compiler listing item.

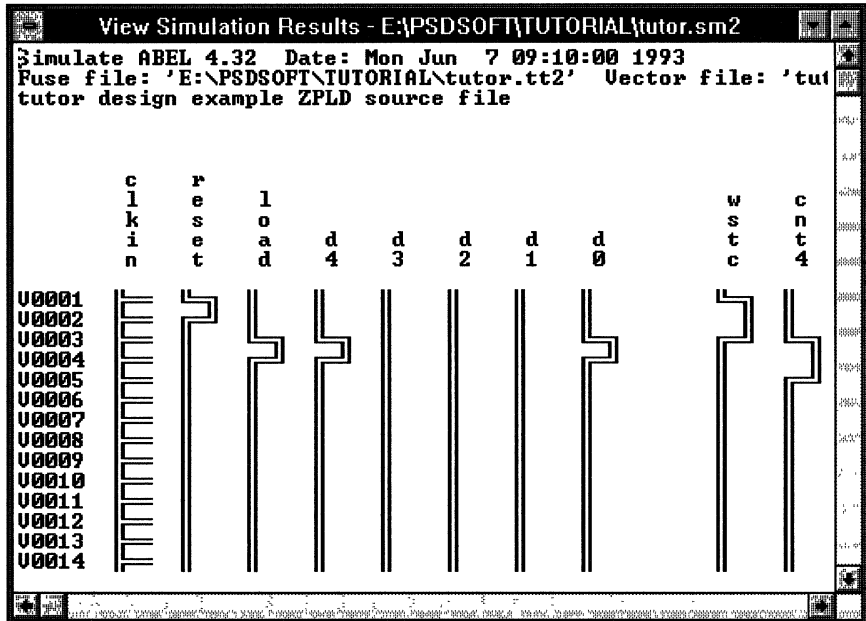
The compiler listing tutor.lst appears in a window.



**Using the Design Example (Cont.)**

**Viewing Source File Components (Cont.)**

3. Return to the PSDabel main menu by double-clicking the close box (upper left corner) of the tutor.lst window.
4. Pull down the View menu and select **Simulation Results**.  
A window showing the simulation results appears.



**Exiting PSDabel**

Pull down the PSDsoft menu in the main PSDsoft window and choose **Exit PSDabel**.

You are now ready to configure the tutorial design.

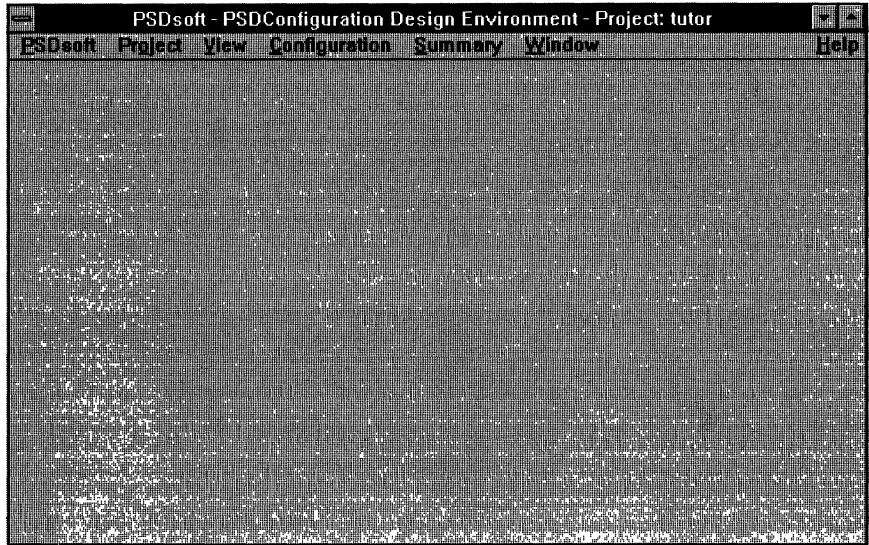
**Using the  
Design  
Example  
(Cont.)****Configuring the Design**

Generating the optimal PLA file is critical in the fitting of the ZPLD and the development of the PSD5XX. After simulating the PLA file and verifying that the logic functions are correct, the next step is to invoke the PSDconfiguration software.

The PSD5XX has a programmable bus interface and is able to interface directly to many microcontrollers. The PSDconfiguration software allows you to specify the bus configuration of the PSD5XX. The tutorial design is based on a microcontroller having a 16-bit multiplexed bus with RD, WR/, and BHE/as control signals.

To perform the design configuration

1. Pull down the PSDsoft menu in the main PSDsoft window and choose PSDconfiguration. The PSDconfiguration main window appears.

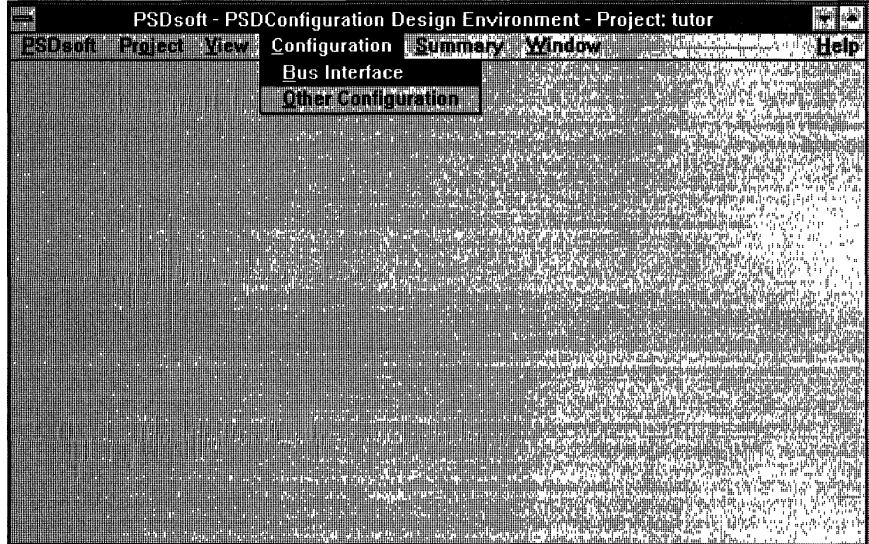
**3**



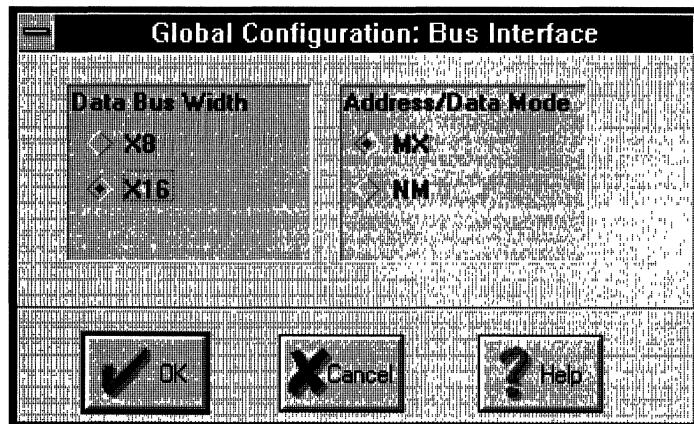
**Using the Design Example (Cont.)**

**Configuring the Design (Cont.)**

2. Pull down the Configuration menu and select Bus Interface.



The Bus Interface window appears.



## Using the Design Example (Cont.)

### Configuring the Bus Interface

To program the Bus Interface

1. Select the data bus width (X8 or X16) and type of microcontroller bus.  
MX designates a multiplexed microcontroller bus and NM designates non-multiplexed.

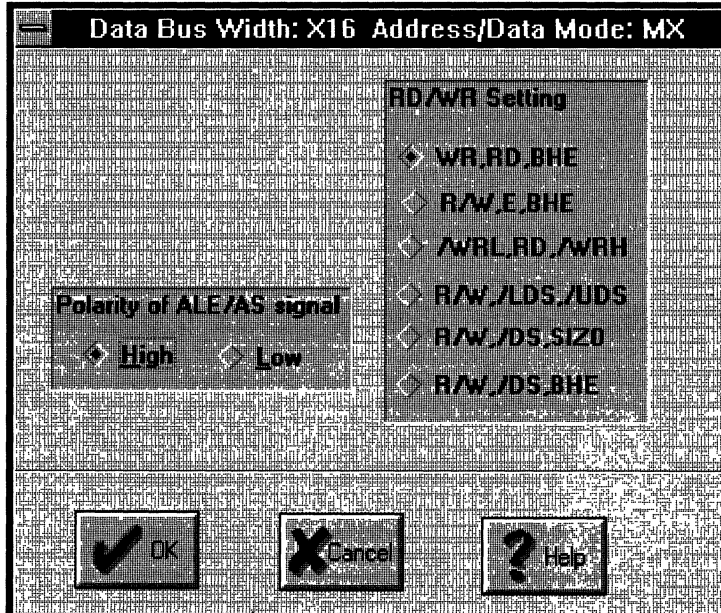
There are four combinations of bus types, as follows:

- 16-Bit Data Bus**  
Multiplexed address/data bus
- 16-Bit Data Bus**  
Non-multiplexed bus
- 8-Bit Data Bus**  
Multiplexed address/data bus
- 8-Bit Data Bus**  
Non-multiplexed bus

The tutorial design uses the 16-bit, multiplexed address/data bus.

2. Click OK.

The Control Signal window for a 16-bit, multiplexed bus appears.



3. Specify your selections for the Address Latch Enable (ALE) polarity and the control signals.

For the tutorial design example, the microcontroller bus configuration is a 16-bit multiplexed bus, the ALE polarity is high, and the RD, WR, and BHE setting is used.

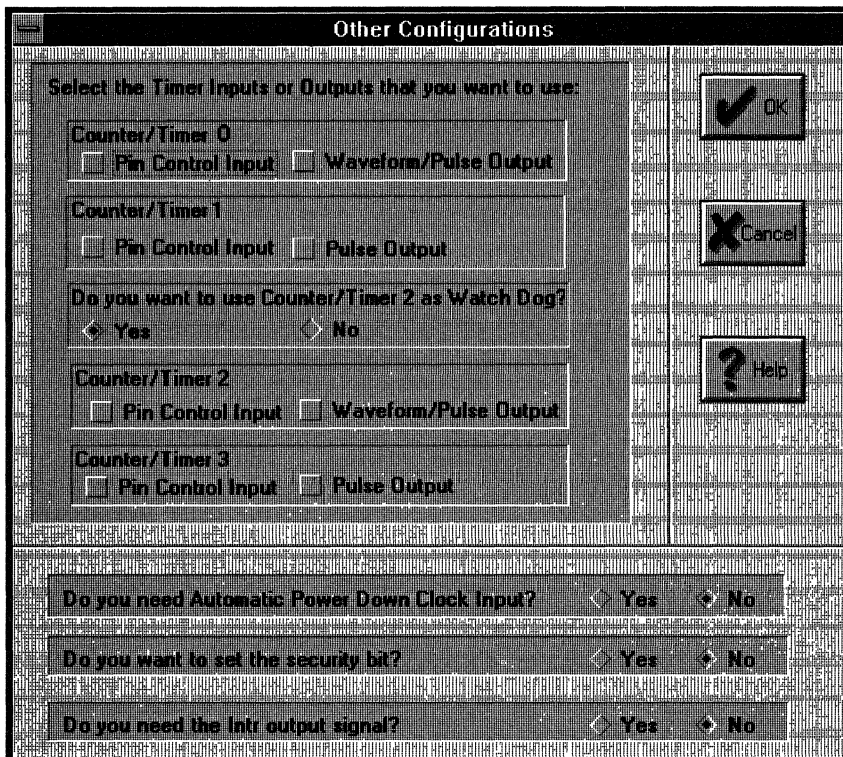
## Using the Design Example (Cont.)

### Configuring the Bus Interface (Cont.)

#### Configuring the Rest of the Design

Besides configuring the Bus Interface, you must program the I/O configuration of the Counter/Timers and Interrupt Controller. To do this

1. Return to the main PSDconfiguration menu.
2. Pull down the Configuration menu and select **Other Configuration**.  
The Other Configuration window appears.



3. Select the configuration for the four Counter/Timers inputs and outputs.

The PSD5XX has reserved input and output pins that are routed directly to the Counter/Timer Units and the Interrupt Controller. PA0-PA3 and PB0-PB3 can be assigned as output ports and PE3-PE6 as input ports for the Counter/Timers.

In the tutorial design, the input to Counter/Timer 0 cannot be assigned to pin PE3 because it consists of three signals (event[1-3]). For this reason, the Counter/Timer 0 is not selected in the Configuration Window.

4. Select the automatic power-down configuration.

The input clock to the Automatic Power Down counter is pin PE7. If PE7 is not used, the Fitter considers it available for fitting ZPLD functions.

## Using the Design Example (Cont.)

### Configuring the Bus Interface (Cont.)

#### Configuring the Rest of the Design

5. Select the security bit configuration.

The security bit can be set in the Other Configuration window and is then embedded in the tutor.obj file generated by the Address Translator. Once the security bit is set, the EPROM or the ZPLD fusemap cannot be copied until it is erased.

6. Select the interrupt controller output configuration.

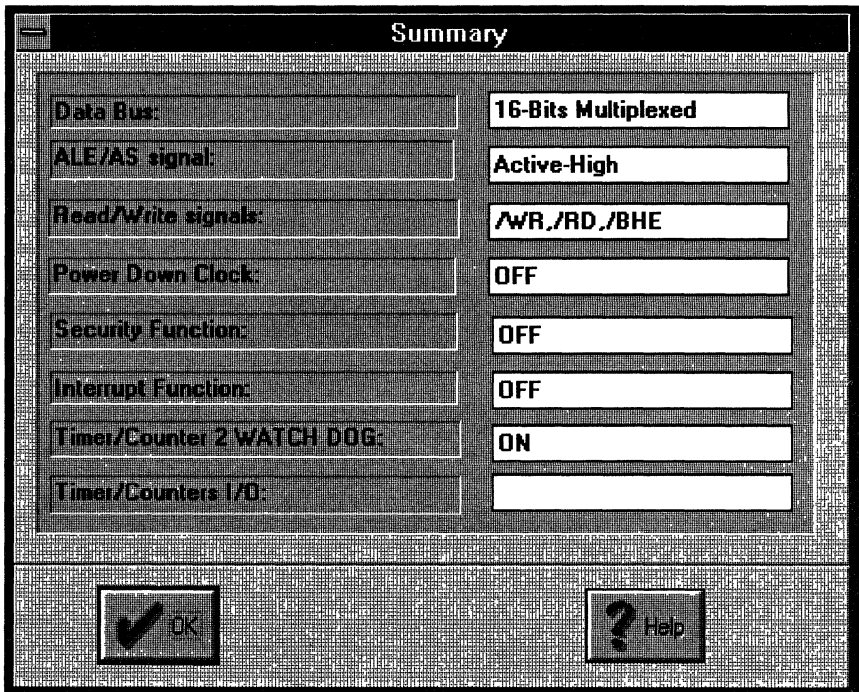
The output of the interrupt controller is pin PE2. If PE2 is not used, the Fitter considers it available for fitting ZPLD functions.

#### Viewing the Design Summary

When you are ready to conclude your configuration session

1. Pull down the Summary menu in the main PSDconfiguration menu and choose **Summary**.

The Summary window appears and displays the configuration specified in the Bus Interface and Other Configuration menus.



3

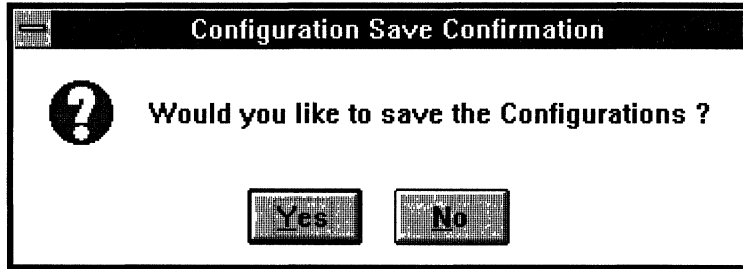
2. Return to the Configuration menu in the main PSDconfiguration window if you want to change any of the PSD5XX configuration parameters.

**Using the Design Example (Cont.)**

**Configuring the Bus Interface (Cont.)**

**Viewing the Design Summary (Cont.)**

3. Pull down the PSDsoft menu in the main window and choose **Exit PSDconfiguration**. The Configuration Save Confirmation window appears.



4. Click **Yes** to save the new configuration in the tutor.glc file.

---

**Compiling the Design**

The PSDcompiler consists of the Fitter and the Address Translator programs. The function of the Fitter is to fit the logic functions into the ZPLD. Once this is done, pin assignments may be made on the design schematic. The constraint of the fitting process is imposed by the GPLD architecture and the availability of I/O pins on the PSD5XX.

The input files to the Fitter are as follows:

- tutor.tt2      PLA file
- tutor.glc      PSD5XX configuration file

The output files generated by the Fitter are:

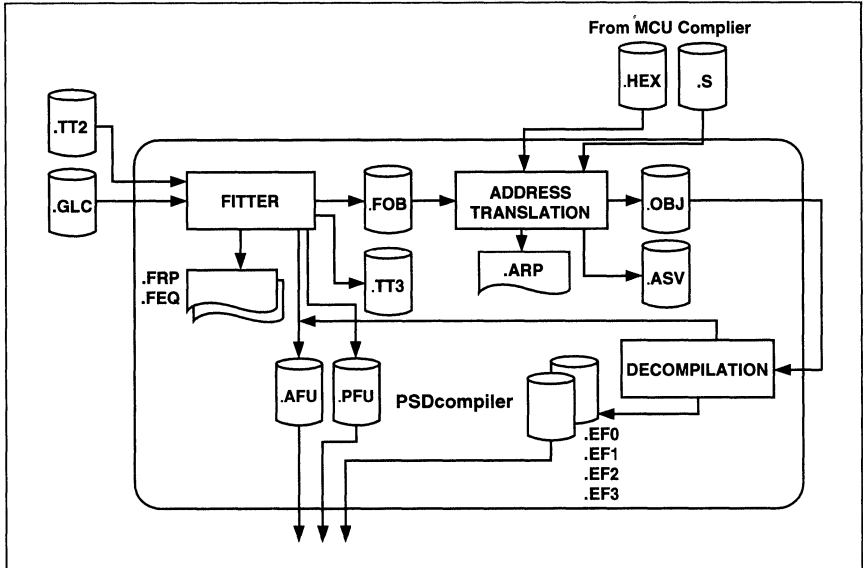
- tutor.fob      Fusemap file in Hex format (PLD+Configuration) for Address Translator
- tutor.tt3      Fitted PLA file
- tutor.afu      Configuration fuse file for PSDsimulator
- tutor.pfu      PLD fuse file for PSDsimulator
- tutor.feq      Fitter equation file using device reserved names
- tutor.frp      Fitter pin assignment file
- tutor.err      Fitter error file

The Address Translator combines the tutor.fob fusemap file with the EPROM codes file and generates the tutor.obj file, which is to be downloaded to EPROM programmer for PSD5XX programming.

Figure 3 shows the flow and input and output files of the PSDcompiler.

**Using the Design Example (Cont.)**

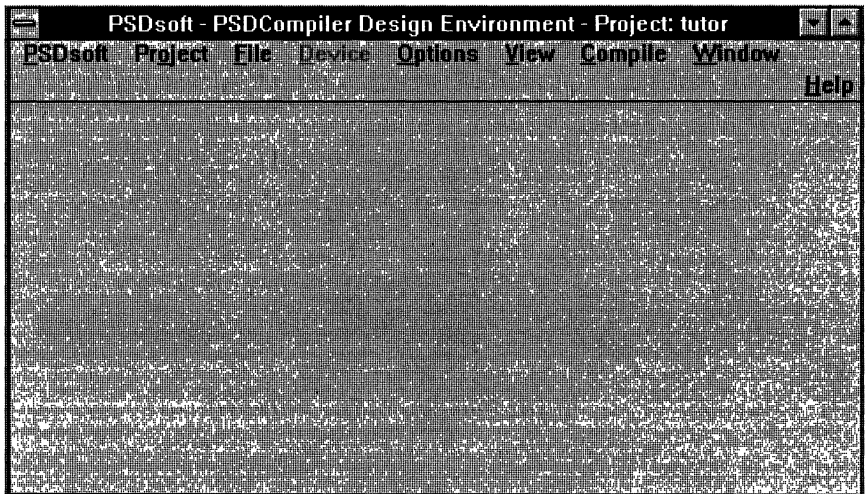
**Figure 3. PSDcompiler Program Flow**



**3**

To compile a design

1. Pull down the PSDsoft menu in the main PSDsoft window and choose **PSD Compiler**.  
The PSD Compiler window appears.

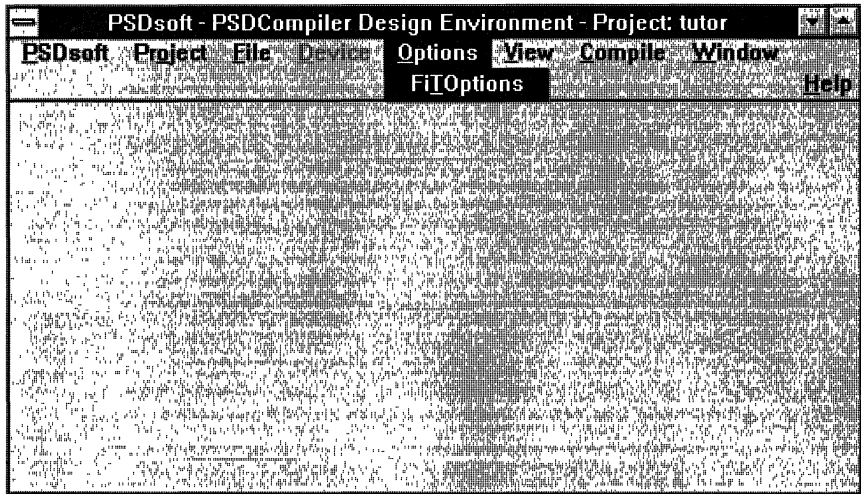


**Using the Design Example (Cont.)**

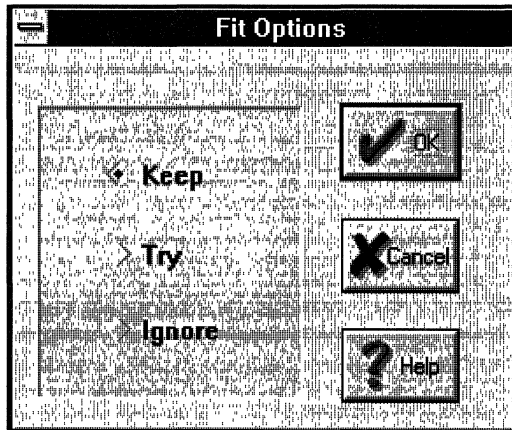
**Compiling the Design (Cont.)**

**Fitting the Design**

2. Pull down the Options menu and choose **Fit Options**.



The Fit Options window appears. The Fit Options window allows you to specify the fitting algorithm before invoking the Fitter.



## Using the Design Example (Cont.)

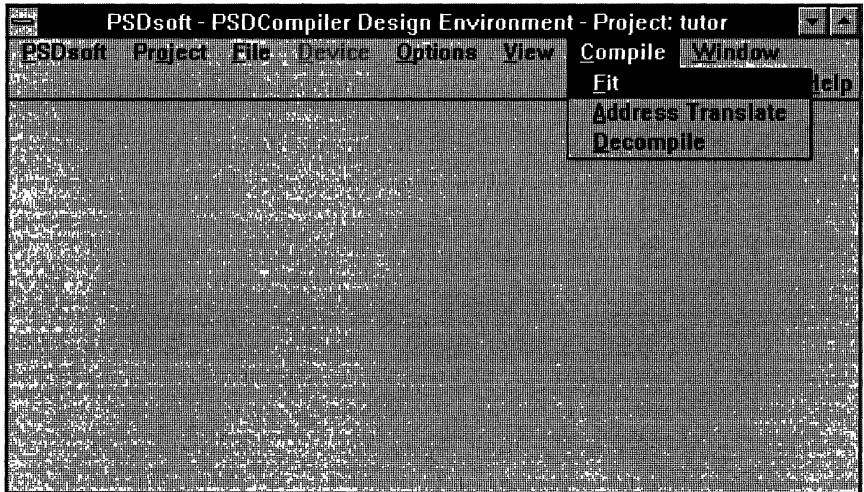
### Compiling the Design (Cont.)

#### Fitting the Design

- Click one of the three fitting options given in the Fit Options window.  
For the tutorial, you may choose Keep. The functions of the three Fitter options are listed below.

| <b>Fitter Option</b> | <b>Description</b>                                                                                 |
|----------------------|----------------------------------------------------------------------------------------------------|
| Keep                 | The Fitter should maintain the pin assignment defined by the user. The Keep option is the default. |
| Try                  | Maintain as much current pin assignment as possible.                                               |
| Ignore               | The Fitter is free to make any pin assignment and ignores the user's pin assignments.              |

- Go back to the PSDcompiler main menu.
- Pull down the Compile menu and choose **Fit**.



3

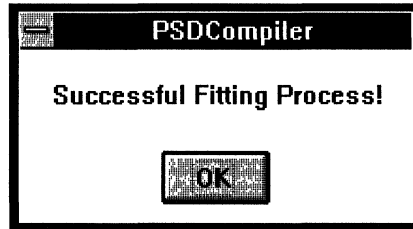


**Using the  
Design  
Example  
(Cont.)**

**Compiling the Design (Cont.)**

**Fitting the Design**

The Fitter window appears, indicating a successful or unsuccessful fitting process. Fitting must be completed successfully before invoking the Address Translator.



If the fitting is not successful, you will have to examine the tutor.feq file to see which logic function causes the fitting problem and modify the tutor.abl file accordingly.

Recompile and optimize the modified tutor.abl file again, if necessary, before you proceed with the fitting process.

6. Examine the tutor.frp report file generated by the Fitter.

The tutor.frp file shows the results of the fitting and the pin assignment of the design.

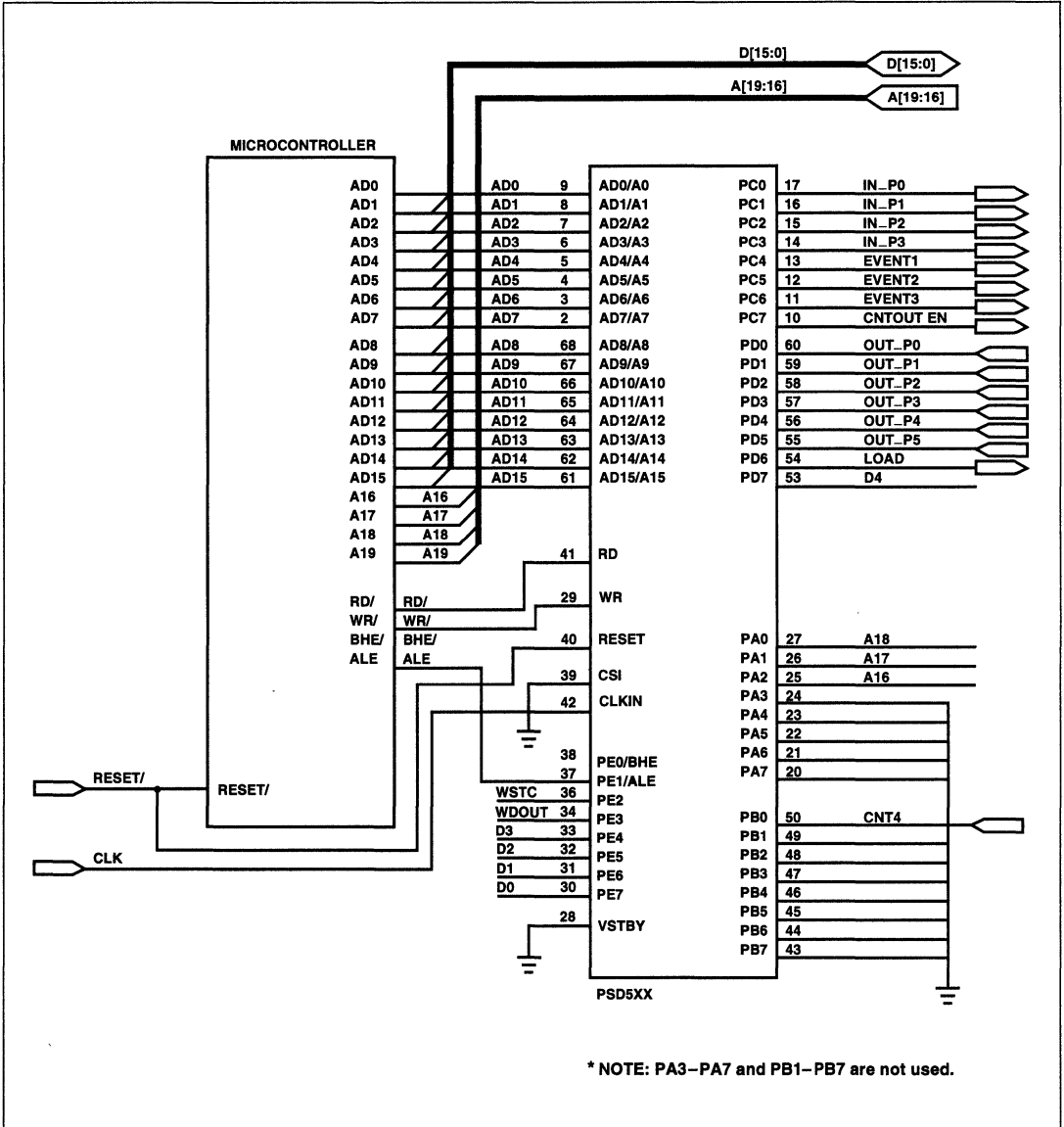
If you want a certain fitting other than the one generated, return to the tutor.abl file to specify the signal and pin assignments.

**Capturing the Schematic**

Begin the actual detailed schematic capture.

Now that the PSD5XX pins have been assigned to specific names and functions, you can create the schematic. The schematic of the tutorial design example is shown in Figure 4. The pins are assigned signal names from the tutor.frp file.

Figure 4. Logic Schematic, Tutorial Design Example



3



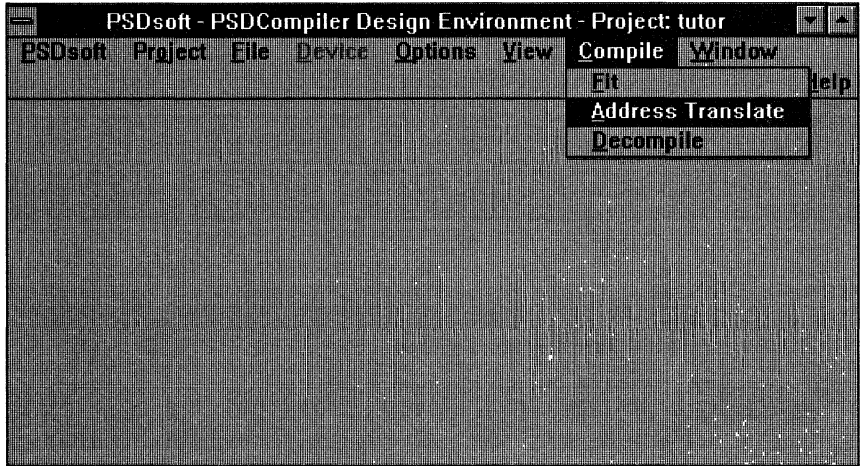
**Using the Design Example (Cont.)**

**Compiling the Design (Cont.)**

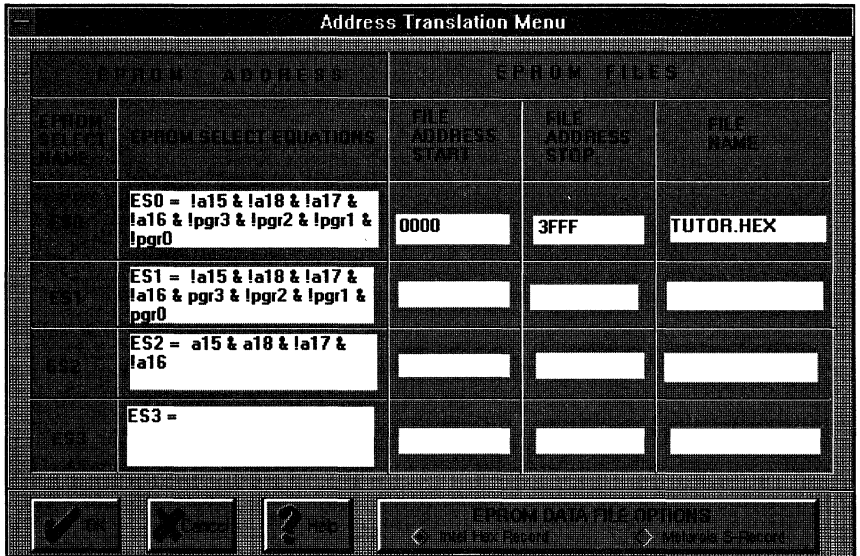
**Performing Address Translation**

The Address Translator in PSDsoft integrates your code file (EPROM file) and the PSD5XX's configuration and fusemap files.

1. Pull down the Compile menu and choose **Address Translate**.



The Address Translation menu appears.



The Address Translation menu has two sections: EPROM ADDRESS and EPROM FILES.

2. Verify that the EPROM ADDRESS section of the menu displays the four chip select equations (ES0-ES3) for the four EPROM blocks defined in the tutor.abl file.

**Using the  
Design  
Example  
(Cont.)****Compiling the Design (Cont.)****Performing Address Translation**

3. Assign to each EPROM block in the EPROM FILES section the file address range and the name of the EPROM file.

For example, the tutor.hex file consists of codes of a small test program which begins at address 2000H. The hex file's data record indicates that the starting address is 2000H and the ending address is 3FF0H, which indicates that it should be assigned to block 0. The Address Translator maps the code from the hex file at the file address specified under File Address Start to the first location of the EPROM block. Since the tutor.hex program has to be stored in EPROM location 2000H, you need to enter 0000H – 3FFFH to the File Address Start/Stop columns. As there are no codes between 0000H and 1FFFH in the tutor.hex file, the Address Translator fills the EPROM block 0 locations 0000H to 1FFFH with "FF".

Each EPROM block in the PSD503B1 has 32K bytes and you must be careful not to enter an address space of more than 32K bytes in the file start/stop columns.

4. Select the Intel Hex Record as the file type of tutor.hex.

The Address Translator accepts the input file in Intel Hex format or in Motorola S-Record format.

5. Click **OK** to Compile.

The status of the Address Translation is indicated in the window that follows.



If there are no errors, the tutor.obj file is generated. Click **OK** to exit the Address Menu. The compilation of the PSD5XX is finished.

**Using the  
Design  
Example  
(Cont.)**

**Simulating the Design**

The PSDsimulator software, WSI's version of PSDsilosIII, provides chip level simulation and design verification. PSDsilosIII can model designs using the Verilog Language. Due to the size of the PSD5XX model, a PC system requires at least 8 MB of DRAM and 8 MB of hard disk swap area (specified as virtual memory in Windows).

The PSD5XX model consists of three components (files):

- PSD5.V (the PSD5XX netlist)
- template.pfu (the ZPLD fusemap)
- template.afu (the PSD5XX configuration fuse map)

Many of the internal nodes of the PSD5XX are available for tracing. A description of the signals that can be traced by the simulator are listed in the PSD5XX.MST file.

The input files required by the PSDsilosIII simulator are generated by the Simulator Preprocessor in PSDsoft. The file you must create is the stimulus file. In the stimulus file (tutor.stl), you must use user-defined names (as in the tutor.abl file) or netlist node names (as in the psd5xx.mst file). The files generated by the Simulator Preprocessor are as follows:

- template.pfu Fusemap of the ZPLD
- template.afu Fusemap of the PSD configuration
- project.grp Group names of signals for PSDsilosIII Data Analyzer
- project.top Top level invocation with user-defined signal names.
- PSDsoft.run A PSDsilosIII command batch file which loads the PSD5.V, project.top, and project.stl before invoking the simulator.
- project.bus Same as psd5xx.mst, except default signal names are replaced with user defined names. Required in simulation display.

## Using the Design Example (Cont.)

### Simulating the Design (Cont.)

A sample of the stimulus file (tutor.stl) is shown at the end of this chapter. A typical stimulus file (microcontroller writing/reading SRAM) is shown below.

```

initial
 begin
 reset=0; //generate reset
 wr =1; rd=1; bhe = 1; //initialize control signal
 ale = 0;
 adio = 16'bz; //initialize addr/data bus
 //(16 bit) to high impedance
 a18=0; a17=0; a16=0; //set high address bits to low
 cs =0; //set PSD5XX chip select low
#300 reset = 1; //after 300ns, reset inactive
#100 ale = 1; //MCU start bus cycle, set up ale
#20 adio = 'h8476; //drive addr on adio bus 20ns after ale
#30 ale = 0; //ale goes low
#30 adio = 'h5a27; //drive data out to bus
 bhe = 0; //bhe low, word operation
#40 wr = 0; //generate 100ns write
#100 wr = 1; //end wr
 bhe = 1;
#20 adio = 16'bz; //end of bus cycle
#50 ale = 1; //start the read cycle, byte read
#20 adio = 'h8476; //drive addr
#30 ale = 0;
#30 adio = 16'bz; //MCU floats bus
#40 rd = 0; //generate rd pulse
#100 rd = 1; //end of rd cycle
 end

```

Please note that the project.top file includes all the necessary data and port declarations, and that the project.stl written by you will be appended to the project.top file.

**Using the Design Example (Cont.)**

**Simulating the Design (Cont.)**

**Simulating the EPROM in the PSD5XX**

The PSDsimulator allows code files to be loaded to the EPROM blocks for simulation. However, if code simulation is not specified, the PSD5XX model fills up the EPROM with a fixed data pattern as shown in Table 7.

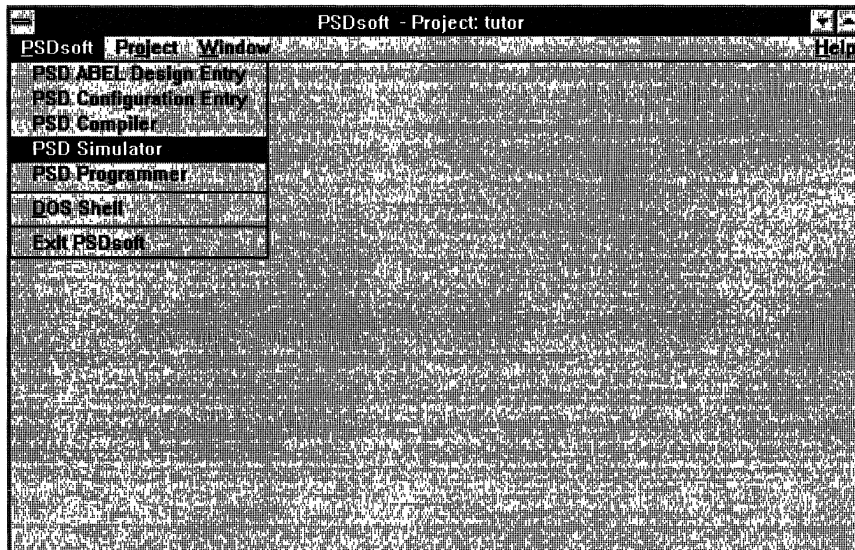
**Table 7. EPROM Data Pattern**

| <b>EPROM Block</b> | <b>Odd Byte</b> | <b>Even Byte</b> |
|--------------------|-----------------|------------------|
| block0 (ES0)       | 01h             | 23h              |
| block1 (ES1)       | 45h             | 67h              |
| block2 (ES2)       | 89h             | ABh              |
| block3 (ES3)       | CDh             | EFh              |

With the pre-filled data pattern, you can verify the EPROM address decoding logic and the even/odd byte orientation.

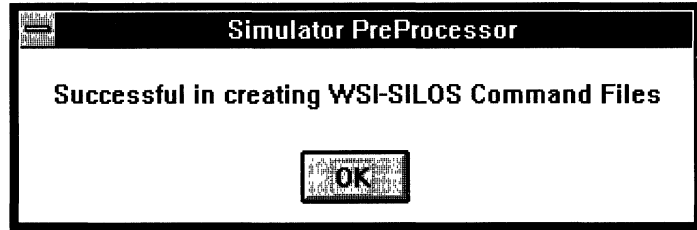
**Running the Logic Simulator**

1. Write the stimulus file, project.stl.
2. Pull down the PSDsoft menu in the main PSDsoft window and select **PSD Simulator**.



**Using the  
Design  
Example  
(Cont.)****Simulating the Design (Cont.)****Running the Logic Simulator (Cont.)**

The preprocessor software generates the necessary command files for PSDsilosIII, and the window below appears.



3. Exit from PSDsoft and double-click the PSDsilosIII icon to invoke the Simulator.  
The PSDsilosIII window appears, and displays the functions and menus available to you.



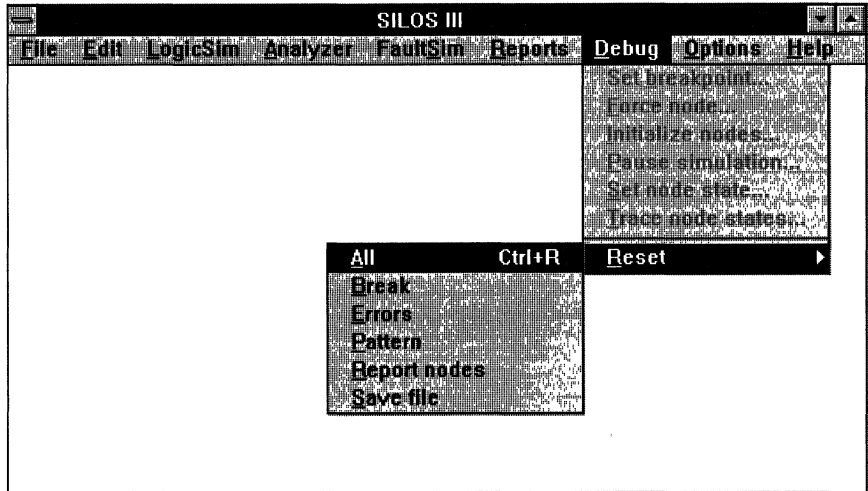


**Using the Design Example (Cont.)**

**Simulating the Design (Cont.)**

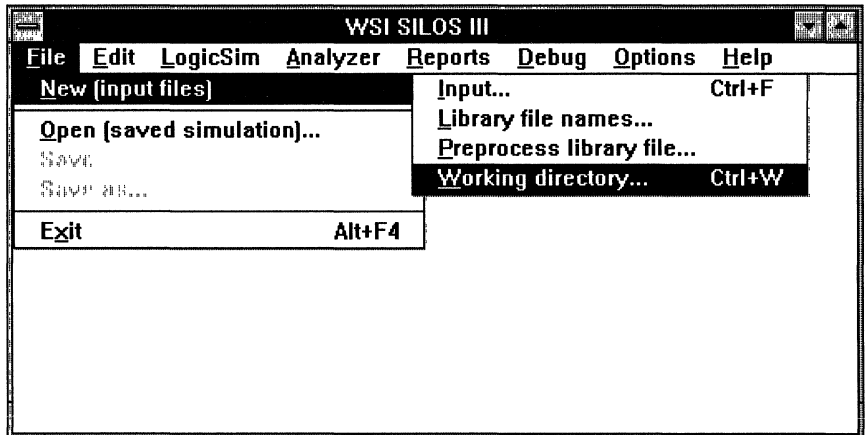
**Running the Logic Simulator (Cont.)**

4. Pull down the Debug menu and choose **Reset All**.



The system is reset before loading any new models or stimulus files.

5. Pull down the File menu and choose **New (input files)** and **Working Directory**.

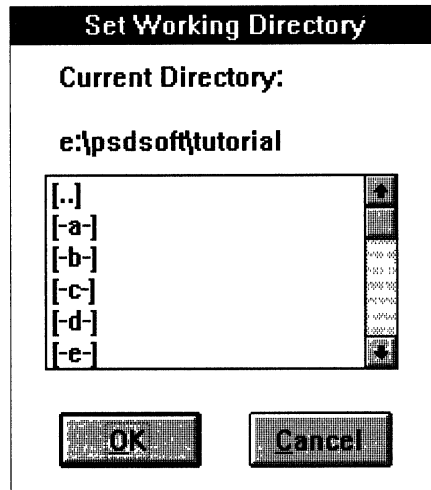


**Using the  
Design  
Example  
(Cont.)**

**Simulating the Design (Cont.)**

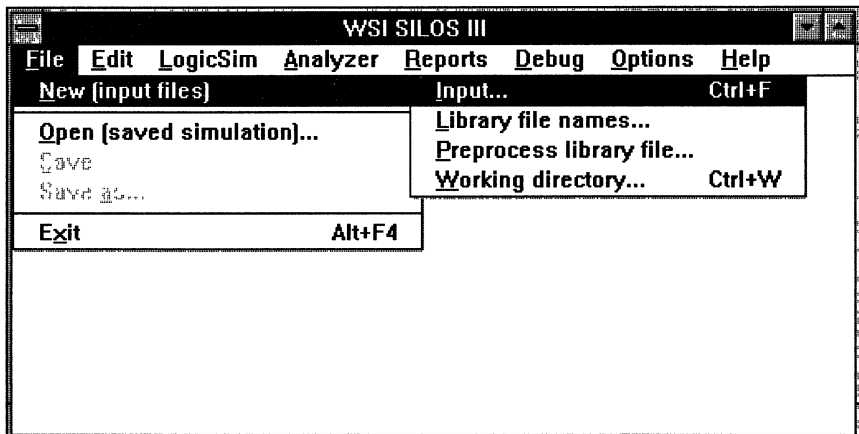
**Running the Logic Simulator (Cont.)**

The Set Working Directory window appears.



**3**

6. Click the directory where the tutorial files reside.
7. When you are satisfied with the working directory path, click OK.
8. Pull down the Files menu and choose File New (input files) Input.

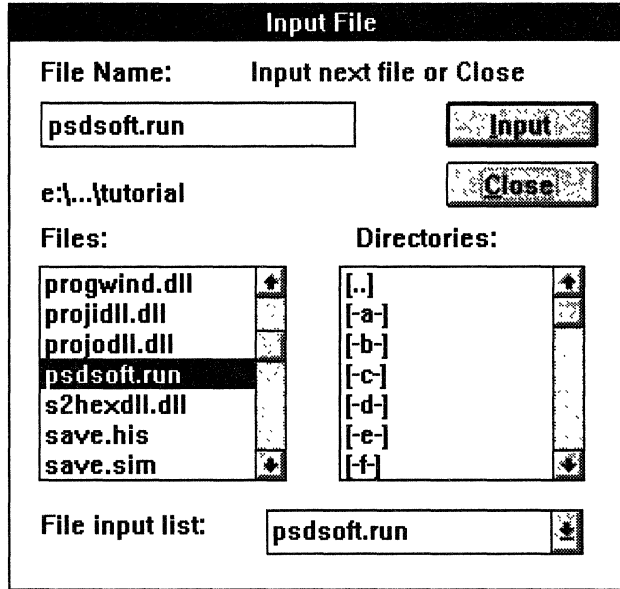


**Using the Design Example (Cont.)**

**Simulating the Design (Cont.)**

**Running the Logic Simulator (Cont.)**

The Input File window appears.

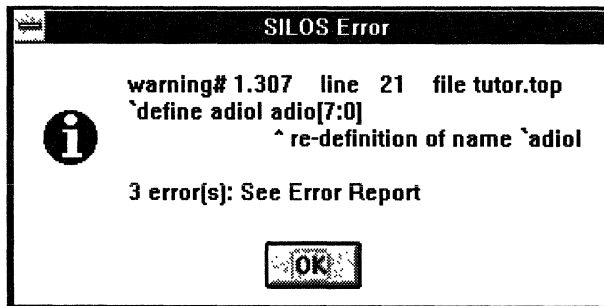


9. Click the PSDsoft.run batch file in the Files window so that PSDsoft.RUN appears in the File Name window.

10. Click **Input**.

The PSDsoft.run batch file executes, loading the tutor.stl and the PSD5XX module into PSDsilosIII.

If a syntax error is detected in the project.stl file while loading the file or running the simulator, an error message is displayed.



Every time the tutor.stl file is modified to correct the errors, PSDsilosIII has to be reset before loading the PSDsoft.run file again.

11. Click **Close**.

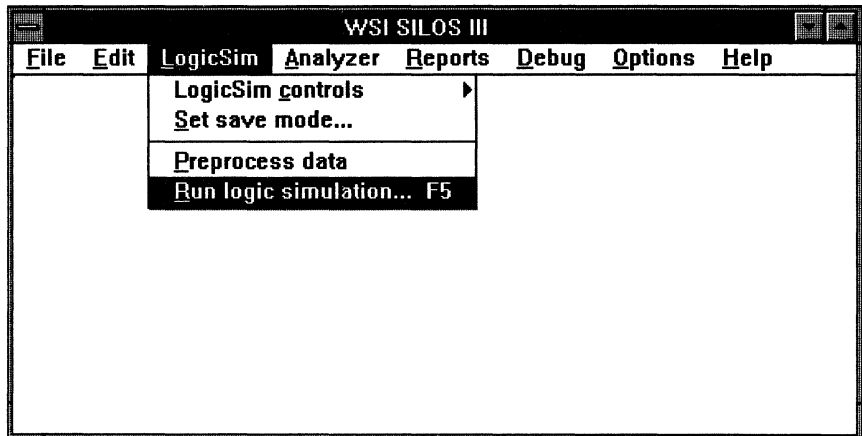
Now that the PSD5XX model and the stimulus file have been loaded, you are ready to invoke the simulator.

## Using the Design Example (Cont.)

### Simulating the Design (Cont.)

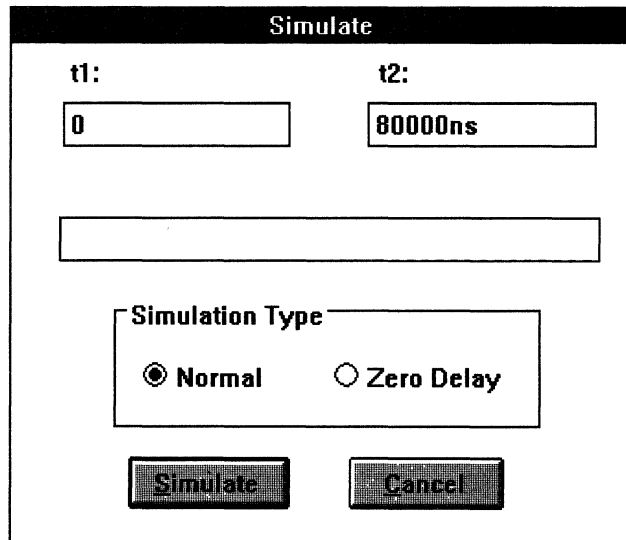
#### Running the Logic Simulator (Cont.)

12. Pull down the LogicSim menu and choose Run logic simulation.



The Simulate dialog box is displayed. The Simulate dialog box allows you to specify the simulation time range defined by t1 and t2.

3



13. Click **Simulate**

The result of the simulation is stored in the tutor.sim file.

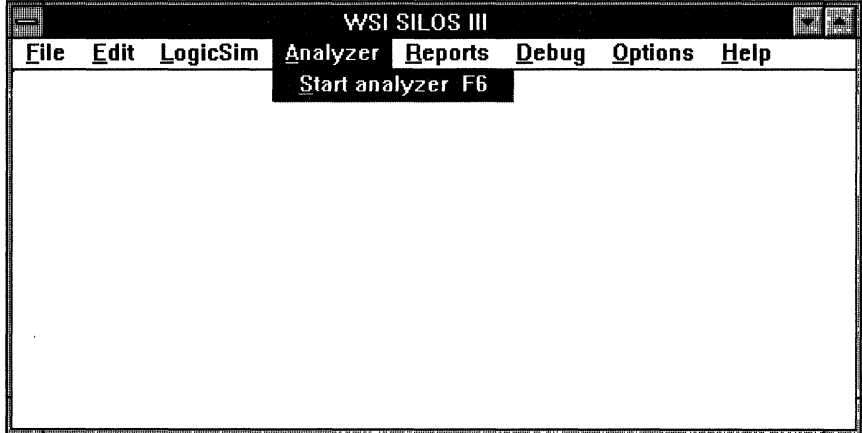
**Using the Design Example (Cont.)**

**Simulating the Design (Cont.)**

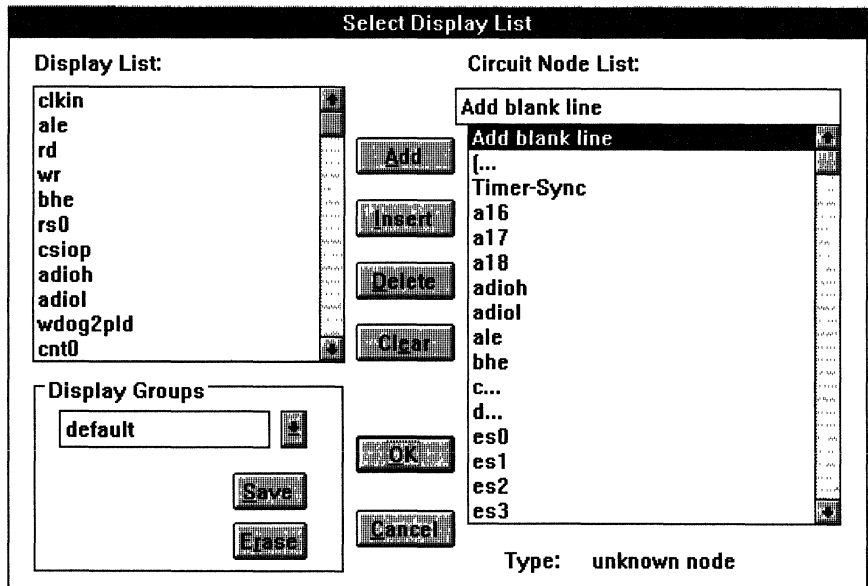
**Running the Analyzer**

Now that logic simulation is complete, the result can be displayed with the PSDsilosIII Data Analyzer (PSDsda) by performing the following steps:

1. Pull down the Analyzer menu and choose **Start analyzer**.



2. The Select Display List window appears.



The first time the Analyzer is invoked, a list of signals arranged in alphabetical order are displayed. You can re-arrange the order of the signals and save this list (click **Save** and save the list in tutor.grp) so that it can be used in the next invocation.

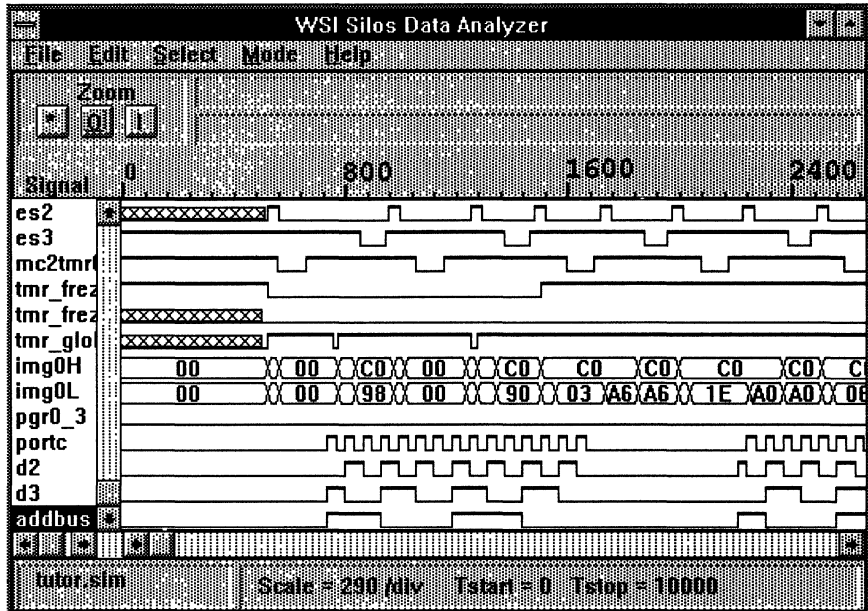
## Using the Design Example (Cont.)

### Simulating the Design (Cont.)

#### Running the Analyzer

3. Select the list of signals to be displayed by the Analyzer.
4. Click **OK** when you are satisfied with the signal list.

The PSDsilosIII Data Analyzer window is displayed.



The waveforms of the selected signals are displayed. Signals inside the signal box can be moved by dragging and the window can be moved up or down with the scroll bar. The Zoom buttons (\* = zoom full, O = zoom out, I = zoom in) and the time scale control the display window. The two time markers, T1 (the left mouse button) and T2 (the right mouse button) specify the range to be displayed.

A signal trace can be displayed in four colors on the PC monitor, depending on the nature of the signal, as follows:

- Blue** The signal is being driven as an input.
- Green** The signal is being driven as an output.
- Red** The signal is floating.
- Black** The signal is in an undefined state.

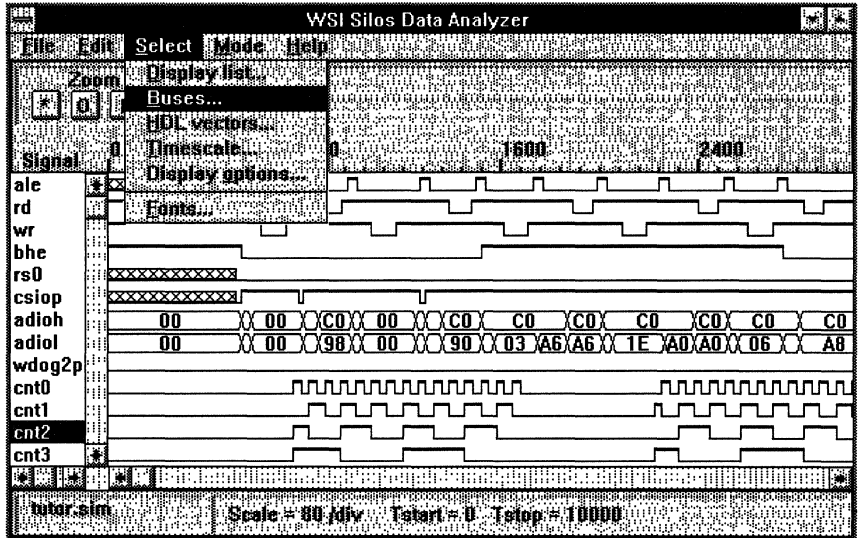
**Using the Design Example (Cont.)**

**Simulating the Design (Cont.)**

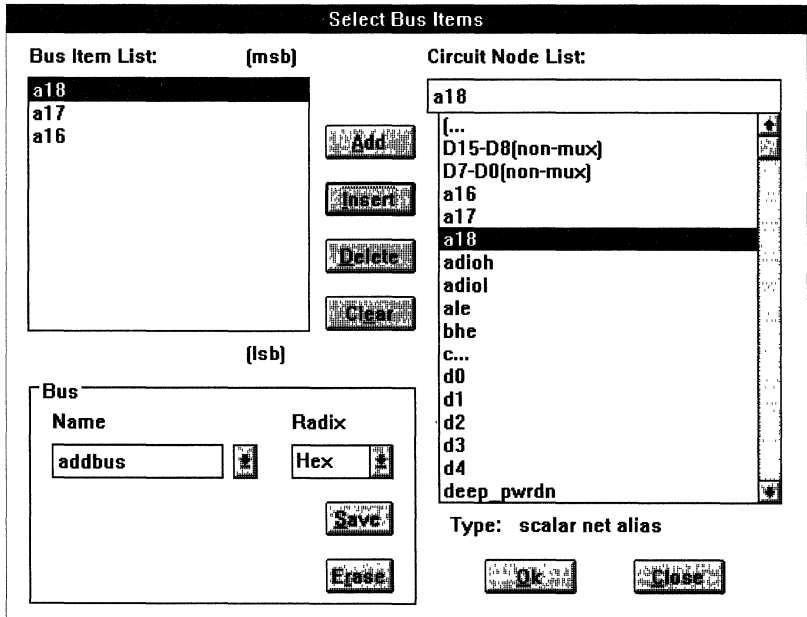
**Creating a Bus**

The PSDsda allows you to put together a group of signals into a bus, which can be displayed in the PSDsda window. To create such a bus

1. In the PSDsda window, pull down the Select menu and choose Buses.



The Select Bus Items window appears. At this point, you can create a bus to add to the PSDsda window for display during debugging. A sample bus will be created using the signals A16, A17, and A18.



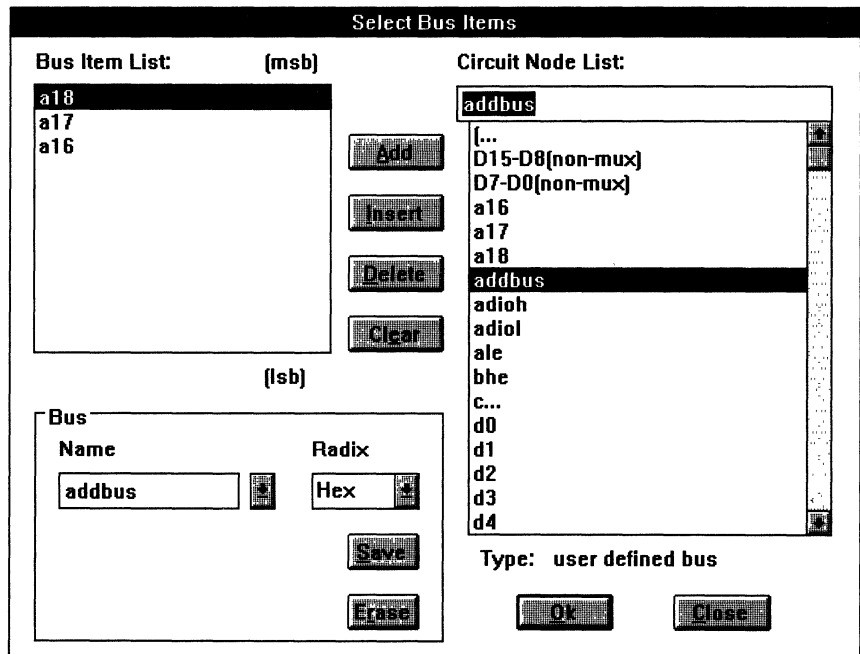
## Using the Design Example (Cont.)

### Simulating the Design (Cont.)

#### Creating a Bus

2. Click Clear to erase any signal names that appear in the Bus Item List window.
3. Enter addbus in the name window within the Bus window.  
addbus will become the name of the new item to eventually be displayed in the PSDsda window. It will consist of the combined A16, A17, and A18 values.
4. Click the A16 signal in the Circuit Node List.
5. Click Add.  
The A16 signal is added to the Bus Item List window.
6. Repeat steps 4 and 5 for A17 and A18.  
At this point, the A16, A17, and A18 signals appear in the Bus Item List window.
7. Click Save in the Bus window.

The addbus name is now added to the Circuit Node List, with a notation at the bottom of the list that it is a user-defined bus type.



8. Click OK.  
The PSDsda window reappears.

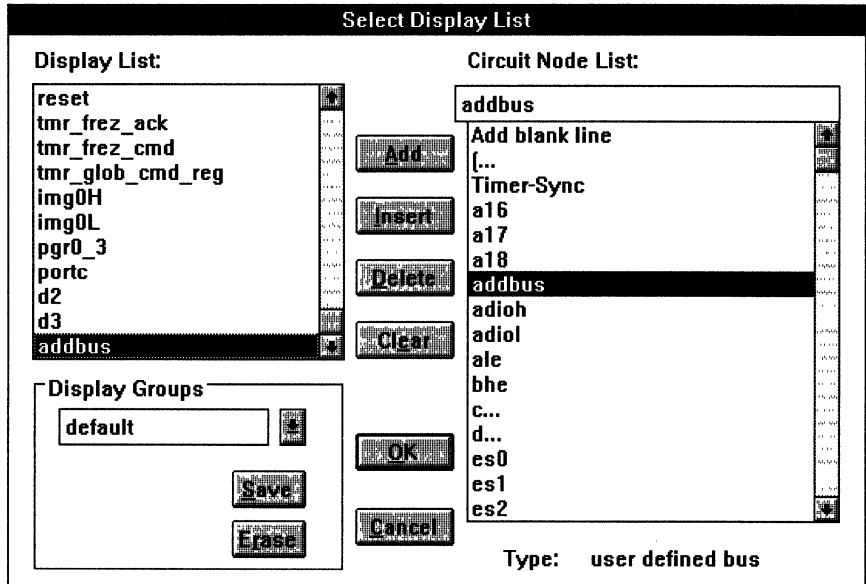


**Using the Design Example (Cont.)**

**Simulating the Design (Cont.)**

**Creating a Bus**

9. Pull down the Select menu and choose **Display list**.  
The Select Display list window appears.



The addbus name appears in the Circuit Node list because it was added as a bus in the previous steps.

10. Click **addbus** in the Circuit Node list to select it.
11. Click **Add**.

The addbus name now appears in the Display list, indicating that addbus is now available for display in the PSDsda window.

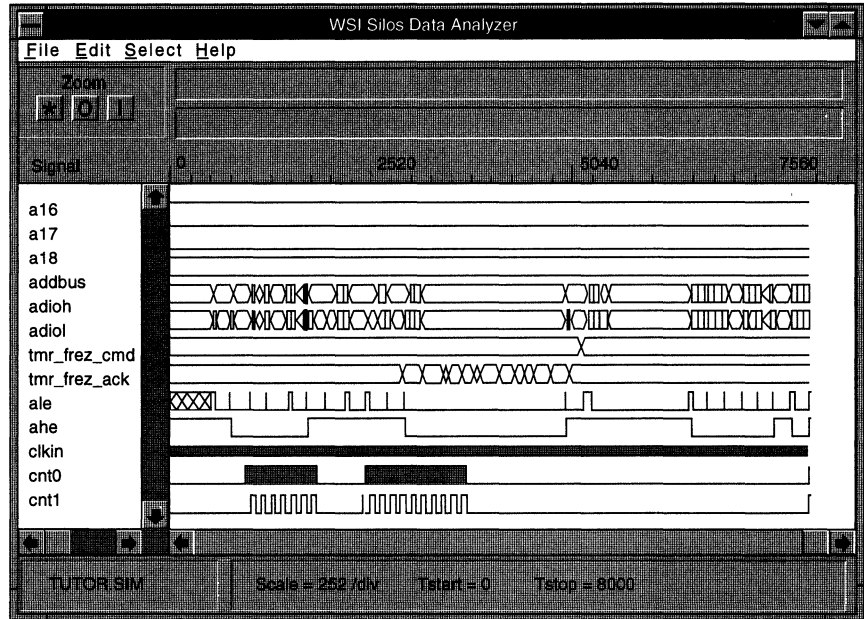
## Using the Design Example (Cont.)

### Simulating the Design (Cont.)

#### Creating a Bus

12. Click OK.

The PSDsda window reappears.



13. Scroll down the Signal name window until you see addbus.

The signal waveform corresponding to the addbus value is displayed in the window to the right.

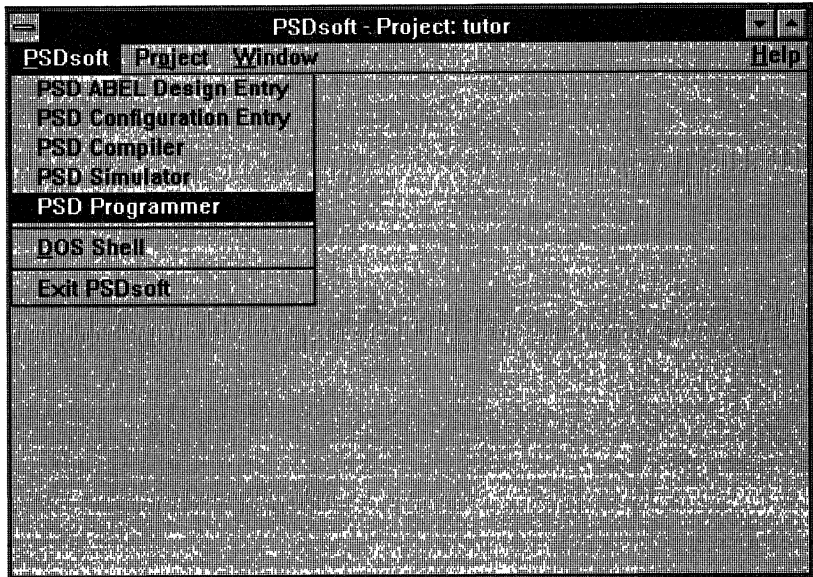
The addbus name is saved in the tutor.bus file and is available for display each time you use PSDsda.

**Using the  
Design  
Example  
(Cont.)**

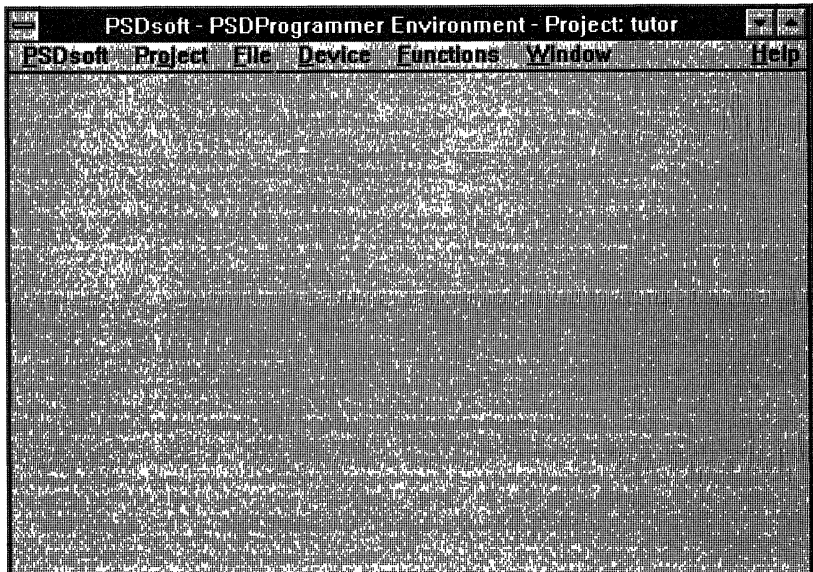
**Programming The PSD5XX**

Take the following steps to program the PSD5XX after the design has been verified through simulation. For more detailed information, refer to the PSDprogrammer chapter in this manual.

1. Pull down the PSDsoft menu in the main PSDsoft window and choose



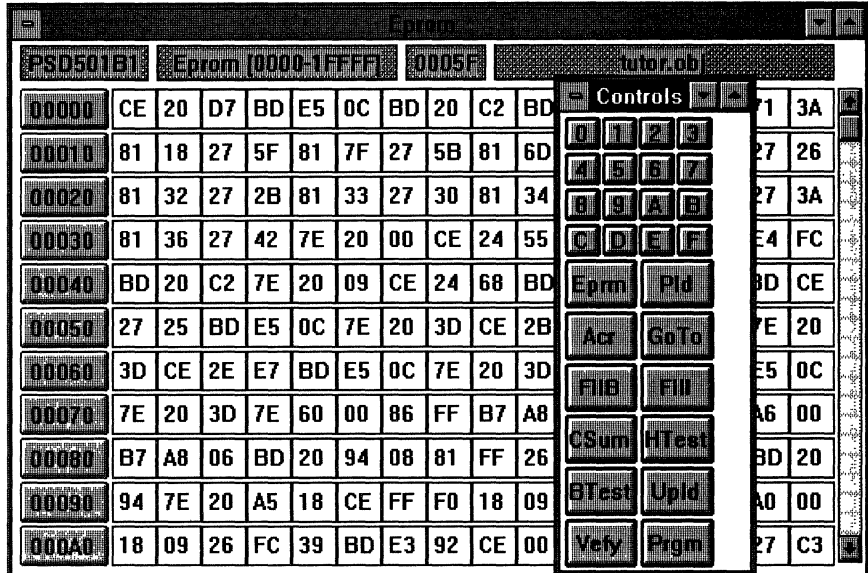
The main PSD Programmer window appears.



**Using the Design Example (Cont.)**

**Programming The PSD5XX (Cont.)**

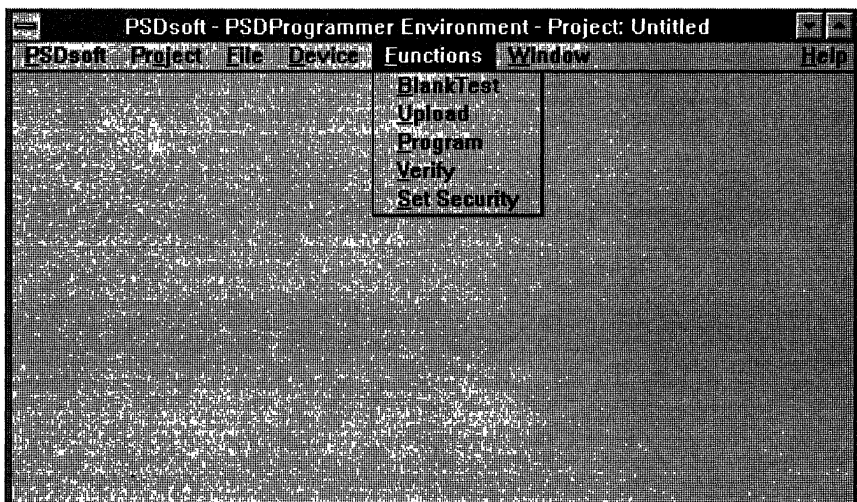
2. Pull down the File menu and choose Open.
3. Select the tutor.obj file to be loaded to the PSDpro buffer.  
The contents of the tutor.obj file are displayed. Use the Edit menu for making any code modifications.



3

There are several options available to you for working with a device. Some of these functions are shown under the Functions menu of the main PSDprogrammer window.

4. Pull down the Functions menu to see the available options.



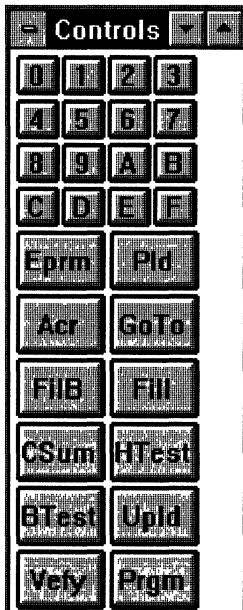
**Using the  
Design  
Example  
(Cont.)**

**Programming The PSD5XX (Cont.)**

The Functions menu provides the following options in programming the PSD5XX:

- Blank Test** Verify the device is blank.
- Upload** Upload the programmed part contents to the buffer.
- Program** Program the device.
- Verify** Verify the programmed device against the .obj file in the buffer.

The Control Panel also displays several functions, some of which are duplicated under the Functions menu in the main window.

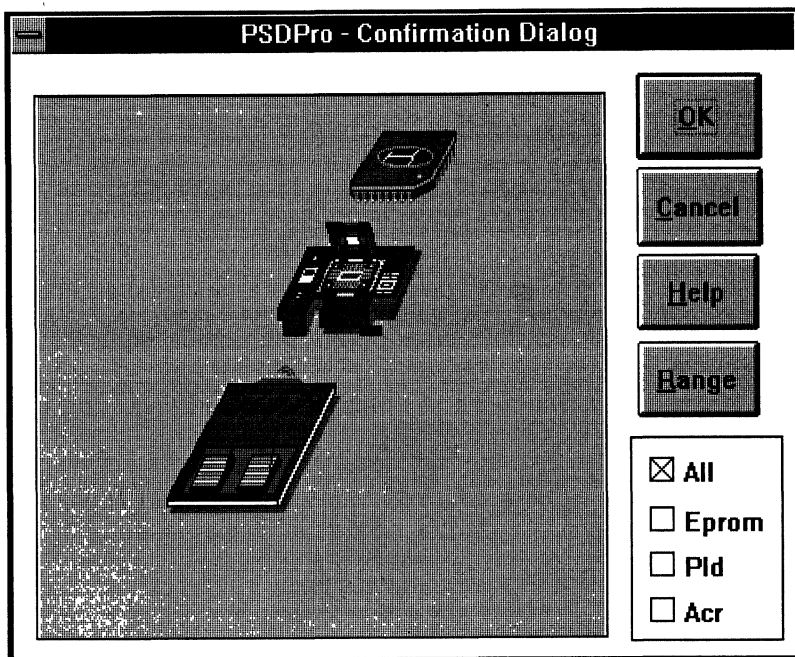


**Using the  
Design  
Example  
(Cont.)**

**To Program a Part**

1. Pull down the Functions menu in the main PSDprogrammer window and select Program or click the Prgm button of the Control Panel.

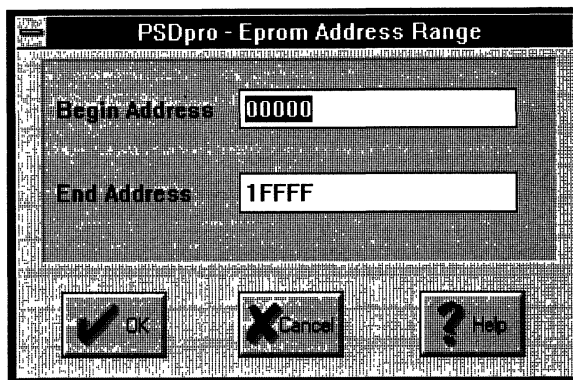
The Confirmation dialog box appears, which allows you to program the EPROM, PLD, or Acr regions of the device.



3

2. Select one or more of the boxes to indicate the regions of the device you want programmed.
3. Click Range to specify the address range within the device where the programming is to take place.

The Eprom Address Range dialog box appears.



**Using the  
Design  
Example  
(Cont.)**

**To Program a Part (Cont.)**

4. Enter the starting and ending addresses where indicated and click OK when you are satisfied with the values.

The Eprom Address Range dialog box disappears. By default, the address range is set to the beginning and ending address of the EPROM, so that the entire range of the EPROM is specified. The range can be specified only for the EPROM, not the PLD or Acr regions.

5. Click OK in the Confirmation dialog box when you are satisfied with the address range that will be programmed as well as with the functional parts of the device that will be programmed (Eprom, PLD,Acr).

A bar graph showing programming progress as well as percent complete is shown on the screen. As the programming takes place, the MagicPro® programmer checks each location as it is programmed to make sure it matches the hexadecimal file contents. If a particular location cannot be programmed properly, an error message appears. If this occurs, you must start over and program a new fully erased and functional part.

When the device has been successfully programmed, the PSDpro software verifies the device by comparing its contents with the contents of the hexadecimal file in system memory. If the device does not properly verify, an error message appears, and you must start over and program a new fully erased and functional part.

## PSDsoft Input/Output File List

| <b>File Extension</b> | <b>Description</b>                                                                                                      |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>project.ERR</i>    | – Error log file generated by various PSDsoft programs                                                                  |
| <i>project.INI</i>    | – Project information file                                                                                              |
| <i>project.ABL</i>    | – PSDabel-HDL language equation file created by you                                                                     |
| <i>project.TT1</i>    | – Non-optimized PLA file                                                                                                |
| <i>project.TT2</i>    | – Optimized PLA file                                                                                                    |
| <i>project.TT3</i>    | – Fitted PLA file                                                                                                       |
| <i>project.TMV</i>    | – Test vector file automatically generated by PSDabel compiler                                                          |
| <i>project.LST</i>    | – PSDabel compiler listing file                                                                                         |
| <i>project.SMn</i>    | – PSDabel simulation output result file generated from .TTn file                                                        |
| <i>project.EQn</i>    | – PSDabel equation files generated from .TTn file                                                                       |
| <i>project.AOP</i>    | – PSDabel options file generated automatically upon exiting PSDabel                                                     |
| <i>project.GLC</i>    | – Global configuration file                                                                                             |
| <i>project.CRP</i>    | – Global configuration report file                                                                                      |
| <i>project.HEX</i>    | – EPROM Hex object file                                                                                                 |
| <i>project.FOB</i>    | – Fuse map file in Hex format (PLD + Configuration)                                                                     |
| <i>project.OBJ</i>    | – Fuse map file in Hex format (PLD + Configuration + EPROM)                                                             |
| <i>project.AFU</i>    | – Architecture configuration fuse file for simulation use                                                               |
| <i>project.PFU</i>    | – PLD fuse file for simulation use                                                                                      |
| <i>project.EFn</i>    | – EPROM fuse file for simulation use where n = 0 through 3, each representing EPROM block ES0 through ES3, respectively |
| <i>project.FEQ</i>    | – Fitter equation file using only device reserve names                                                                  |
| <i>project.FRP</i>    | – Fitter pin assignment report file                                                                                     |
| <i>project.ASV</i>    | – Address translator save file                                                                                          |
| <i>project.ARP</i>    | – Address translator report file                                                                                        |
| <i>project.STL</i>    | – PSDsilosIII stimulus file created by you                                                                              |
| <i>project.TOP</i>    | – PSDsilosIII top level model file                                                                                      |
| TEMPLATE.*            | – Intermediate fuse files for PSDsilosIII Logic simulation use                                                          |
| <i>project.BUS</i>    | – User-defined bus names for the PSDsilosIII Data Analyzer                                                              |
| <i>project.CMM</i>    | – Current state of the simulator                                                                                        |
| <i>project.GRP</i>    | – Group names of signals for the PSDsilosIII Data Analyzer                                                              |
| <i>project.HIS</i>    | – History of any commands used for this session of PSDsilosIII                                                          |
| <i>project.LOG</i>    | – PSDsilosIII log file                                                                                                  |
| <i>project.SIM</i>    | – PSDsilosIII simulation history                                                                                        |
| <i>project.STM</i>    | – Stimulus values related to expected results                                                                           |
| <i>project.VTR</i>    | – Vector names for the PSDsilosIII Data Analyzer                                                                        |
| PSDsoft.RUN           | – For automatic loading of user netlist file and device model library                                                   |



**Tutor  
Equation File –  
tutor.eq2**

MODULE tutor  
TITLE 'tutor design example ZPLD source file

```

a15 PIN ;
a14 PIN ;
a13 PIN ;
a12 PIN ;
a11 PIN ;
a10 PIN ;
a9 PIN ;
a8 PIN ;
a1 PIN ;
a0 PIN ;
a18 PIN ;
a17 PIN ;
a16 PIN ;
pgr3 PIN ;
pgr2 PIN ;
pgr1 PIN ;
pgr0 PIN ;
bhe PIN 38;
clkkin PIN 42;
reset PIN 40;
event1 PIN ;
event2 PIN ;
event3 PIN ;
cntout_en PIN ;
load PIN ;
d4 PIN ;
d3 PIN ;
d2 PIN ;
d1 PIN ;
d0 PIN ;
in_p0 PIN 17;
in_p1 PIN 16;
in_p2 PIN 15;
in_p3 PIN 14;
wstc PIN ;
cnt4 PIN ;
out_p0 PIN 60;
out_p1 PIN 59;
out_p2 PIN 58;
out_p3 PIN 57;
out_p4 PIN 56;
out_p5 PIN 55;
wdout PIN ;
wdog2pld PIN ;
csiop PIN ;
rs0 PIN ;
es0 PIN ;
es1 PIN ;
es2 PIN ;
mc2tmr0 PIN ;
cnt3 PIN ;
cnt2 PIN ;
cnt1 PIN ;
cnt0 PIN ;

```

## Tutor Equation File – tutor.eq2

### EQUATIONS

```

csiop = (a15 & a14 & !a13 & !a12 & !a11 & !a10 & !a9 & !a8 &
 !a18 & !a17 & !a16);

rs0 = (a15 & !a14 & !a13 & !a12 & !a11 & !a18 & !a17 & !a16);
es0 = (!a15 & !a18 & !a17 & !a16 & !pgr3 & !pgr2 & !pgr1 & !pgr0);
es1 = (!a15 & !a18 & !a17 & !a16 & pgr3 & !pgr2 & !pgr1 & pgr0);
es2 = (a15 & a18 & !a17 & !a16);
wstc = (!cnt4.FB & !cnt3.FB & !cnt2.FB & !cnt1.FB & !cnt0.FB);

cnt4 := (!load & !wstc & !cnt4.FB & !cnt3.FB & !cnt2.FB & !cnt1.FB & !cnt0.FB
 # load & d4
 # !load & !wstc & cnt4.FB & cnt0.FB
 # !load & !wstc & cnt4.FB & cnt1.FB
 # !load & !wstc & cnt4.FB & cnt2.FB
 # !load & !wstc & cnt4.FB & cnt3.FB);

cnt4.C = (clk);
cnt4.RE = (reset);
cnt4.OE = (!cntout_en);

cnt3 := (!load & !wstc & !cnt3.FB & !cnt2.FB & !cnt1.FB &
 !cnt0.FB
 # !load & !wstc & cnt3.FB & cnt0.FB
 # !load & !wstc & cnt3.FB & cnt1.FB
 # !load & !wstc & cnt3.FB & cnt2.FB
 # load & d3);

cnt3.C = (clk);
cnt3.RE = (reset);

cnt2 := (!load & !wstc & !cnt2.FB & !cnt1.FB & !cnt0.FB
 # !load & !wstc & cnt2.FB & cnt0.FB
 # !load & !wstc & cnt2.FB & cnt1.FB
 # load & d2);

cnt2.C = (clk);
cnt2.RE = (reset);

cnt1 := (!load & !wstc & !cnt1.FB & !cnt0.FB
 # !load & !wstc & cnt1.FB & cnt0.FB
 # load & d1);

cnt1.C = (clk);
cnt1.RE = (reset);

cnt0 := (!load & !wstc & !cnt0.FB
 # load & d0);

cnt0.C = (clk);
cnt0.RE = (reset);

mc2tmr0 = (event1 & !event2
 # !event3);

wdout = (!wdog2pld);

END

```

**PSDXXX.mst  
Files**

**PSD5B1.mst File**

The following 159 signals are available to you for PSDsilosIII simulation of a PSD5XX device.

- |     |                 |                                                           |
|-----|-----------------|-----------------------------------------------------------|
| 1.  | datah           | ;Upper byte of the 16-bit data bus in non-mux mode only   |
| 2.  | datal           | ;Lower byte of the 8/16-bit data bus in non-mux mode only |
| 3.  | intr_prior_stat | ;Interrupt Priority Status register                       |
| 4.  | pe_mc0          | ;Port E macrocell output-0                                |
| 5.  | pe_mc1          | ;Port E macrocell output-1                                |
| 6.  | pe_mc2          | ;Port E macrocell output-2                                |
| 7.  | pe_mc3          | ;Port E macrocell output-3                                |
| 8.  | pe_mc4          | ;Port E macrocell output-4                                |
| 9.  | pe_mc5          | ;Port E macrocell output-5                                |
| 10. | pe_mc6          | ;Port E macrocell output-6                                |
| 11. | pe_mc7          | ;Port E macrocell output-7                                |
| 12. | pb_mc0          | ;Port B macrocell output-0                                |
| 13. | pb_mc1          | ;Port B macrocell output-1                                |
| 14. | pb_mc2          | ;Port B macrocell output-2                                |
| 15. | pb_mc3          | ;Port B macrocell output-3                                |
| 16. | pb_mc4          | ;Port B macrocell output-4                                |
| 17. | pb_mc5          | ;Port B macrocell output-5                                |
| 18. | pb_mc6          | ;Port B macrocell output-6                                |
| 19. | pb_mc7          | ;Port B macrocell output-7                                |
| 20. | pa_mc0          | ;Port A macrocell output-0                                |
| 21. | pa_mc1          | ;Port A macrocell output-1                                |
| 22. | pa_mc2          | ;Port A macrocell output-2                                |
| 23. | pa_mc3          | ;Port A macrocell output-3                                |
| 24. | pa_mc4          | ;Port A macrocell output-4                                |
| 25. | pa_mc5          | ;Port A macrocell output-5                                |
| 26. | pa_mc6          | ;Port A macrocell output-6                                |
| 27. | pa_mc7          | ;Port A macrocell output-7                                |
| 28. | deep_pwrdsn     | ;Deep Sleep mode of PSD                                   |
| 29. | pwrdsn          | ;Standby mode of PSD                                      |
| 30. | psen_to_ram_en  | ;SRCODE bit in VM register                                |
| 31. | periph_mode     | ;Peripheral I/O mode                                      |
| 32. | pmmr1           | ;Power Management mode register-1                         |
| 33. | pmmr0           | ;Power Management mode register-0                         |
| 34. | wdog2pld        | ;WatchDog output routed as a PPLD input                   |
| 35. | intr2pld        | ;Interrupt output routed as a PPLD input                  |
| 36. | mc2tmr0         | ;Counter/Timer-0 PPLD macrocell output                    |
| 37. | mc2tmr1         | ;Counter/Timer-1 PPLD macrocell output                    |
| 38. | mc2tmr2         | ;Counter/Timer-2 PPLD macrocell output                    |
| 39. | mc2tmr3         | ;Counter/Timer-3 PPLD macrocell output                    |
| 40. | mc2int6         | ;Interrupt-6 PPLD macrocell output                        |
| 41. | mc2int7         | ;Interrupt-7 PPLD macrocell output                        |
| 42. | pt2int4         | ;Interrupt-4 PPLD product term output                     |
| 43. | pt2int5         | ;Interrupt-5 PPLD product term output                     |
| 44. | dout_b          | ;Port B data out register                                 |
| 45. | dirff_b         | ;Port B direction register                                |

**PSDXXX.mst  
Files****PSD5B1.mst File (Cont.)**

|     |                  |                                                          |
|-----|------------------|----------------------------------------------------------|
| 46. | ctrl_b           | ;Port B control register                                 |
| 47. | spec_b           | ;Port B special function register                        |
| 48. | ctrl_e           | ;Port E control register                                 |
| 49. | dout_e           | ;Port E data register                                    |
| 50. | dirff_e          | ;Port E direction register                               |
| 51. | spec_e           | ;Port E special function register                        |
| 52. | intr_level       | ;Interrupt Edge/Level definition register                |
| 53. | intr_msk         | ;Interrupt Mask register                                 |
| 54. | intr_req         | ;Interrupt request latch register                        |
| 55. | tmr_wait_cnt     | ;DLCY(delay) value of Counter/Timers<br>clock input      |
| 56. | tmr_glob_cmd_reg | ;Global command register of Counter/Timers               |
| 57. | tmr_frez_cmd     | ;Freeze Command register of Coueter/Timers               |
| 58. | tmr_frez_ack     | ;Freeze Acknowledge status register of<br>Counter/Timers |
| 59. | tmr_soft_ld      | ;Software load command register of<br>Counter/Timers     |
| 60. | cmd3             | ;Counter/Timer-3 command register                        |
| 61. | cmd2             | ;Counter/Timer-2 command register                        |
| 62. | cmd1             | ;Counter/Timer-1 command register                        |
| 63. | cmd0             | ;Counter/Timer-0 command register                        |
| 64. | ctrl_d           | ;Port D control register                                 |
| 65. | dout_d           | ;Port D data register                                    |
| 66. | dirff_d          | ;Port D direction register                               |
| 67. | opn_drn_d        | ;Port D Open Drain/CMOS definition register              |
| 68. | dout_c           | ;Port C data register                                    |
| 69. | ctrl_c           | ;Port C control register                                 |
| 70. | dirff_c          | ;Port C direction register                               |
| 71. | opn_drn_c        | ;Port C Open Drain/CMOS definition register              |
| 72. | pgr0_3           | ;Page Registers 0 through 3                              |
| 73. | psel1            | ;Peripheral I/O mode select product term 2               |
| 74. | psel0            | ;Peripheral I/O mode select product term 1               |
| 75. | csiop            | ;Chip Select I/O ports                                   |
| 76. | es3              | ;EPROM Chip select for block-3                           |
| 77. | es2              | ;EPROM Chip select for block-2                           |
| 78. | es1              | ;EPROM Chip select for block-1                           |
| 79. | es0              | ;EPROM Chip select for block-0                           |
| 80. | rs0              | ;PSD SRAM Chip Select                                    |
| 81. | pb0              | ;Port B pin-0                                            |
| 82. | pb1              | ;Port B pin-1                                            |
| 83. | pb2              | ;Port B pin-2                                            |
| 84. | pb3              | ;Port B pin-3                                            |
| 85. | pb4              | ;Port B pin-4                                            |
| 86. | pb5              | ;Port B pin-5                                            |
| 87. | pb6              | ;Port B pin-6                                            |
| 88. | pb7              | ;Port B pin-7                                            |
| 89. | pa0              | ;Port A pin-0                                            |
| 90. | pa1              | ;Port A pin-1                                            |

**PSDXXX.mst  
Files**

**PSD5B1.mst File (Cont.)**

|      |         |                                           |
|------|---------|-------------------------------------------|
| 91.  | pa2     | ;Port A pin-2                             |
| 92.  | pa3     | ;Port A pin-3                             |
| 93.  | pa4     | ;Port A pin-4                             |
| 94.  | pa5     | ;Port A pin-5                             |
| 95.  | pa6     | ;Port A pin-6                             |
| 96.  | pa7     | ;Port A pin-7                             |
| 97.  | pe2     | ;Port E pin-2                             |
| 98.  | pe3     | ;Port E pin-3                             |
| 99.  | pe4     | ;Port E pin-4                             |
| 100. | pe5     | ;Port E pin-5                             |
| 101. | pe6     | ;Port E pin-6                             |
| 102. | pe7     | ;Port E pin-7                             |
| 103. | pd0     | ;Port D pin-0                             |
| 104. | pd1     | ;Port D pin-1                             |
| 105. | pd2     | ;Port D pin-2                             |
| 106. | pd3     | ;Port D pin-3                             |
| 107. | pd4     | ;Port D pin-4                             |
| 108. | pd5     | ;Port D pin-5                             |
| 109. | pd6     | ;Port D pin-6                             |
| 110. | pd7     | ;Port D pin-7                             |
| 111. | pc0     | ;Port C pin-0                             |
| 112. | pc1     | ;Port C pin-1                             |
| 113. | pc2     | ;Port C pin-2                             |
| 114. | pc3     | ;Port C pin-3                             |
| 115. | pc4     | ;Port C pin-4                             |
| 116. | pc5     | ;Port C pin-5                             |
| 117. | pc6     | ;Port C pin-6                             |
| 118. | pc7     | ;Port C pin-7                             |
| 119. | spec_a  | ;Port A special function register         |
| 120. | dirff_a | ;Port A direction register                |
| 121. | ctrl_a  | ;Port A Control register                  |
| 122. | dout_a  | ;Port A data register                     |
| 123. | cntr3H  | ;Counter/Timer-3 high byte register       |
| 124. | cntr3L  | ;Counter/Timer-3 low byte register        |
| 125. | cntr2H  | ;Counter/Timer-2 high byte register       |
| 126. | cntr2L  | ;Counter/Timer-2 low byte register        |
| 127. | cntr1H  | ;Counter/Timer-1 high byte register       |
| 128. | cntr1L  | ;Counter/Timer-1 low byte register        |
| 129. | cntr0H  | ;Counter/Timer-0 high byte register       |
| 130. | cntr0L  | ;Counter/Timer-0 low byte register        |
| 131. | img3H   | ;Counter/Timer-3 Image high byte register |
| 132. | img3L   | ;Counter/Timer-3 Image low byte register  |
| 133. | img2H   | ;Counter/Timer-2 Image high byte register |
| 134. | img2L   | ;Counter/Timer-2 Image low byte register  |
| 135. | img1H   | ;Counter/Timer-1 Image high byte register |

**PSDXXX.mst  
Files****PSD5B1.mst File (Cont.)**

|      |           |                                               |
|------|-----------|-----------------------------------------------|
| 136. | img1L     | ;Counter/Timer-1 Image low byte register      |
| 137. | img0H     | ;Counter/Timer-0 Image high byte register     |
| 138. | img0L     | ;Counter/Timer-0 Image low byte register      |
| 139. | portb     | ;Port B register                              |
| 140. | clkIn     | ;PSD input Clock                              |
| 141. | reset     | ;PSD input reset                              |
| 142. | csi       | ;PSD Chip Select                              |
| 143. | pe1       | ;Port E pin-1 (ALE etc.,)                     |
| 144. | pe0       | ;Port E pin-0 (PSEN, BHE etc.,)               |
| 145. | wr        | ;PSD write signal                             |
| 146. | rd        | ;PSD read signal                              |
| 147. | portd     | ;Port D register                              |
| 148. | portc     | ;Port C register                              |
| 149. | adiOh     | ;Address/Data bus high byte                   |
| 150. | adiOl     | ;Address/Data bus low byte                    |
| 151. | porta     | ;port A register                              |
| 152. | timerout0 | ;Counter/Timer-0 output (only when used)      |
| 153. | timerout1 | ;Counter/Timer-1 output (only when used)      |
| 154. | timerout2 | ;Counter/Timer-2 output (only when used)      |
| 155. | timerout3 | ;Counter/Timer-3 output (only when used)      |
| 156. | pgr0      | ;Page Register bit 0                          |
| 157. | pgr1      | ;Page Register bit 1                          |
| 158. | pgr2      | ;Page Register bit 2                          |
| 159. | pgr3      | ;Page Register bit 3                          |
| 160. | timer_clk | ;The actual clock input to the Counter/Timers |

**PSDXXX.mst  
Files**

**PSD4A1.mst File**

The following 100 signals are available to you for PSDsilosIII simulation of a PSD4XXA1 device.

1. datah ;Upper byte of the 16-bit data bus in non-mux mode only
2. datal ;Lower byte of the 8/16-bit data bus in non-mux mode only
3. pb\_mc0 ;Port B macrocell output-0
4. pb\_mc1 ;Port B macrocell output-1
5. pb\_mc2 ;Port B macrocell output-2
6. pb\_mc3 ;Port B macrocell output-3
7. pb\_mc4 ;Port B macrocell output-4
8. pb\_mc5 ;Port B macrocell output-5
9. pb\_mc6 ;Port B macrocell output-6
10. pb\_mc7 ;Port B macrocell output-7
11. deep\_pwrndn ;Deep Sleep mode of PSD
12. pwrndn ;Standby mode of PSD
13. psen\_to\_ram\_en ;SRCODE bit in VM register
14. periph\_mode ;Peripheral I/O mode
15. pmmr1 ;Power Management mode register-1
16. pmmr0 ;Power Management mode register-0
17. dout\_b ;Port B data out register
18. dirff\_b ;Port B direction register
19. ctrl\_b ;Port B control register
20. spec\_b ;Port B special function register
21. ctrl\_e ;Port E control register
22. dout\_e ;Port E data register
23. dirff\_e ;Port E direction register
24. spec\_e ;Port E special function register
25. ctrl\_d ;Port D control register
26. dout\_d ;Port D data register
27. dirff\_d ;Port D direction register
28. opn\_drn\_d ;Port D Open Drain/CMOS definition register
29. dout\_c ;Port C data register
30. ctrl\_c ;Port C control register
31. dirff\_c ;Port C direction register
32. opn\_drn\_c ;Port C Open Drain/CMOS definition register
33. pgr0\_3 ;Page Registers 0 through 3
34. psel1 ;Peripheral I/O mode select product term 2
35. psel0 ;Peripheral I/O mode select product term 1
36. csiop ;Chip Select I/O ports
37. es3 ;EPROM Chip select for block-3
38. es2 ;EPROM Chip select for block-2
39. es1 ;EPROM Chip select for block-1
40. es0 ;EPROM Chip select for block-0
41. rs0 ;PSD SRAM Chip Select
42. pb0 ;Port B pin-0
43. pb1 ;Port B pin-1
44. pb2 ;Port B pin-2
45. pb3 ;Port B pin-3
46. pb4 ;Port B pin-4
47. pb5 ;Port B pin-5
48. pb6 ;Port B pin-6

**PSDXXX.mst  
Files****PSD4A1.mst File (Cont.)**

|      |         |                                   |
|------|---------|-----------------------------------|
| 49.  | pb7     | ;Port B pin-7                     |
| 50.  | pa0     | ;Port A pin-0                     |
| 51.  | pa1     | ;Port A pin-1                     |
| 52.  | pa2     | ;Port A pin-2                     |
| 53.  | pa3     | ;Port A pin-3                     |
| 54.  | pa4     | ;Port A pin-4                     |
| 55.  | pa5     | ;Port A pin-5                     |
| 56.  | pa6     | ;Port A pin-6                     |
| 57.  | pa7     | ;Port A pin-7                     |
| 58.  | pe2     | ;Port E pin-2                     |
| 59.  | pe3     | ;Port E pin-3                     |
| 60.  | pe4     | ;Port E pin-4                     |
| 61.  | pe5     | ;Port E pin-5                     |
| 62.  | pe6     | ;Port E pin-6                     |
| 63.  | pe7     | ;Port E pin-7                     |
| 64.  | pd0     | ;Port D pin-0                     |
| 65.  | pd1     | ;Port D pin-1                     |
| 66.  | pd2     | ;Port D pin-2                     |
| 67.  | pd3     | ;Port D pin-3                     |
| 68.  | pd4     | ;Port D pin-4                     |
| 69.  | pd5     | ;Port D pin-5                     |
| 70.  | pd6     | ;Port D pin-6                     |
| 71.  | pd7     | ;Port D pin-7                     |
| 72.  | pc0     | ;Port C pin-0                     |
| 73.  | pc1     | ;Port C pin-1                     |
| 74.  | pc2     | ;Port C pin-2                     |
| 75.  | pc3     | ;Port C pin-3                     |
| 76.  | pc4     | ;Port C pin-4                     |
| 77.  | pc5     | ;Port C pin-5                     |
| 78.  | pc6     | ;Port C pin-6                     |
| 79.  | pc7     | ;Port C pin-7                     |
| 80.  | spec_a  | ;Port A special function register |
| 81.  | dirff_a | ;Port A direction register        |
| 82.  | ctrl_a  | ;Port A Control register          |
| 83.  | dout_a  | ;Port A data register             |
| 84.  | portb   | ;Port B register                  |
| 85.  | clkin   | ;PSD input Clock                  |
| 86.  | reset   | ;PSD input reset                  |
| 87.  | csi     | ;PSD Chip Select                  |
| 88.  | pe1     | ;Port E pin-1 (ALE etc.)          |
| 89.  | pe0     | ;Port E pin-0 (PSEN, BHE etc.)    |
| 90.  | wr      | ;PSD write signal                 |
| 91.  | rd      | ;PSD read signal                  |
| 92.  | portd   | ;Port D register                  |
| 93.  | portc   | ;Port C register                  |
| 94.  | adioh   | ;Address/Data bus high byte       |
| 95.  | adiol   | ;Address/Data bus low byte        |
| 96.  | porta   | ;port A register                  |
| 97.  | pgr0    | ;Page Register bit 0              |
| 98.  | pgr1    | ;Page Register bit 1              |
| 99.  | pgr2    | ;Page Register bit 2              |
| 100. | pgr3    | ;Page Register bit 3              |



## PSDXXX.mst Files

### PSD4A2.mst File

The following 116 signals are available to you for PSDsilosIII simulation of a PSD4XXA2 device.

|     |                |                                                           |
|-----|----------------|-----------------------------------------------------------|
| 1.  | datah          | ;Upper byte of the 16-bit data bus in non-mux mode only   |
| 2.  | datal          | ;Lower byte of the 8/16-bit data bus in non-mux mode only |
| 3.  | pe_mc0         | ;Port E macrocell output-0                                |
| 4.  | pe_mc1         | ;Port E macrocell output-1                                |
| 5.  | pe_mc2         | ;Port E macrocell output-2                                |
| 6.  | pe_mc3         | ;Port E macrocell output-3                                |
| 7.  | pe_mc4         | ;Port E macrocell output-4                                |
| 8.  | pe_mc5         | ;Port E macrocell output-5                                |
| 9.  | pe_mc6         | ;Port E macrocell output-6                                |
| 10. | pe_mc7         | ;Port E macrocell output-7                                |
| 11. | pb_mc0         | ;Port B macrocell output-0                                |
| 12. | pb_mc1         | ;Port B macrocell output-1                                |
| 13. | pb_mc2         | ;Port B macrocell output-2                                |
| 14. | pb_mc3         | ;Port B macrocell output-3                                |
| 15. | pb_mc4         | ;Port B macrocell output-4                                |
| 16. | pb_mc5         | ;Port B macrocell output-5                                |
| 17. | pb_mc6         | ;Port B macrocell output-6                                |
| 18. | pb_mc7         | ;Port B macrocell output-7                                |
| 19. | pa_mc0         | ;Port A macrocell output-0                                |
| 20. | pa_mc1         | ;Port A macrocell output-1                                |
| 21. | pa_mc2         | ;Port A macrocell output-2                                |
| 22. | pa_mc3         | ;Port A macrocell output-3                                |
| 23. | pa_mc4         | ;Port A macrocell output-4                                |
| 24. | pa_mc5         | ;Port A macrocell output-5                                |
| 25. | pa_mc6         | ;Port A macrocell output-6                                |
| 26. | pa_mc7         | ;Port A macrocell output-7                                |
| 27. | deep_pwrn      | ;Deep Sleep mode of PSD                                   |
| 28. | pwrn           | ;Standby mode of PSD                                      |
| 29. | psen_to_ram_en | ;SRCODE bit in VM register                                |
| 30. | periph_mode    | ;Peripheral I/O mode                                      |
| 31. | pmmr1          | ;Power Management mode register-1                         |
| 32. | pmmr0          | ;Power Management mode register-0                         |
| 33. | dout_b         | ;Port B data out register                                 |
| 34. | dirff_b        | ;Port B direction register                                |
| 35. | ctrl_b         | ;Port B control register                                  |
| 36. | spec_b         | ;Port B special function register                         |
| 37. | ctrl_e         | ;Port E control register                                  |
| 38. | dout_e         | ;Port E data register                                     |
| 39. | dirff_e        | ;Port E direction register                                |
| 40. | spec_e         | ;Port E special function register                         |
| 41. | ctrl_d         | ;Port D control register                                  |
| 42. | dout_d         | ;Port D data register                                     |
| 43. | dirff_d        | ;Port D direction register                                |
| 44. | opn_drn_d      | ;Port D Open Drain/CMOS definition register               |
| 45. | dout_c         | ;Port C data register                                     |

**PSDXXX.mst  
Files****PSD4A2.mst File**

|     |           |                                             |
|-----|-----------|---------------------------------------------|
| 46. | ctrl_c    | ;Port C control register                    |
| 47. | dirff_c   | ;Port C direction register                  |
| 48. | opn_drn_c | ;Port C Open Drain/CMOS definition register |
| 49. | pgr0_3    | ;Page Registers 0 through 3                 |
| 50. | psel1     | ;Peripheral I/O mode select product term 2  |
| 51. | psel0     | ;Peripheral I/O mode select product term 1  |
| 52. | csiiop    | ;Chip Select I/O ports                      |
| 53. | es3       | ;EPROM Chip select for block-3              |
| 54. | es2       | ;EPROM Chip select for block-2              |
| 55. | es1       | ;EPROM Chip select for block-1              |
| 56. | es0       | ;EPROM Chip select for block-0              |
| 57. | rs0       | ;PSD SRAM Chip Select                       |
| 58. | pb0       | ;Port B pin-0                               |
| 59. | pb1       | ;Port B pin-1                               |
| 60. | pb2       | ;Port B pin-2                               |
| 61. | pb3       | ;Port B pin-3                               |
| 62. | pb4       | ;Port B pin-4                               |
| 63. | pb5       | ;Port B pin-5                               |
| 64. | pb6       | ;Port B pin-6                               |
| 65. | pb7       | ;Port B pin-7                               |
| 66. | pa0       | ;Port A pin-0                               |
| 67. | pa1       | ;Port A pin-1                               |
| 68. | pa2       | ;Port A pin-2                               |
| 69. | pa3       | ;Port A pin-3                               |
| 70. | pa4       | ;Port A pin-4                               |
| 71. | pa5       | ;Port A pin-5                               |
| 72. | pa6       | ;Port A pin-6                               |
| 73. | pa7       | ;Port A pin-7                               |
| 74. | pe2       | ;Port E pin-2                               |
| 75. | pe3       | ;Port E pin-3                               |
| 76. | pe4       | ;Port E pin-4                               |
| 77. | pe5       | ;Port E pin-5                               |
| 78. | pe6       | ;Port E pin-6                               |
| 79. | pe7       | ;Port E pin-7                               |
| 80. | pd0       | ;Port D pin-0                               |
| 81. | pd1       | ;Port D pin-1                               |
| 82. | pd2       | ;Port D pin-2                               |
| 83. | pd3       | ;Port D pin-3                               |
| 84. | pd4       | ;Port D pin-4                               |
| 85. | pd5       | ;Port D pin-5                               |
| 86. | pd6       | ;Port D pin-6                               |
| 87. | pd7       | ;Port D pin-7                               |
| 88. | pc0       | ;Port C pin-0                               |
| 89. | pc1       | ;Port C pin-1                               |
| 90. | pc2       | ;Port C pin-2                               |

**PSDXXX.mst  
Files**

**PSD4A2.mst File (Cont.)**

- 91. pc3 ;Port C pin-3
- 92. pc4 ;Port C pin-4
- 93. pc5 ;Port C pin-5
- 94. pc6 ;Port C pin-6
- 95. pc7 ;Port C pin-7
- 96. spec\_a ;Port A special function register
- 97. dirff\_a ;Port A direction register
- 98. ctrl\_a ;Port A Control register
- 99. dout\_a ;Port A data register
- 100. portb ;Port B register
- 101. clkln ;PSD input Clock
- 102. reset ;PSD input reset
- 103. csi ;PSD Chip Select
- 104. pe1 ;Port E pin-1 (ALE etc..)
- 105. pe0 ;Port E pin-0 (PSEN, BHE etc..)
- 106. wr ;PSD write signal
- 107. rd ;PSD read signal
- 108. portd ;Port D register
- 109. portc ;Port C register
- 110. adioh ;Address/Data bus high byte
- 111. adiol ;Address/Data bus low byte
- 112. porta ;port A register
- 113. pgr0 ;Page Register bit 0
- 114. pgr1 ;Page Register bit 1
- 115. pgr2 ;Page Register bit 2
- 116. pgr3 ;Page Register bit 3

**Stimulus  
File  
tutor.stl**

```
//+++++
// User-Defined Parameters
//+++++

parameter pmmr0='hC0B0, cntr0='hC098, img0='hC090;
parameter dlcY='hC0A6, cmd0='hC0A0, global='hC0A8;
parameter freeze='hC0A4, status='hC0A9, page='hC0E0;
parameter sram_loc='h8476, es_loc1='h39FE, es_loc2='h146C;
parameter es_loc3='h39FD;
parameter bhe_on=0, bhe_off=1, page9=9;
parameter clear=0, freeze_on=1, unfreeze=0;

//+++++
// User-Defined Tasks
//+++++

task write (addr_bus,bhe_value,data_in);

input [15:0] addr_bus;
input [15:0] data_in;
input bhe_value;

 begin

 #20 ale = 1; //Latch the address lines
 #20 adio = addr_bus; //Set-up the right address
 bhe = bhe_value;
 #20 ale = 0; //Ale inactive

 #20 adio = data_in; // Write operation

 #40 wr = 0; // Write pulse
 #100 wr = 1; // Write ends
 #10 adio = Z16; bhe =Z;

 end

endtask

task read (addr_bus);

input [15:0] addr_bus;

 begin

 #20 ale = 1; //Latch the address lines
 #20 adio = addr_bus; //Set-up the right address
 bhe = 0;
 #20 ale = 0; //Ale inactive

 #20 adio = Z16; // Float Address bus

 #40 rd = 0; // Rd pulse
 #100 rd = 1; // Rd ends
 #10 bhe = Z;

 end

endtask
```

**Stimulus  
File  
tutor.stl  
(Cont.)**

```
//+++++
// USER defined buses
//+++++

reg [4:0] din;
assign {d4, d3, d2, d1, d0} = din;
//din defines the 5 parallel-in bits loaded to the down-counter.
//The down-counter is implemented using the PLD macrocells
//(unrelated to timer-0 unit that is configured here as an event-counter).

reg [2:0] event_in;
assign {event3, event2, event1} = event_in;
//event_in combines the 3 bits whose level changes are regarded as events.

reg [2:0] hi_ad;
assign {a18, a17, a16} = hi_ad;
//hi_ad groups together high-order address lines.

// 16-Bit, mux mode, ale is used.

//+++++
//-----> Starting Point of Stimulus File <-----
//+++++

initial
begin

 wr = 1; rd = 1; clkIn = 0;
 reset = 0; csi = 0; adio = 'h0000;
 bhe = 1; cnt4 = Z; wstc = Z;
 wdout = Z; hi_ad = 0;

//d4 - d0 have a value of 29
 din = 29;

//Mc2tMr0 pulses create timer-0 events on their low-to-high transitions.
// mc2tMr0 = event1 & !event2 + !event3,
// according to the ABEL description.
 event_in = 7;

 load = 0; //Initialize the down-counter to no-load
 cntout_en = 1; //cnt4 is tri-stated
 #500 reset = 1;

//Initialize the part to power-saving mode. Write 38H to the PMMR0 reg:
//Disable clkIn from the PLD-AND array, put PLD in non-turbo mode, EPROM in
//CMISER mode.
```

**Stimulus  
File  
tutor.stl  
(Cont.)**

```
//Invoke the task with the right parameters
10
 write (pmmr0,bhe_off,'h38); //Byte-low write operation

//Timer-0 data initialized to 0
 write (cntr0,bhe_on,clear); //Word-write operation

//Load down-counter with 29, enable cnt4 to output pin
 load = 1; cntout_en = 0; // Counter starts
#32 load = 0; //End of load pulse, load duration is a clock cycle

//Clear IMG0 high & low byte registers
 write (img0,bhe_on,clear); //Word-write operation

//read-back data on IMG0 reg
 read (img0); //Word-read operation

//Writing DLCY data. Timer Clock is the clock input (clkIn) frequency
//divided by 7.
 write (dlcy,bhe_off,3); //Byte-write operation

//Writing CMD-0 data to configure Timer-0

 //Write Data of 6:

//0 Event Count mode/waveform
//1 Increment mode
//1 Enable Timer_0
//X Timer output active level (don't care - no timer output)
//X Determines whether the timer increments on
// the rising or falling edge of the PIN. Since Macrocell is
// selected, this is a don't care bit.
//0 Trigger(=load/store) from Macrocell, not from pin
//0 Enable trigger command from macrocell
//0 Enable/Disable by MACROCELL/PIN

 write (cmd0,bhe_off,6); //Byte-write operation

//Issue another load down-counter pulse.
//Load counter with 26, enable cnt4 to output pin
 din =26; load = 1; cntout_en = 0;
#32 load = 0;

//Global Reg data written to
#200
 write (global,bhe_off,6); //Byte-write operation
//Write Data of 06h, no clock division, Event Count mode/Time Capt mode

// d4 - d0 have a value of 31, load down-counter
 din = 31;
 load = 1;
#32 load = 0; //End of load pulse
```

**Stimulus  
File  
tutor.stl  
(Cont.)**

```
// Timer starts counting here !

// Create events. Note that their width is not important.
// Timer-0 increments on every low-to-high transition of
// the mc2tmr0 PPLD signal.

#40 event_in = 1; //Create 1st event
#30 event_in = 7;
#290 event_in = 1; //Create 2nd event
#30 event_in = 7;
#290 event_in = 1; //Create 3rd event
#30 event_in = 7;
#290 event_in = 1; //Create 4th event
#25 event_in = 7;
#365 event_in = 1; //Create 5th event
#22 event_in = 7;

// Write to Freeze Command Reg data

#240 write (freeze,bhe_off,freeze_on); //Byte-write operation

// Create 6th event. It occurs together with the issuance of a freeze command.
event_in = 1;
#24 event_in = 7;

// Create more events, the timer continues counting while IMG0 is frozen.
#365 event_in = 1; //Create 7th event
#30 event_in = 7;

// read data on Status reg, verify that freeze_ack is high
read (status); //Byte-High read operation

// read data on IMG0 reg since the counter is frozen.
read (img0); //Word-read operation

event_in = 1; //Create 8th event
#61 event_in = 7;

// Write to Unfreeze the Freeze Command Reg data

#700 write (freeze,bhe_off,unfreeze); //Byte-write operation
```

**Stimulus  
File  
tutor.stl  
(Cont.)**

```
//-----
// MEMORY TESTS
//-----

//Setting up address 8476h of SRAM
 write (sram_loc,bhe_on,'h5A27'); //Word-write operation

//read data of EPROM location 39FEh, es0 is active
 read (es_loc1); //Word-Read operation
//Expect 0123h on data bus

//read data of EPROM location 146Ch, es0 is active
 read (es_loc2); //Word-Read operation
//Expect 0123h on data bus

//Setting up address C0E0h of Page-Reg. Write #9 to it.
 write (page,bhe_off,page9); //Byte-write operation

//read data of SRAM location 8476h
 read (sram_loc); //Word-Read operation
//Expect 5A27 on the data bus

 #20 din = 12; // Change din to 12
 #10 load = 1; //Load 12 to down-counter
 #32 load =0; //end of down-counter load

//read data of Page-Reg. location C0E0h
 read (page); //Word-Read operation
//Expect 9 on the low order byte of data bus

//Read data of EPROM location 39FDh, es1 will be selected(based on page 9)
 read (es_loc3); //Byte-high read operation
//Expect 45 on D15-D8

//Read data of EPROM location 146Ch, es1 will be selected
 read (es_loc2); //Word-Read operation
//Expect 4567h on D15 - D0

 end

//Generate a continuous clock signal
always
 #16 clkIn = ~clkIn;
```



## Files For Other Bus Structures

Included in the Examples subdirectory is a set of .abl, .glc and .stl files for four design examples. These designs are similar to the tutorial design example except for the bus interface configuration. More examples will be included later.

The following is the current file list:

|                   |                                          |
|-------------------|------------------------------------------|
| <b>mux8.abl</b>   | ABEL file for 8-bit multiplexed bus      |
| <b>mux8.stl</b>   | Stimulus file for mux8.abl               |
| <b>nmux8.abl</b>  | ABEL file for 8-bit non-multiplexed bus  |
| <b>nmux8.stl</b>  | Stimulus file for nmux8.abl              |
| <b>nmux16.abl</b> | ABEL file for 16-bit non-multiplexed bus |
| <b>nmux16.stl</b> | Stimulus file for nmux16.abl             |
| <b>m683xx.abl</b> | ABEL file for Motorola 683XX type bus    |
| <b>m683xx.stl</b> | Stimulus file for m683xx.abl             |

---

## PSD5XX/4XX Architecture Overview

The PSD5XX/4XX devices are new members of the Field Programmable Microcontroller Peripheral product line from a WSI. The PSD5XX/4XX devices provide advanced features such as a complex ZPLD, Timer/Counters, Interrupt Controller, Page Logic, and expanded I/O Ports to greatly enhance the performance of virtually any microcontroller.

The PSD5XX/4XX also replaces the basic building blocks in embedded designs. These include the EPROM block, SRAM, decoders, address latches, I/O Ports and other discrete components. Two of the advantages of the PSD5XX/4XX are the flexibility and programmability of the part. Chip functions can be modified or changed by reconfiguration or by redefining the ZPLD logic equations.

Because of its flexible configuration options, the PSD5XX/4XX is able to interface to a wide range of microcontrollers or microprocessors.

---

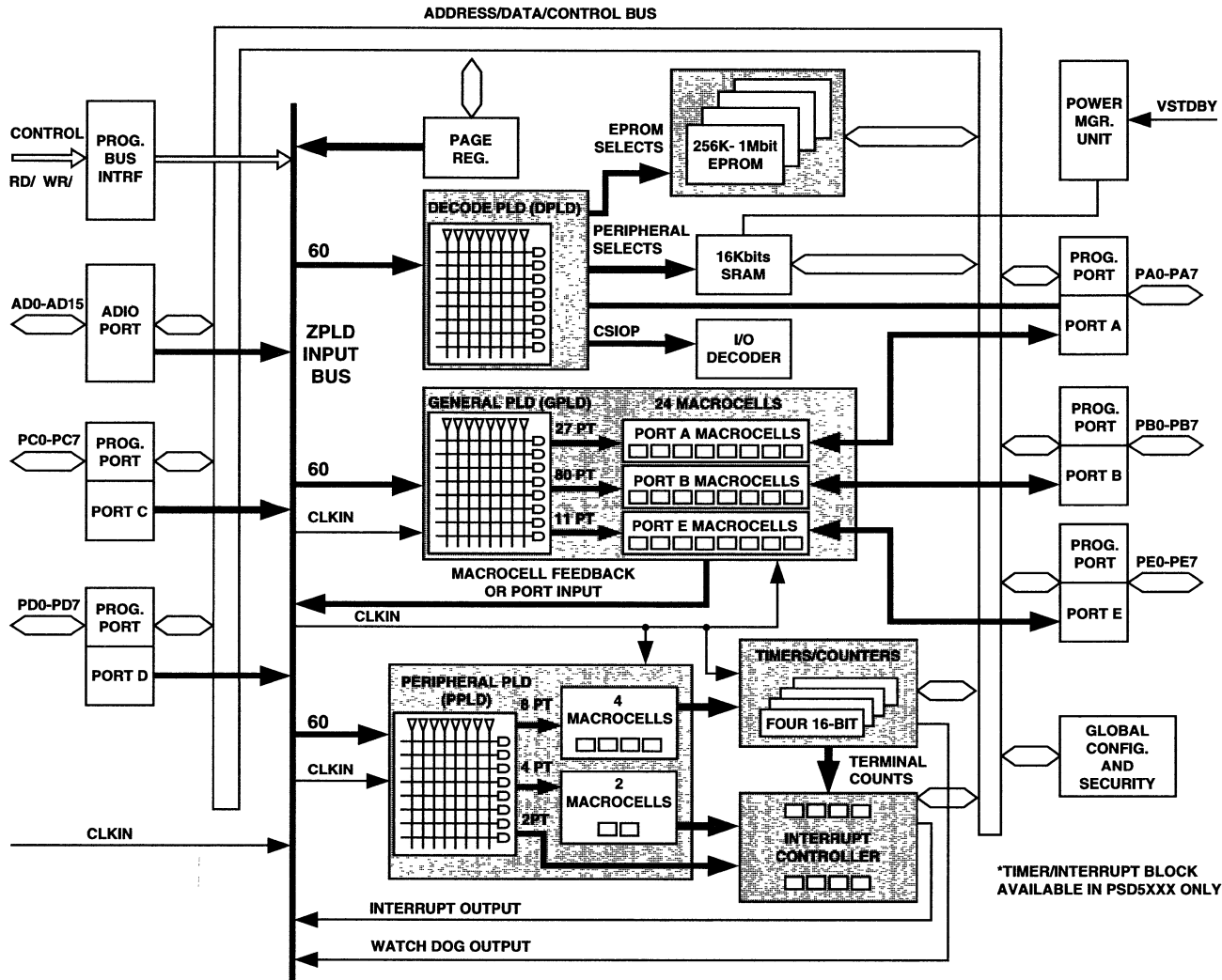
## PSD5XX/4XX Architecture

Figure 5 is the top-level block diagram of the PSD5XX/4XX. The PSD5XX/4XX consists of the following main functional blocks and features:

- Bus Interface**
- ZPLD Block**
- Memory Block**
- I/O Ports**
- Counter/Timer and Interrupt Controller Block (PSD5XX only)**
- Power Management**
- Chip Security**
- Page Logic**
- Peripheral I/O Mode**

All the functional blocks are connected to the internal Address and Data bus. The Data Bus is 8- or 16-bit, depending on the PSD5XX/4XX configuration. The Address Bus width is variable and is defined by the user. The ZPLD (Zero Power PLD) has its own input and output buses. The GPLD (General Purpose PLD) and PPLD (Peripheral PLD) can operate by themselves and be independent from the microcontroller.

During normal bus cycles, the Decoding PLD (DPLD) monitors the Address Bus and determines if any of the PSD5XX/4XX internal devices should be selected and enabled. All the internal blocks can be accessed by the microcontroller, including the output of macrocells in the ZPLDs.



\*TIMER/INTERRUPT BLOCK AVAILABLE IN PSD5XXX ONLY

Figure 5. PSD5XX/4XX Block Diagram

**PSD5XX/4XX  
Architecture  
(Cont.)**

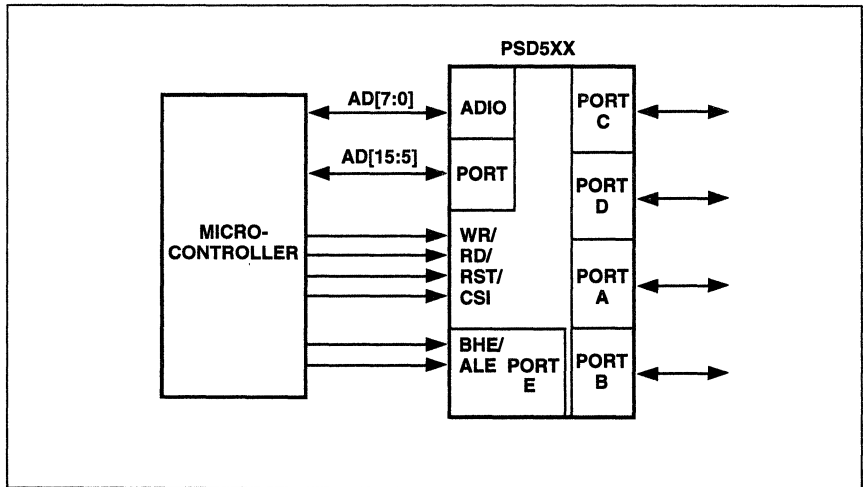
**Bus Interface**

The PSD5XX/4XX can interface to many microcontrollers or microprocessors. The Bus Interface is user configurable, and is able to support many types of bus structures. Figure 6 shows the interface between the PSD5XX/4XX and a processor with a 16-bit multiplexed address/data bus (AD0-AD15). The AD bus from the processor connects directly to the ADIO port on the PSD5XX/4XX. The Bus Interface latches the address lines at the falling edge of the ALE signal. Data is driven onto the AD bus in a read bus cycle. Bus control signals (RD/, WR/, and so on) from the processor also connect directly to the PSD5XX/4XX with no glue logic.

For processors that have non-multiplexed buses, the bus interface configuration requires that the address bus connect to the ADIO port, while the data bus goes to Port C and Port D, depending on the bus width.

The data ports of the PSD5XX/4XX are in tri-state mode if none of the internal devices are selected.

**Figure 6. Bus Interface Connection**



## PSD5XX/4XX Architecture (Cont.)

### ZPLD Block

The ZPLD Block consists of three embedded ZPLDs: the DPLD, GPLD, and PPLD.

#### □ DPLD

The Decoding PLD (DPLD) generates select signals to internal I/O devices, EPROM blocks, and SRAM. The DPLD has 61 inputs and 8 outputs. Each output has one product term.

#### □ GPLD

The General Purpose PLD (GPLD) provides up to 24 programmable macrocells for general or complex logic implementation. The GPLD shares the same input bus as the DPLD. The input/output of the 24 macrocells are connected to I/O pins on Port A, B, and E. Figure 7 shows a macrocell circuit that is connected to Port B. Macrocells connected to Port A and E have similar circuitry.

The PSD4XXA2 has 16 macrocells on Port A and B, while the PSD4XXA1 has only eight macrocells on Port B, with eight combinatorial macrocells on Port A.

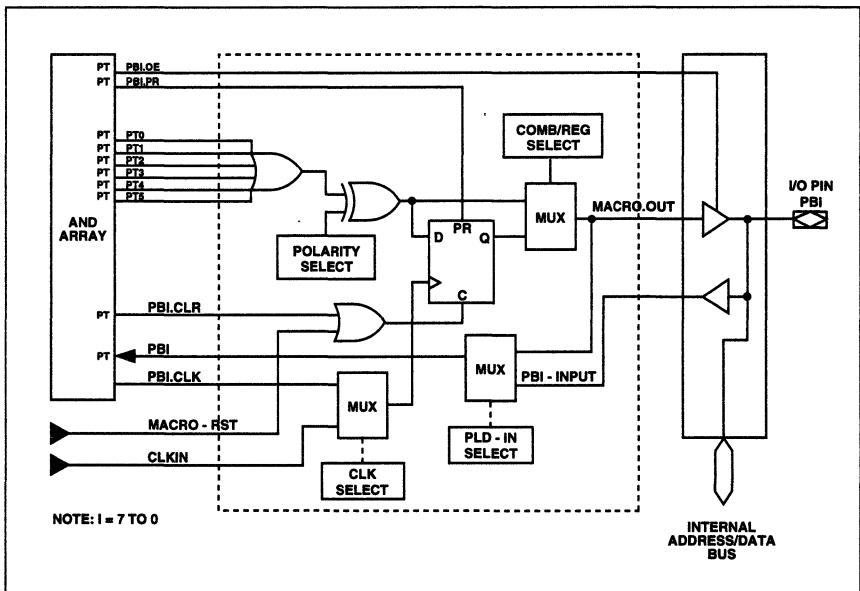
#### □ PPLD

The Peripheral PLD (PPLD), which is available in the PSD5XX only, has six programmable macrocells. The output of the macrocells are used as inputs to the Timer and Interrupt Controller, which provide additional control over the operation of the Timers.

The three ZPLDs share the same input bus which consists of up to 61 signals. These signals include the address lines and control signals from the microcontroller, the Timer/Interrupt Controller outputs, the Page Logic outputs, and inputs from Ports A, B, C, D, and E. Ports A, B, and E can also be configured as output ports for the GPLD's macrocells.

You can reduce the power consumption of the ZPLDs by turning the ZPLD Turbo bit off in the Power Management Mode Register. In this mode, the ZPLD puts itself into standby mode if none of the 61 inputs are switching for a period of 100 ns or more.

**Figure 7. Port B Macrocell Circuit**



## **PSD5XX/4XX Architecture**

(Cont.)

### **Memory Block**

The PSD5XX/4XX Memory Block consists of two sections, EPROM and SRAM.

#### **EPROM**

EPROM is used for program code and data storage. The EPROM consists of four separate blocks, each having its own chip select signal defined by you through the DPLD. There are three EPROM sizes, as follows:

- 256 Kbits
- 512 Kbits
- 1 Mbit

#### **SRAM**

SRAM supplements to the microcontroller's internal RAM. The SRAM has one 16Kbit block, which has a battery back-up mode.

Both the EPROM and SRAM can be configured as X8 or X16, depending on the data bus width of the microcontroller.

### **I/O Ports**

The PSD5XX/4XX has five 8-bit I/O Ports. Each port performs multiple functions and is user-programmable. The port functions can be classified into three groups, I/O Ports to the microcontroller, Address or Data Ports, and I/O Ports for internal PSD5XX devices.

#### **I/O Ports to the Microcontroller**

I/O Ports to the Microcontroller (Standard MCU I/O can be read or written to by the microcontroller).

#### **Address or Data Ports**

For microcontrollers with non-multiplexed buses, Port C is connected to the low byte on the data bus and Port D is connected to the high byte (Address Bus connects to the ADIO Port).

Port A can also be used as input for the higher address lines (A16 and up). These address lines are included in the ZPLD input bus and are used in address decoding.

In applications where lower order address lines are needed for peripheral I/O devices, the I/O Ports can be configured to provide latched address output.

## PSD5XX/4XX Architecture (Cont.)

### □ I/O Ports for Internal PSD5XX/4XX Devices

Ports C and D may serve as input ports for the GPLD, and Port A, B, and E may serve as I/O ports for the macrocells.

Ports A, B, and E may serve as I/O ports for the Timer/Counter and Interrupt Controller.

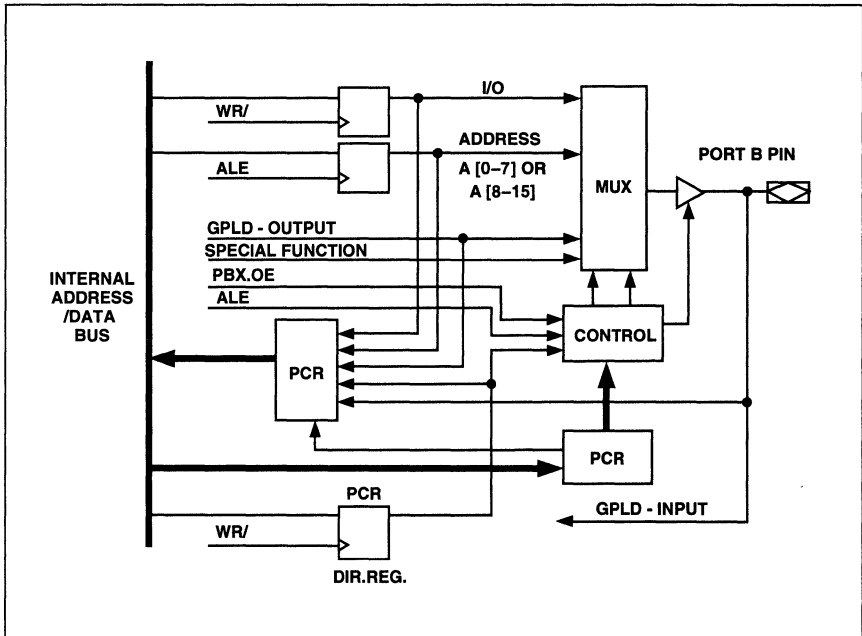
There are additional functions that are unique to each port. Port A has a Peripheral I/O mode which, if activated, allows Port A to serve as a transceiver on the microcontroller data bus.

Figure 8 shows the pin structure and circuitry of an I/O pin on Port B. The PCR (Port Configuration Register) controls the operation of the Port. As an output port, the MUX select one of the four sources as an output. For Port B, these outputs are as follows:

- Standard MCU I/O
- Latched address output
- GPLD macrocell I/O
- Timer output (Special Function)

As an input pin, the pin can be configured as an input to the ZPLD, or as an input for the Standard MCU I/O mode. Other registers in the pin structure can be accessed by you through the PDR (Port Data Register).

**Figure 8. Pin Structure, Port B**



**PSD5XX/4XX  
Architecture  
(Cont.)**

**Counter/Timer and Interrupt Controller**

The Counter/Timer block, which is available in the PSD5XX only, consists of four 16-bit counters. All four counters run on the same input clock. The desired clock frequency (the maximum input clock, CLKIN, is 30 MHz—the maximum counter/timer clock is 7.5 MHz) is selected by programming the Clock Scaler with the proper divisor.

The Counter/Timers have five modes of operation:

- Waveform Mode
- Pulse Mode
- Event Counter Mode
- Time Capture Mode
- WatchDog Mode

The mode of operation is specified through the Command Register. Figure 9 shows the Counter/Timer and Interrupt Controller block diagram. The MUX selects the source of the Counter/Timer control inputs. The control source can come from user software, external inputs, or macrocell outputs from the PPLD. Outputs from the Counter/Timers in Waveform or Pulse Mode are routed to Port A or B. WatchDog output, WDOG2PLD, needs to go to the ZPLD before it can be taken to an I/O pin as a ZPLD output defined by you.

The Interrupt Controller provides a convenient way to manage a design with multiple interrupts. The Interrupt Controller accepts eight interrupt inputs, including four Terminal Counts from the Counter/Timers, two from the macrocells and two from the AND ARRAY of the PPLD. The PSD5XX/4XX does not have dedicated pins for external interrupt inputs. You have to specify the input as ZPLD input on Port C or D in order to generate the proper product term for the Controller.

Interrupt inputs can be either level or edge sensitive. The inputs are priority decoded, where IR7 has the highest priority. The Controller also has the ability to mask out any unwanted input.

**Power Management**

The PSD5XX/4XX has a Power Management Register that allows you to configure the chip power consumption in real time. You may activate four power saving options.

**Power Down Mode**

In this mode, the PSD5XX/4XX automatically puts itself into power-down mode if the microcontroller is inactive. You can also put the PSD5XX/4XX into power-down mode by deselecting the chip select input (CSI) pin.

**Sleep Mode**

Once in the Power Down Mode, the PSD5XX/4XX has the option to go into Sleep Mode. The PSD5XX/4XX consumes less power but requires recovery time to get back to normal operation.

**EPROM CMISER Mode**

This mode allows the PSD5XX/4XX to turn off the EPROM when it is not being accessed.

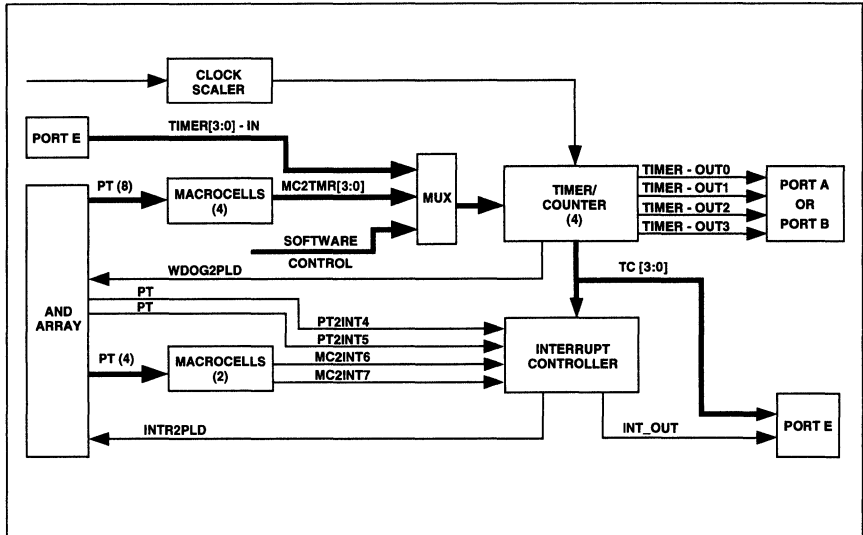
**ZPLD Turbo Mode**

The PSD5XX/4XX's ZPLD saves power by turning off the Turbo bit. This adds 10 ns additional delay to the ZPLD.

Through the Power Management Register, the input clocks to the ZPLD and Counter/Timers can be turned off to save power due to AC activity.

## PSD5XX/4XX Architecture (Cont.)

**Figure 9. Counter/Timer and Interrupt Block**



### Chip Security

The PSD5XX/4XX has a programmable security bit that offers protection from unauthorized duplication. When the security bit is active, the contents of the EPROM, ZPLD fusemap, and nonvolatile configuration bits are prevented from being read by an EPROM programmer. If a special decoding technique is implemented, it will also prevent the codes from being disassembled by Emulators.

### Page Logic

For microcontrollers with limited addressing capability, the PSD5XX/4XX provides a four-bit Page Register that increases the memory space by a factor of 16. Outputs from the Page Register are available as inputs to the ZPLD for decoding purpose.

### Peripheral I/O Mode

The Peripheral I/O Mode is available on Port A only. In this mode, Port A acts as a tri-state transceiver on the microcontroller data bus. The enable and directional control signals to the Port are defined in the DPLD.







# Programmable Peripheral Application Note 033

## Keypad Interface to PSD4XX/5XX with Autoscanning

By Ching Lee

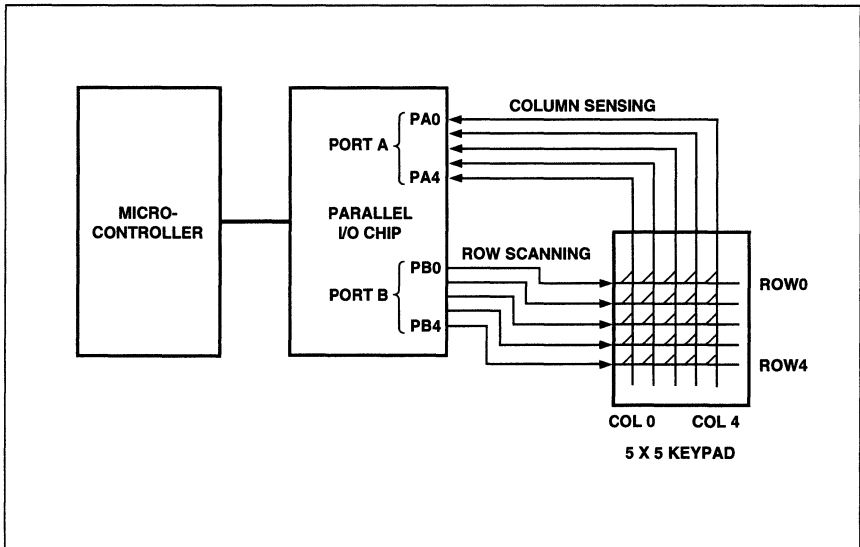
### Introduction

The integration of complex PLD and I/O functions in the PSD4XX/5XX is well suited to the implementation of I/O interface logic such as a keypad controller. This application note describes how to take advantage of this PSD4XX/5XX feature to design an efficient and power saving keypad interface.

### Typical Keypad Interface

A keypad consists of a matrix of pressure or touch activated switches. Figure 1 shows a typical keypad interface using a PIO (parallel I/O) chip. It is assumed that the keypad has internal pull ups for the rows and columns. The keypad has 25 keys, and is arranged in a 5 (row) x 5 (column) matrix. In this example, Port B is configured as an output port (PB0 – PB4) and driving logic "0" to the 5 row inputs of the keypad. Port A is configured as an input port (PA0 – PA4). PA0 – PA4 are normally pulled high by internal keypad resistors until one of the keys is pressed. For example, if key [3,1] (row 3, column 1) is pressed, then the "0" on PB3 is passed through the closed switch to column 1.

Figure 1. Keypad Interface



**Typical  
Keypad  
Interface  
(Cont.)**

Detection of the key closure usually involves the following steps:

- The microcontroller program continues to poll Port A to determine if any of the inputs are low. If data on Port A is switched from “1F” (no keys are pressed) to “17” (PA3 is low), the microcontroller can then identify that one of the keys in column 1 is pressed.
- To eliminate erroneous read due to key switch bouncing, the software executes a delay routine and reads Port A again after the column inputs are stable.
- After a key closure from column 1 is detected, the microcontroller reverses the direction bits of Port A and Port B. Now Port A acts as an output port and Port B as an input port. Port A drives back “17” to the column inputs.
- The microcontroller then reads Port B which acts as an input port for the rows. If it reads “17” (PB3 is low), then it can identify that the key common to row 3 and column 1 (key [3,1]) is pressed. This can be done through a look up table.

This keypad interface technique can also be implemented in the PSD4XX/5XX by connecting the rows and columns to the I/O ports as described above. The microcontroller must be always active and must keep on polling the Ports for keypad input.

---

**A More  
Efficient  
Keypad  
Interface  
Implementation**

The major overhead of the above keypad interface is:

- The microcontroller must poll the port at a fixed frequency, thus reduce the processor performance.
- The microcontroller must remain active and consumes power even when the keypad is idle.

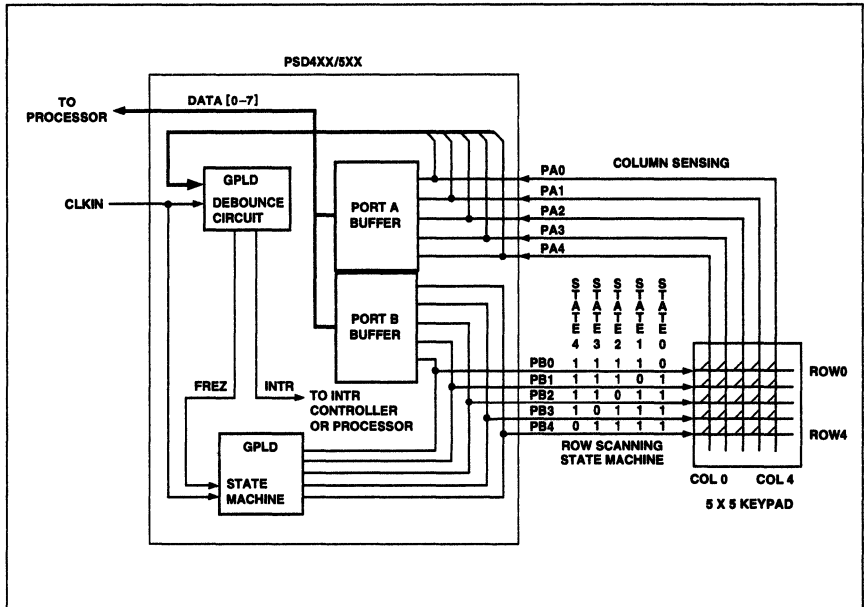
A more efficient way of interfacing to a keypad which reduces the above overhead is described here. The PSD device will perform the interface function automatically by:

- Implementing a hardware debounce circuit in the GPLD of the PSD4XX/5XX, replacing software debouncing.
- Implementing a state machine in the GPLD to scan the rows of the keypad automatically, replacing software polling.
- Setting Port A as a column input port and Port B as a scan output port.
- Generating an interrupt to the microcontroller only when a key is pressed.

The concept of this design is shown in the block diagram in Figure 2. The block diagram shows only the I/O Ports and GPLD portion of the PSD4XX/5XX which are used in the keypad interface. The following paragraphs describe the PSD configuration and GPLD logic function.

**A More Efficient Keypad Interface Implementation (Cont.)**

**Figure 2. PSD Implementation**



3

**PSD I/O Port Configuration**

Port B is configured as an output port for the GPLD. Outputs of the scanning state machine are routed to Port B and are connected to the row inputs of the keypad. The outputs of the state machine can be read by the microcontroller via the Port B Buffer (Data In Register or Macrocell Out Register).

Port A is configured as an input port for the GPLD and is connected to the column outputs of the keypad. The column outputs can also be read by the microcontroller via the Data In Register of Port A.

**GPLD Logic Implementation**

The GPLD implements both a debounce circuit and a scanning state machine. Both functions can be fitted in the PB macrocells and can run on the same input clock (clkIn). The state machine is clocked by the rising edge of clkIn, while the debounce circuit uses the falling edge of clkIn.

## The Debounce Circuit

The bounces on the keypad column outputs due to switch opening/closing can lead to an erroneous result. The debounce circuit performs two functions:

- Generates a “freeze” signal when a key is pressed. This signal, *frez*, is used to stop the state machine until the key is released. The ABEL equation is

$$\text{frez} := !(\text{col0} * \text{col1} * \text{col2} * \text{col3} * \text{col4});$$

- Generates an interrupt, “intr”, to the microcontroller when the column outputs stay low for two (or more) consecutive clocks. This is to ensure that the inputs are stable before interrupting the microcontroller. The ABEL equation is

$$\text{intr} := \text{frez} * !(\text{col0} * \text{col1} * \text{col2} * \text{col3} * \text{col4});$$

The clock input to the debounce circuit can be derived from the system clock, but the clock period should be larger than the switch bounce time.

## The Scanning State Machine

The state machine does the keypad scanning by sending a “running 0” pattern to the row inputs at the rising edge of the input clock via Port B. For a 5 row keypad, the “running 0” patterns at each clock are:

| <i><b>Clock</b></i> | <i><b>Row 0</b></i> | <i><b>Row 1</b></i> | <i><b>Row 2</b></i> | <i><b>Row 3</b></i> | <i><b>Row 4</b></i> |
|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| 1                   | 0                   | 1                   | 1                   | 1                   | 1                   |
| 2                   | 1                   | 0                   | 1                   | 1                   | 1                   |
| 3                   | 1                   | 1                   | 0                   | 1                   | 1                   |
| 4                   | 1                   | 1                   | 1                   | 0                   | 1                   |
| 5                   | 1                   | 1                   | 1                   | 1                   | 0                   |
| 6                   | 0                   | 1                   | 1                   | 1                   | 1                   |
| 7                   | 1                   | 0                   | 1                   | 1                   | 1                   |

The pattern is repeated every five clocks. The sequence of events when a key [3,1] (row 3, column 1) is pressed at clock 2 are:

- At clock 2: Key [3,1] is pressed. The “0” in the pattern (row 1) is not passed to column 1 output.
- At clock 3: The “0” in the pattern (row 2) is not passed to column 1 output.
- At clock 4: The “0” in the pattern (row 3) is passed to column 1 output via the closed/pressed key [3,1].
- At the falling edge of clock 4, the “0” causes the debounce circuit to generate the “frez” signal and freezes the state machine.
- At the next clock, if column inputs are stable and remain low, the debounce circuit generates an interrupt which wakes up the microcontroller.
- The microcontroller reads Port A. The column inputs are “17h” which indicates a key in column 1 was pressed.
- The microcontroller reads the output of the state machine (“running 0” pattern). The value is “1Dh”. This indicates a key in row 3 was pressed.
- By using a look up table, the microcontroller identifies the pressed key to be key [3,1]. The microcontroller puts itself back to power down/sleep mode.
- The state machine remains in a stop condition until the pressed key is released. After the key is released, the state machine returns to generating the “running 0” pattern.

## The Scanning State Machine

(Cont.)

The state machine has 5 states and you can assign the “running 0” pattern as the state value. The operation of the state machine, including the debounce circuit, is described in ABEL as follows:

```

“state values (running 0 pattern)
sreset = ^b00000;
scanr0 = ^b11110;
scanr1 = ^b11101;
scanr2 = ^b11011;
scanr3 = ^b10111;
scanr4 = ^b01111;

frez := !(col0 * col1 * col2 * col3 * col4); active high

intr := frez * !(col0 * col1 * col2 * col3 * col4); active high

“frez is active when key is pressed

rowreg.c = clk; “scanning clk = clk
rowreg.re = !rst; “clear registers at reset

state_diagram rowreg;

state sreset: goto scanr0;
state scanr0: if !frez then scanr1 else scanr0;
state scanr1: if !frez then scanr2 else scanr1;
state scanr2: if !frez then scanr3 else scanr2;
state scanr3: if !frez then scanr4 else scanr3;
state scanr4: if !frez then scanr0 else scanr4;

“if no frez, state machine runs continuously

```

3

## Implement The Keypad Interface In The PSD4XX/5XX

This Keypad design can be implemented in any of the PSD4XX/5XX devices. There are two ways to implement the keypad row scanning function:

- Use the state machine as described above. This approach is restricted to a keypad with a few rows. As the number of rows increase, the number of product terms required by the state machine also increases and soon there will not be enough product terms. The ABEL file which defines the GPLD logic function of this implementation, keya.abl, is shown in Appendix A.
- Use a circular shift register to generate the “running 0” pattern instead of a state machine. The shift register needs only one product term per output and can interface to keypads with large row counts. During reset, the register is set/preset with the “running 0” pattern (11110). After reset, the “0” in the pattern is shifted and repeated between the row inputs. The clock input to the shift register is “anded” with the frez signal and will stop shifting after a key is pressed. The ABEL file of this implementation is shown in Appendix B.

A stimulus file, keypad.stl, which simulates the keypad operation is included in Appendix C. The stimulus file shows the steps required to set up Port A and the reading of column and row values by the microcontroller after a key is pressed.

The PSD4XX/5XX frees up valuable I/O ports on the microcontroller, and off-loads some of the keypad software overhead. The resulting design allows better utilization of microcontroller resources.

**Appendix A.**  
**KEYA.ABL**  
**File**

```
module keya
title 'test:keyboard autoscanning, 80C196 bus interface';

"Input signals

col0, col1, col2, col3, col4 pin 27,26,25,24,23; "key bd column inputs

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

clkln, rst pin 42, 40;

"PLD output signals.

csiop, rs0, es0, es1, es2, es3 node; "More outputs using reserved names.
intr pin; "key board interrupt
frez node;
nclkln node; "reverse of clkln

row0, row1, row2, row3, row4 pin 50,49,48,47,46; "row scanning outputs
row0, row1, row2, row3, row4 is type 'buffer, reg_d';

"Definitions

rowreg = [row4, row3, row2, row1, row0];

"state values
sreset = ^b00000;
scanr0 = ^b11110;
scanr1 = ^b11101;
scanr2 = ^b11011;
scanr3 = ^b10111;
scanr4 = ^b01111;

c = .c. ; " Clock pulse definition
X = .x. ; " Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];
```

**Appendix A.**  
**KEYA.ABL**  
**File**  
**(Cont.)**

**equations**

```
csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF); "256 block
rs0 = (Address <= ^h087FF) & (Address >= ^h08000); "2k block
es0 = (Address <= ^h01FFF) & (Address >= ^h00000); "32KB block
```

```
frez := !(col0 * col1 * col2 * col3 * col4); "active high frez
intr := frez * !(col0 * col1 * col2 * col3 * col4); "active high intr
```

"intr is active when key is pressed

```
nckin = !ckin; "reverse ckin for debounce circuit
frez.c = nckin; intr.c = nckin;
rowreg.c = ckin; "scanning clk = ckin
frez.re = !rst; intr.re = !rst;
rowreg.re = !rst; "reg. clear input
```

state\_diagram rowreg;

```
state sreset: goto scanr0;
state scanr0: if !frez then scanr1 else scanr0;
state scanr1: if !frez then scanr2 else scanr1;
state scanr2: if !frez then scanr3 else scanr2;
state scanr3: if !frez then scanr4 else scanr3;
state scanr4: if !frez then scanr0 else scanr4;
```

"if no interrupt, state machine runs continuously

**test\_vectors**

```
([ckin, rst, col0, col1, col2, col3, col4] -> [row0, row1, row2, row3, row4, intr])
[c, 0, 1, 1, 1, 1, 1] -> [0, 0, 0, 0, 0, 1];
[c, 0, 1, 1, 1, 1, 1] -> [0, 0, 0, 0, 0, 1];
[c, 1, 1, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1] -> [1, 0, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1] -> [1, 1, 0, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1] -> [1, 1, 1, 0, 1, 1];
[c, 1, 1, 1, 1, 1, 1] -> [1, 1, 1, 1, 0, 1];
```

"key (1,1) is pressed/closed

```
[c, 1, 1, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1];
[c, 1, 1, 0, 1, 1, 1] -> [0, 1, 1, 1, 1, 0];
```

"column (col1) detects key is pressed, intr is generated. Scanning stops

"until intr goes away

```
[c, 1, 1, 0, 1, 1, 1] -> [0, 1, 1, 1, 1, 0];
[c, 1, 1, 0, 1, 1, 1] -> [0, 1, 1, 1, 1, 0];
[c, 1, 1, 0, 1, 1, 1] -> [0, 1, 1, 1, 1, 0];
```

"

"MCU reads column inputs and scanning outputs, determined key (1,1) has been "closed. Later key (1,1) is released, intr becomes inactive and scanning resumes

```
[c, 1, 1, 1, 1, 1, 1] -> [1, 0, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1] -> [1, 1, 0, 1, 1, 1];
```

**END**





**Appendix B.**  
**KEYB.ABL**  
**File**

```

module keyb
title 'test:keyboard autoscanning, 80C196 bus interface';

"Input signals

col0, col1, col2, col3, col4 pin 27,26,25,24,23; "column inputs, Port A

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

clkln, rst pin 42, 40;

"PLD output signals.

csiop, rs0, es0, es1, es2, es3 node; "More outputs using reserved names.
intr pin "key board interrupt
frez node;
nckln node; "reverse of clkln
row0, row1, row2, row3, row4 pin 50, 49, 48, 47, 46; "row scanning outputs
row0, row1, row2, row3, row4 is type 'buffer, reg_d';

"Definitions

rowreg = [row4, row3, row2, row1, row0];

c = .c. ; "Clock pulse definition
X = .x. ; "Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF); "256 block
rs0 = (Address >= ^h08000) & (Address <= ^h087FF); "2k block
es0 = (Address >= ^h00000) & (Address <= ^h01FFF); "8KB block
es1 = (Address >= ^h02000) & (Address <= ^h03FFF); "8KB block

frez := !(col0 * col1 * col2 * col3 * col4); "active high frez
intr := frez * !(col0 * col1 * col2 * col3 * col4); "active high intr

"frez/intr is active when key is pressed

```

**Appendix B.**  
**KEYB.ABL**  
**File**  
**(Cont.)**

```

nclkln = !clkln; "reverse clkln for debounce circuit
frez.c = nclkln; intr.c = nclkln;
frez.re = !rst; intr.re = !rst;
rowreg.c = clkln & !frez; "scanning clk = clkln if no frez

row0.re = !rst ; "set row registers initial value to 11110
row1.pr = !rst ; "PSD macrocell has active high reset
row2.pr = !rst ;
row3.pr = !rst ;
row4.pr = !rst ;

row0.d = row4.q; "5-bit shift register
row1.d = row0.q; "shifting stops if frez is active
row2.d = row1.q;
row3.d = row2.q;
row4.d = row3.q;

```

"if no frez, shift register runs continuously

test\_vectors

```

([clkln, rst, col0, col1, col2, col3, col4] -> [row0, row1, row2, row3, row4, intr])
[c, 0, 1, 1, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 1];
[c, 0, 1, 1, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1, 1] -> [1, 0, 1, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1, 1] -> [1, 1, 0, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1, 1] -> [1, 1, 1, 0, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1, 1] -> [1, 1, 1, 1, 0, 1, 1];

```

"key (1,1) is pressed/closed

```

[c, 1, 1, 1, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 1];
[c, 1, 1, 0, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 0];

```

"column (col1) detects key is pressed, intr is generated. Scanning stops

"until intr goes away

```

[c, 1, 1, 0, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 0];
[c, 1, 1, 0, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 0];
[c, 1, 1, 0, 1, 1, 1, 1] -> [0, 1, 1, 1, 1, 1, 0];

```

"

"MCU reads column inputs and scanning outputs, determined key (1,1) has been

"closed. Later key (1,1) is released, intr becomes inactive and scanning resumes

```

[c, 1, 1, 1, 1, 1, 1, 1] -> [1, 0, 1, 1, 1, 1, 1];
[c, 1, 1, 1, 1, 1, 1, 1] -> [1, 1, 0, 1, 1, 1, 1];

```

END

## Appendix C. KEYPAD.STL File

```

//auto scanning simulation
//start scanning, press key, read port A (column) and port B (row)

//+++++
// Defining tasks to simplify the stimulus file
//+++++

task write (addr_bus,bhe_value,data_in); // 80196 write bus cycle

input [15:0] addr_bus;
input [15:0] data_in;
input bhe_value;

begin

#20 ale = 1; //Latch the address lines
#20 adio = addr_bus; //Set-up the right address
 bhe = bhe_value;
#20 ale = 0; //Ale inactive

#20 adio = data_in; // Write operation

#40 wr = 0; // Write pulse
#100 wr = 1; // Write ends
#10 adio = Z16; bhe = Z;

end

endtask

task read (addr_bus); //80196 read bus cycle

input [15:0] addr_bus;

begin

#20 ale = 1; //Latch the address lines
#20 adio = addr_bus; //Set-up the right address
 bhe = 0;
#20 ale = 0; //Ale inactive

#20 adio = Z16; // Float Address bus

#40 rd = 0; // Rd pulse
#100 rd = 1; // Rd ends
#10 bhe = Z;

end

endtask

reg [4:0] column;
assign {col4, col3, col2, col1, col0} = column;
assign {row4, row3, row2, row1, row0} = row;
reg intr, frez;
initial

```

**Appendix C.**  
**KEYPAD.STL**  
**File**  
**(Cont.)**

```

begin
 rst = 0; //generate reset
 wr = 1; rd = 1; //initialize control signal
 ale = 0; bhe = 1;
 adio = 16'bz; //initialize addr/data bus
 intr = 'bz; frez = 'bz;
 clkln = 0; pd5 = 0; pd6 = 0; pd7 = 0; //init not used port pins
 pa5 = 0; pa6 = 0; pa7 = 0;
 row = 5'bz;
 column = 5'b11111;
 csi = 0; //set PSD5XX chip select low

#300 rst = 1; //after 500ns, rst inactive

//write and read to the sram, verify bus interface is ok
write ('h8476,0,'h5a27);

//read sram,word
read ('h8476); //Word-read operation

//write Port A Control Register, configure Port as I/O
write ('hC002,1,'hff);

//write Port A Direction Register, configure Port A as input
write ('hC006,1,'h00);

#635 column = 'b11101; //press key (3,1) -- row 3, column 1
 //state machine is frozen
 //intr is generated to the MCU

//MCU reads Port A Data In Reg. (column inputs)
read ('hc000);

//MCU reads Port B Macrocell Out Reg. (state machine row pattern)
read ('hc00d);

#500 column = 'b11111; //key is released
 state machine resumes operation

end

always
#200 clkln = ~clkln;

```





# **Programmable Peripheral Application Note 035**

## **How To Design With The PSD4XX/5XX ZPLD**

*By Dan Friedman*

---

### **Abstract**

The PSD4XX and PSD5XX programmable MCU peripherals both contain a Zero-power PLD (ZPLD) array. Below are several tips, information and procedures for working with the ZPLD.

---

### **Generate Address A7-A2 as ZPLD Input**

Address lines A7-A2 are not routed directly into the ZPLDs. They can be routed into the ZPLD by configuring an I/O port to the Address Out Mode and routing these signals into the ZPLDs. Listed below is the method of implementing this function.

#### ***For a Multiplexed MCU***

In PSDabel define 6 inputs (Port C for example):

addr7, addr6, addr5, addr4, addr3, addr2 pin 10, 11, 12, 13, 14, 15;

I/O port pins 7-2 must be used. These input signals then can then be used in your logic equations in PSDabel.

In the initialization software executed by the MCU;

Initialize Port C to the Address Out Mode by writing 00H (actually 0000 00XX in binary) to the Configuration Register. The default condition, after reset, of the Configuration Register is 00H.

Write FFH (actually 1111 11XX in binary) to the Direction Register.  
The default condition, after reset, of the Direction Register is 00H.

The address signals A7-A2 will always appear on Port C7-2 and will be routed to the ZPLDs.

#### ***For a Non-multiplexed MCU***

Route Address lines A7-A2 to any unused I/O port pins.

In PSDabel define 6 inputs (Port A for example):

addr7, addr6, addr5, addr4, addr3, addr2 pin 10, 11, 12, 13, 14, 15;

These input signals then can then be used in your logic equations in PSDabel.

There is no software initialization. Any Port A I/O port pins can be used in this example.

**Load Data  
D7–D0 to  
Macrocells**

On a multiplexed MCU, the data bus is not routed into the GPLD. This application note discusses how to write data from the data bus on the MCU to the macrocells inside the General PLD (GPLD) inside the PSD4XX/5XX parts. Three methods are discussed in this application note. The detailed implementation of each of these methods can be found in application note 034 called “Loading Data into the PSD4XX/5XX GPLD Macrocells”. This application note can be found on the WSI bulletin board. The file name is “appnote34.zip”.

**Method 1**

The MCU writes the lower 4 bits of data into the 4-bit page register. It then writes to an arbitrary address to generate a clock input to the macrocells to transfer the data from the page register to 4 of the macrocells. The MCU repeats this process for the upper 4 bits of data.

**Example:** Transfer A5H from the MCU to 8 macrocells (Port B) inside the GPLD. Assume CSIOP is defined from 2000H to 20FFH. Assume the arbitrary addresses the macrocells are mapped into are 2100H to 21FFH and these 256 bytes are used for this address range because the resolution of address decoding is 256 bytes. This address range cannot be used by anything else in the system.

**Step 1:**

The MCU will write X5H to memory location 20E0H. This is the location of the page register inside the PSD4XX and PSD5XX devices. “X” address means “don’t care”.

**Step 2:**

The MCU will write XXH (don’t care condition) to memory location 2100H. This will generate a clock input to the macrocells (clock defined as “^h2100 & !wr”) and will transfer the 4 bits of data in the page register to the four least significant bit macrocells.

**Step 3:**

The MCU will write XAH to memory location 20E0H.

**Step 4:**

The MCU will write XXH to memory location 2101H. This will generate a clock input to the macrocells (clock defined as “^h2100 & !wr”) and will transfer the 4 bits of data from the page register to the most significant bit macrocells.

**Step 5:**

The value of the Port B Macrocells can be read from the Macrocell Out Register of Port B at 200DH.

---

**Load Data  
D7–D0 to  
Macrocells  
(Cont.)****Method 2**

The processor will write data to an I/O port. The data on the I/O port will be routed back into the GPLD and latched into the macrocells. No external signal routing is required to route the output port back into the GPLD. The MCU will write the data to an I/O port. The data is transferred to the macrocells when the MCU generates a clock input to the macrocells by writing to an arbitrary address.

**Example:** Transfer A5H from the MCU to 8 macrocells (Port B) inside the GPLD. Assume CS1OP is defined from 2000H to 20FFH. Any I/O port can be used on the PSD4XXA2 and the PSD5XXB1. On the PSD4XXA1, Ports A or B must be used. Port C will be used in this example. Assume that the arbitrary address that the macrocells are mapped into is from 2100H to 21FFH. 256 bytes are used for this address range because the resolution of address decoding is 256 bytes. This address range cannot be used by anything else in the system.

**Step 1:**

The MCU writes FFH to memory location 2012H to the Control Register. This will change Port C to the MCU I/O Mode.

**Step 2:**

The MCU writes FFH to memory location 2016H to set Port C I/O port pins to all outputs.

**Step 3:**

The MCU writes A5H to memory location 2014H to latch the data out on Port C.

**Step 4:**

The MCU generates a clock input (defined as “^h2100 & !wr”) to the macrocells by writing XXH (don’t care condition) to memory location 2100H to latch the data on Port C to the internal macrocells in Port B.

**Step 5:**

The value of the Port B Macrocells can be read from the Macrocell Out Register at 200DH.



## Load Data D7–D0 to Macrocells (Cont.)

### Method 3

This method will use the individual Preset and Reset signals from Port B to initialize (or load) data into the Port B Macrocells. An active Preset will load a logical “1” into the corresponding macrocell and an active Reset will load a logical “0”. Each Preset/Reset occupies an address and is activated when the MCU writes to that address.

**Example:** Transfer A5H from the MCU to 8 macrocells in Port B. Assume Preset/Reset of the macrocells occupy address 2100H to 24FFH. For example,

|                       |                       |
|-----------------------|-----------------------|
| PB7.RE = ^h2402 & !WR | PB7.PR = ^h2403 & !WR |
| PB6.RE = ^h2400 & !WR | PB6.PR = ^h2401 & !WR |
| PB5.RE = ^h2302 & !WR | PB5.PR = ^h2303 & !WR |
| PB4.RE = ^h2300 & !WR | PB4.PR = ^h2301 & !WR |
| PB3.RE = ^h2202 & !WR | PB3.PR = ^h2203 & !WR |
| PB2.RE = ^h2200 & !WR | PB2.PR = ^h2201 & !WR |
| PB1.RE = ^h2102 & !WR | PB1.PR = ^h2103 & !WR |
| PB0.RE = ^h2100 & !WR | PB0.PR = ^h2101 & !WR |

#### Step 1:

The MCU writes XXH (a don't care condition) to 2403H. This will set Port B Macrocell PB7 to a logic 1.

#### Step 2:

The MCU writes XXH (a don't care condition) to 2400H. This will set Port B Macrocell PB6 to a logic 0.

#### Step 3:

The MCU writes XXH (a don't care condition) to 2303H. This will set Port B Macrocell PB5 to a logic 1.

#### Step 4:

The MCU writes XXH (a don't care condition) to 2300H. This will set Port B Macrocell PB4 to a logic 0.

#### Step 5:

The MCU writes XXH (a don't care condition) to 2203H. This will set Port B Macrocell PB3 to a logic 0.

#### Step 6:

The MCU writes XXH (a don't care condition) to 2200H. This will set Port B Macrocell PB2 to a logic 1.

#### Step 7:

The MCU writes XXH (a don't care condition) to 2103H. This will set Port B Macrocell PB1 to a logic 0.

#### Step 8:

The MCU writes XXH (a don't care condition) to 2100H. This will set Port B Macrocell PB0 to a logic 1.

#### Step 9:

The value of the Port B Macrocells can be read from the Macrocell Out Register at location 200DH.

The method best to use depends on the resource still available after implementing the rest of the design. If speed is critical, Method 2 will execute the fastest. One write cycle can be achieved by using Method 2 and routing the data bus to the I/O Port.

## **Use a Macrocell to Latch External Data/Status and Read with an MCU**

When the I/O ports are configured as an input port in the MCU I/O Mode, the input pins are sampled by the MCU. In some designs it is desirable to latch the data. This data is latched by an external strobe signal. The GPLD macrocells can be used to latch data from an external source with an external strobe signal and have the MCU read this latched data.

In the PSDLabel file specify the following:

module example

“input data

din7, din6, din5, din4, din3, din2, din1, din0 pin;

“data flip-flops containing the latched data in

data7, data6, data5, data4, data3, data2, data1, data0 node istype ‘reg’;

“strobe or clock signal to latch the data into 8 macrocells.

strobe pin;

“DEFINITIONS

data\_in = [din7,din6,din5,din4,din3,din2,din1,din0];

latch\_data = [data7,data6,data5,data4,data3,data2,data1,data0 ];

EQUATIONS

latch\_data := data\_in;

latch\_data.c = strobe;

end

The MCU can read the latched data by reading the Macrocell Out Register. There is a Macrocell Out Register for Port A, B, and E. The address locations are specified in the Systems Configuration section of the WSI “PSD Programmable Peripherals Design and Applications Handbook”.

## **Use Macrocells to Latch MCU Data and Read with a Co-processor**

The MCU can use the macrocells to latch data out on an I/O port. The Output Enable Control of this I/O port can be controlled by an external device such as a co-processor. If you are trying to pass data from the MCU to a co-processor, connect an I/O port directly to the co-processor's data bus. The Output Control of this I/O port will be controlled by the co-processor thus avoiding any conflicts on the co-processor's data bus. The MCU will load data into 8 macrocells and those macrocells will be routed to the I/O port connected to the co-processor's data bus. The method of loading data into the macrocells is described in section 2.0 of this Application Note.

In the PSDabel file specify the following:

Method 1 was used from Section 2.0 to load data into the macrocells from the MCU.

module example

```
"define the page reg
pgr3, pgr2, pgr1, pgr0 node;
```

```
"define the macrocells to latch the data out
data7, data6, data5, data4 pin istype 'reg';
data3, data2, data1, data0 pin istype 'reg';
```

"This signal is the output enable signal from the co-processor.

"This signal will enable the output of the I/O port.

```
proc_enable pin;
```

"DEFINITIONS

```
page_reg = [pgr3,pgr2,pgr1,pgr0];
```

"Since the page register is only 4 bits wide, the byte of data must

"be split into two nibbles.

```
upper_nibble = [data7, data6, data5, data4];
```

```
lower_nibble = [data3, data2, data1, data0];
```

EQUATIONS

"Because of the resolution of the address decoding (from A15 to A8),

"the address range from ^h2000 to ^h20FF is reserved for loading the

"macrocells with data from the page register.

```
lower_nibble.oe = proc_enable;
```

```
lower_nibble.c = (Address == ^h2000) & !wr;
```

```
lower_nibble := page_reg;
```

```
upper_nibble.oe = proc_enable;
```

```
upper_nibble.c = (Address == ^h2001) & !wr;
```

```
upper_nibble := page_reg;
```

```
end
```

## **Generate Reset, Preset, Clock and Output Enable Inputs to the Macrocells**

Port B macrocells are the most flexible macrocells. Each macrocell on Port B can have individual preset, reset, and clock product terms. The reset, preset, and output enable signals for Port A and E are grouped together. The clock input for all macrocells associated with Port A and E comes from the clk<sub>in</sub> signal. The Reset, Preset, and Output Enable input signals to the macrocells are all active high. The following definitions are available for each port.

### Port A

boo\_a.re “same reset product term for all 8 macrocells  
 boo\_a.pr “same preset product term for all 8 macrocells  
 boo\_a.c “same clock input (clk<sub>in</sub>) for all 8 macrocells  
 boo\_a.oe “same output enable product term for all 8 macrocells

### Port B

boo\_ai.re “individual reset product term for each of the 8 macrocells  
 boo\_ai.pr “individual preset product term for each of the 8 macrocells  
 boo\_ai.c “individual clock product term or the clk<sub>in</sub> signal for each of the 8 macrocells  
 boo\_ai.oe “individual output enable product term for each of the 8 macrocells

### Port E

boo\_e.re “same reset product term for all 8 macrocells  
 boo\_e.pr “same preset product term for all 8 macrocells  
 boo\_e.c “same clock input (clk<sub>in</sub>) for all 8 macrocells  
 boo\_e.oe “same output enable product term for all 8 macrocells

Listed below are the most powerful to the least powerful macrocells:

Port B macrocells  
 Port A macrocells  
 Port E macrocells

3

## **Macrocells Implement Buried PLD Function, Port Configured as MCU I/O**

If an I/O port cell is configured as a MCU I/O, the associated macrocell can still be used as a buried feedback macrocell.

### module example

“example of a two bit shift register

iosignal pin 50; “Port B0 is used as a general MCU I/O port, reserving this pin.

burried\_mc node 50 istype ‘reg’; “The macrocell associated with Port B0.

data\_in pin; “input data to be shifted in.

data\_out pin istype ‘reg’; “output of shift register

### equations

burried\_mc.c = clk<sub>in</sub>;

burried\_mc := data\_in;

data\_out.c = clk<sub>in</sub>;

data\_out := burried\_mc.fb;

end





# Programmable Peripheral Application Note 036 How To Fit Your Design Into The PSD4XX/5XX

By Dan Friedman

---

## Abstract

This application brief is a step-by-step procedure for fitting your design. This is not the only method of fitting your design but is an effective one.

---

## Method of Fitting Your Design

### Step 1

When specifying a new project name, specify a project directory under the PSDsoft directory.

### Step 2

Copy an old PSDlabel (.ABL) file from a previous project into your project directory. There are some examples of PSDlabel files in the "C:\psdsoft\examples" directory.

### Step 3

When declaring input and output signals, do not specify pin assignments. After the design fits, add pin assignments and move signals to more desired pin locations.

### Step 4

Any signals used as Standard MCU I/O or Latched Address Out can be declared in the PSDlabel file. As long as these signals are not used in the PLD equations, they will default as Standard MCU I/O or Latched Address Out. There are some exceptions to this rule. These exceptions relate to Bus Interface Signals, Special Function Signals, and Alternate Function Signals.

### Step 5

Compile the design in PSDlabel Design Entry.

### Step 6

After eliminating all syntax errors, view Optimized Equations. For a given signal, there are two numbers indicating the number of product terms (one from the Default Polarity and one from the Reverse Polarity). The fitter will always use the lowest number of the two columns. If the lowest number for a given signal is greater than 6, this signal will not fit. To solve this problem, break up the product terms and use a buried register as shown below.

#### before:

```
boo pin; "Output signal
a,b,c,d,e,f,g,h,i,j pin; "Input signals
equations
boo = (a & b) # (c & d) # (e & f) # g # h # i # j; "This equation uses 7 product terms.
```

#### after:

```
boo pin; "Output signal
a,b,c,d,e,f,g,h,i,j pin; "Input signals
buried_reg node; "Intermediate term
equations
buried_reg = (a & b) # (c & d) # (e & f);
boo = buried_reg.fb # g # h # i # j;
```

The above example shows one method of manually performing product term expansion. However, PSDsoft automatically performs product term expansion based on the available device resource. When PSDsoft automatically performs product term expansion, see the Fitter report for detailed information.

**Method of  
Fitting Your  
Design  
(Cont.)**

**Step 6 (Cont.)**

**Note:** Product terms for ES0-3 and RS0 for internal EPROM and SRAM respectively have only one product term as defined in the Decoding PLD (DPLD). If more product terms are required, the above method (using a buried macrocell from the GPLD) can be used.

**Note:** General PLD (GPLD) features for the PSD4XXA2 and PSD5XXB1 products:

- Port A Macrocells have 3 product terms each.
- Port B Macrocells have 6 product terms each.
- Port E Macrocells have 1 product term each.
- All Port B Macrocells can use the *clk<sub>in</sub>* signal or product term clocks (clocks other than the *clk<sub>in</sub>* signal). These product term clocks can come from any I/O port pin. Each Port B Macrocell can have individual product term clocks.
- Port A and E Macrocells are clocked by the *clk<sub>in</sub>* signal only.
- All Port B Macrocells have individual preset, reset, and output enable product terms.
- Port A and E Macrocells have common preset, reset, and output enable product terms.

**Note:** GPLD features for the PSD4XXA1 products:

- Port A Macrocells have 3 product terms each.
- Port B Macrocells have 6 product terms each.
- Port E Macrocells do not exist
- All Port B Macrocells can use the *clk<sub>in</sub>* signal or product term clocks (clocks other than the *clk<sub>in</sub>* signal). These product term clocks can come from any I/O port pin. Each Port B Macrocell can have individual product term clocks.
- Port A Macrocells do not contain flip-flops. They are combinational outputs only. Therefore there is no clock, preset, or reset inputs to these macrocells.
- All Port B Macrocells have individual preset, reset, and output enable product terms.
- Port A Macrocells have common output enable product terms.
- Port C, Port D, and Port E (PE2-7 only) are not routed into the PLDs. Therefore, inputs on these port pins can not be used as part of the PLD logic equations.

In the Optimized Equations Report, determine the number of signals requiring 4 to 6 product terms and 2 to 3 product terms. If there are more than 8 signals requiring 4 to 6 product terms, the design will not fit. If there are more than 16 signals requiring 2 to 6 product terms, the design will not fit. By splitting up the product terms and using a buried register as described above, this problem can be solved.

Designs often do not fit because the designer has defined too many product term clocks (clocks other than the *clk<sub>in</sub>* signal). Two methods of working around this problem are shown below.

**Method of  
Fitting Your  
Design  
(Cont.)**

**Step 6 (Cont.)**

**Method 1.** Take the largest group of signals associated with a clock and route that clock into the **clk**in pin.

**Example:** Make the following list on paper.

| <u>Clock W</u> | <u>Clock X</u> | <u>Clock Y</u> | <u>Clock Z</u> |
|----------------|----------------|----------------|----------------|
| sig 1          | sig 4          | sig 8          | sig 10         |
| sig 2          | sig 5          | sig 9          |                |
| sig 3          | sig 6          |                |                |
|                | sig 7          |                |                |

If the designer is using a PSD4XXA2, route Clock X into the **clk**in pin.  
Route sig4, sig5, sig6, sig7 to port A or E macrocells.  
Route sig1, sig2, sig3, sig8, sig9, sig10 to port B macrocells.

**Method 2.** If the designer is using the following definition,

```
x:=1;
x.re = !reset
x.clk = A&B;
```

Convert the above function as follows:

```
x.re = !reset;
x:= x.fb # (A&B);
x.clk = clk;
period of clk < pulse width generated by (A&B)
```

**Note:** The number of signals with 4 to 6 product terms plus the number of signals requiring a product term clock with less than 4 product terms on the D input of the flip-flop cannot exceed 8.



**Method of Fitting Your Design (Cont.)**

**Step 7**

Fit the design in PSD compiler under the Compile Menu.

To understand why a signal does not fit, look at the Report File under the View Menu. Look at the Resource Usage Summary along with the OMC Resource Assignment. The Resource Usage Summary will tell the designer how the pins on a given I/O Port were assigned and how those resources were allocated. The OMC Resource Assignment will indicate which macrocell was utilized for each output signal used in an equation in the PSDabel file.

**Example:**

|                       |                                | <b>OMC Resource Assignment</b> |                                                                                                              |
|-----------------------|--------------------------------|--------------------------------|--------------------------------------------------------------------------------------------------------------|
| <b>Resources Used</b> |                                | <b>User Name</b>               |                                                                                                              |
| <b>Port A :</b>       |                                |                                |                                                                                                              |
| macro cell 3          | cnt0 => Register               |                                | “cnt0 used as a buried register.<br>“cnt0 was defined as a node.<br>“cnt0 is a registered output node.       |
| <b>Port B :</b>       |                                |                                |                                                                                                              |
| macro cell 7          | cnt4 (mc_pb7) => Register      |                                | “cnt4 is routed to an output pin on “PB7.<br>“cnt4 was defined as a pin.<br>“cnt4 is a registered output.    |
| <b>Port E :</b>       |                                |                                |                                                                                                              |
| macro cell 2          | wstc (mc_pe2) => Combinatorial |                                | “wstc is routed to an output pin “on PE2.<br>“wstc was defined as a pin.<br>“wstc is a combinatorial output. |

All signals followed by “(mc\_pxx)” are output pins. If “(mc\_pxx)” is omitted, the signal was defined as a node and is a buried register. From this report the designer can determine the exact reason why a given signal would not fit.

**Note:** In the Options menu, Fitter Options are Keep, Try, or Ignore.

Keep Current – Uses the pin assignments specified in the PSDabel file.

Keep Previous – Uses the pin assignment from the previous fitting process.

Try – Tries to use the pin assignments specified in the PSDabel file.

Ignore – Does not use the pin assignments specified in the PSDabel file.

This is the same as not specifying any pin assignments in the PSDabel file. For pins which use reserved names, the pin assignments are always fixed.

---

**Method of  
Fitting Your  
Design  
(Cont.)****Step 8**

If several signals will not fit, start by commenting out all unfitted signals until the design fits. Fit one signal at a time by using some of the above methods and other methods described in other Application Notes.

**Step 9**

Assign pin numbers to all the signals in the PSDabel file. Move signals around to desired pin numbers. The designer may not be able to move certain signals to desired pin numbers as a design violation may occur.

**Note:** Some important things to remember about the PSD4XXA2 and PSD5XXB1 devices are that only Ports A, B, and E have PLD I/O. Port C and D are PLD inputs only. If Port A, B, or E is used as a PLD input, the macrocell associated with that pin cannot be used as a buried register or routed to an output pin. It is best to use Port C and D and PLD inputs first. On the PSD4XXA1 devices, Ports A and B can be used as PLD I/O while port pins E0 and E1 can be used as PLD inputs only. **Any signal pins reserved in the PSDabel file that are used as Latched Address Out Signals must be in sequential order (i.e., addr0 must be assigned to PC0, addr1 must be assigned to PC1 etc.).**





# Programmable Peripheral Application Note 037

## How to Implement a Latch Function in Port A of PSD4XX/5XX that is Independent of the System Clock

By Mohan Maghera

### Introduction

The macrocells in PSD4XX/5XX devices include D-type registers. When mapping discrete solutions to these PSDs, it is sometimes necessary to replace transparent latches (e.g., '573) with the PSD macrocells. Since the PSDs do not have transparent latches, the easiest alternative is to make the design edge-triggered and use the D-type registers. However, there are some situations where the designer must use a transparent latch. In these cases it is possible to use a 2:1 multiplexer configured to perform the function of a latch.

There is an added bonus in using this approach: the PSD4XX/5XX devices offer up to 24 macrocells in the GPLD. Of these, 16 Port-A and Port-E macrocells are clocked by the system clock on the CLKIN pin. The other 8 (Port-B) macrocells may be individually configured to use either the system clock or a product term clock.

For designs that fully utilize the Port-B macrocells and still need further register elements that must remain independent of the system clock (but do not have to be edge triggered), it is possible to realize up to 8 more registers by using 2:1 multiplexers configured as transparent latches.

Two examples are shown in this application note. The first shows the basic idea by realizing a latch with one Port-A macrocell, and the second example shows a "real life" situation where a one-way communications port (e.g., Centronics: host to target, where the PSD would be located in the target) is realized in the Port-A macrocells.

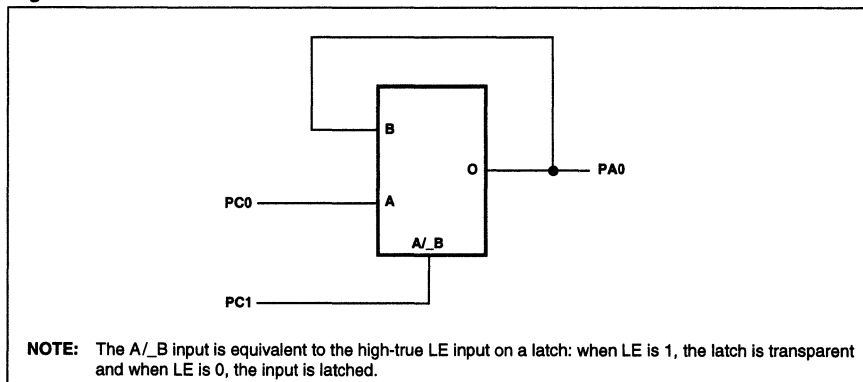
3

### Example 1

Figure 1 shows how one of the Port-A macrocells performs this latch function: Port-C has been used to input the signal to be latched, as well as the LE control signal. Of course, another port (or ports) may be used as long as it is usable as an input to the ZPLD-bus. The output of the macrocell may then be brought out to the respective Port-A pin, if needed. Otherwise, if it is to be accessed by the MCU, the pin may be kept free for some other I/O function since the MCU can access the outputs of the macrocells directly by reading the "Macrocell Out" register of Port-A.

When PC1 is HIGH, PC0 is enabled through to PA0 – Transparent.  
When PC1 is LOW, PA0 is looped back on itself – Latched.

**Figure 1.**



**Example 1**  
(Cont.)

Below is a sample ABEL file that describes this function:

**module latch1**

title 'transparent latch using a 2:1 mux.';

“Since the PSDs offer D\_type registers and not transparent latches, the easiest alternative for the designer is to make the design edge triggered and use the D-type registers. However, there are some situations where the designer must use a transparent latch. In these cases it is possible to use a 2:1 multiplexer configured to perform the function of a latch.

## “INPUTS and OUTPUTS

```
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
wr, rd pin;
psen, ale pin 38,37; "PE0-1
clkln, reset pin;
csi pin;
```

“PA0 acts as the latch output

```
pa0_latch_out pin 27;
```

“PC0 acts as the input to the latch, and PC1 as the latch enable signal.

```
pc0_latch_in, pc1_le pin 17, 16;
```

“base address for i/o chip selects

```
csiop node;
```

“ \*\*\*\*\*

## “DEFINITIONS

```
X = .x.;
CK = .c.;
addr=[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];
```

“ \*\*\*\*\*

## EQUATIONS

“DPLD equations

```
csiop = (addr >= ^h0C000) & (addr <= ^h0C0FF);
```

“GPLD equations

```
pa0_latch_out = (pc0_latch_in & pc1_le) "transparent"
 # (pa0_latch_out.fb & !pc1_le) "latched"
 # (pa0_latch_out.fb & pc0_latch_in); "removes any glitches"
```

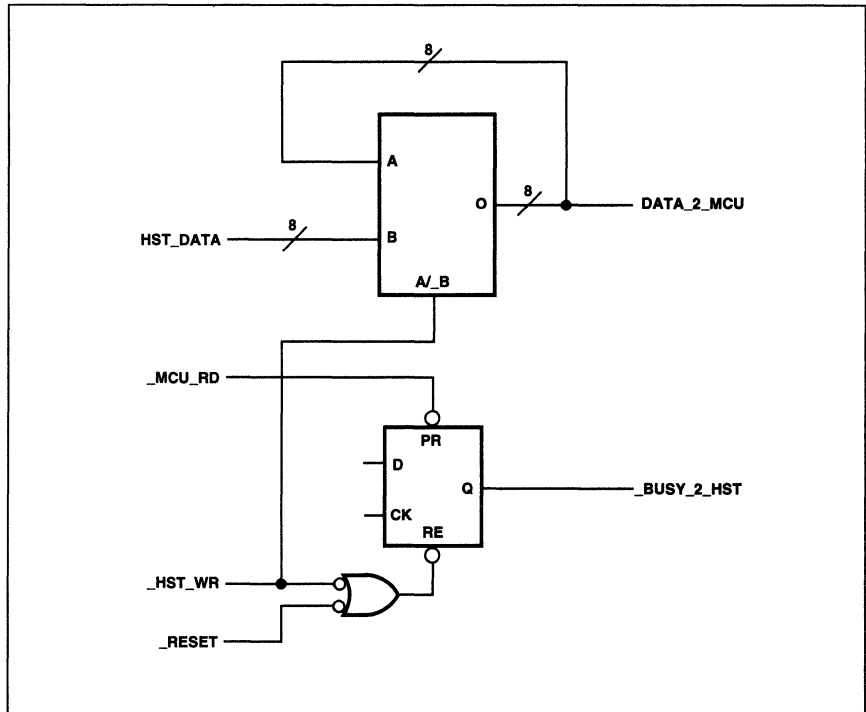
**end latch1**

**Example 2**

Figure 2 shows a communications port that allows a host to write data into an 8-bit register with the `_HST_WR` signal. Simultaneously, this signal is used to set a `_BUSY_2_HST` flag which is polled by the host to see if the MCU has read the data. When reading this data (`_MCU_RD`), the MCU clears the `_BUSY_2_HST` flag, thus indicating to the host that it may write the next data byte.

In a discrete solution the `_HST_WR` signal would be used as a clock to the D-type registers, but in this example it is assumed that the Port-B macrocells are used for functions that need `CLKIN` (system clock) and other independent (product term) clock inputs. In this situation the D-type registers in Port-A would be clocked by `CLKIN` and thus cannot be driven by the `_HST_WR` signal. Since the data is required to be stable when `_HST_WR` is High and is “Don’t Care” when `_HST_WR` is Low, we can replace the edge-triggered registers with a transparent latch function realised using 2:1 multiplexers.

Figure 2 also shows that the `_RESET` signal is ORed with `_HST_WR`. So, after a system reset it will be necessary for the MCU to do a dummy read of the data register to clear the busy flag. The reason for including this is to ensure that the host does not try and write to this port while the MCU is still in a reset cycle.

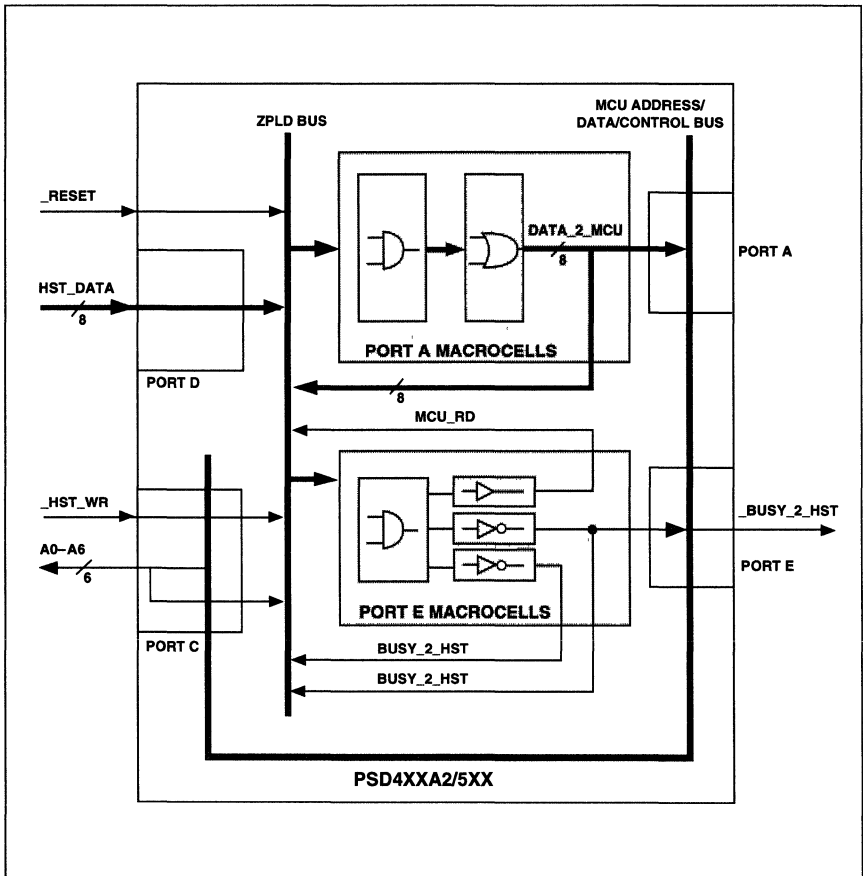
**Figure 2.**

**Example 2**  
(Cont.)

Figure 3 shows how the data register can be realised in the Port-A macrocells on the PSD. The `_HST_WR` flag is generated in the Port-E macrocells and avoids using the clock on the D-type register by implementing an S-R flip-flop using cross-coupled NAND functions (thus giving the same functionality as the preset and reset functions of a D-type register). The MCU would either use the `_HST_WR` signal's rising edge to generate an interrupt to indicate that a valid data byte is available, or would test for the `_BUSY_2_HST` flag being Low and `_HST_WR` signal being High (i.e., host write cycle is complete). When the MCU reads the data register (i.e., reads the "Macrocell Out" register of the Port-A macrocells), the `_BUSY_2_HST` register must be cleared. In order to do this, it is necessary to decode the full address of the Port-A Macrocell Out register ANDed with the MCU's `_RD` signal. In the PSD4XX/5XX, the address lines A8–A15 and A0–A1 are directly available on the ZPLD but in order to have access to the A2–A7 lines we must configure Port-C to output these latched addresses (on PC2-7). These Port-C pins are then available for decoding on the ZPLD-bus.

Since the busy flag will clear as soon as the `_RD` signal goes low, the host must avoid writing the next data byte too early, i.e., after seeing `_BUSY_2_HST` go High, it must insert a short delay equivalent to, or greater than, the `_RD` Low width before writing the next data byte.

**Figure 3.**



**Example 2**  
(Cont.)

Below is a sample ABEL file that describes this function:

**module latch2**

title 'One way comms. link: Host-to-MCU/PSD using Port-A macrocells configured as transparent latches';

"This example shows a simple, mono-directional communications link between a remote "host and a local MCU. The MCU uses the PSD to latch the incoming data and to generate "a busy flag back to the host.

"Since the PSDs offer D-type registers and not transparent latches, an octal 2:1 mux "realised in the Port-A macrocells is configured to perform the transparent latch function. "The design pre-supposes that the Port-B macrocells and the CLKIN pin are not available.

"                   \*\*\*\*\*

"INPUTS and OUTPUTS

"MCU interface signals (using a mixture of reserved names and explicit pin number "declarations)

```

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
wr, rd pin;
psen, ale pin 38,37; "PE0-1
clkln, reset pin;
csi pin;
```

"Port-A macrocell outputs (nodes) are reserved for the 2:1 mux-latch outputs

```

data_2_mcu7, data_2_mcu6, data_2_mcu5, data_2_mcu4,
data_2_mcu3, data_2_mcu2, data_2_mcu1, data_2_mcu0
node 20, 21, 22, 23, 24, 25, 26, 27;
```

"Port-C0 is used as the \_HST\_WR input pin  
\_hst\_wr pin 17;

"Port-C2-7 are configured to output latched addresses A2–A7 and these are fed back on to "the ZPLD-bus for use in decoding a read of the Port-A Macrocell Out register address "(= mcu\_rd = csiop + ^h0C for an Intel MCU design, and = csiop + ^h0D for a Motorola "16-bit MCU design). A0–1 are always available on the ZPLD-bus  
pc2, pc3, pc4, pc5, pc6, pc7 pin;

"Port-D is used to input the host data onto the ZPLD-bus

```

hst_data7, hst_data6, hst_data5, hst_data4,
hst_data3, hst_data2, hst_data1, hst_data0
pin 53, 54, 55, 56, 57, 58, 59, 60;
```

"Port-E2 macrocell and its pin is used as the \_BUSY\_2\_HST flag output  
"via one half of an S-R flip-flop (cross-coupled NAND gates)  
\_busy\_2\_hst pin 36;

"Port-E3 macrocell is used to generate the other half of the S-R flip-flop  
busy\_2\_hst node 34;



**Example 2**  
(Cont.)**Sample ABEL file (Cont.)**

“Port-E4 macrocell is used to decode MCU read of Port-A Macrocell Out Register address.

“This is necessary because Port-E macrocells can only support a single product term

```
mcu_rd node 33;
```

“base address for I/O chip selects — for this design it will be assumed that this

“address is ^hC000

```
csiop node;
```

```
“ *****
```

**“DEFINITIONS**

```
X = .x.;
```

```
CK = .c.;
```

```
addr=[a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];
```

```
full_addr = [a15,a14,a13,a12,a11,a10,a9,a8,
pc7,pc6,pc5,pc4,pc3,pc2,a1,a0];
```

```
data_2_mcu = [data_2_mcu7, data_2_mcu6, data_2_mcu5,
data_2_mcu4, data_2_mcu3, data_2_mcu2,
data_2_mcu1, data_2_mcu0];
```

```
hst_data = [hst_data7, hst_data6, hst_data5, hst_data4,
hst_data3, hst_data2, hst_data1, hst_data0];
```

```
“ *****
```

**EQUATIONS**

“DPLD

```
csiop = (addr >= ^h0C000) & (addr <= ^h0C0FF);
```

“GPLD

“realise the 8-bit latch

```
data_2_mcu = (hst_data & !_hst_wr) “transparent”
(data_2_mcu.fb & _hst_wr) “latched”
(data_2_mcu.fb & hst_data); “removes any glitches”
```

“busy flag

```
mcu_rd = (full_addr == ^h0C00C) & !rd;
_busy_2_hst = !(mcu_rd & busy_2_hst.fb);
```

```
busy_2_hst = !(_hst_wr & reset & _busy_2_hst.fb);
```

**end latch2**



# Programmable Peripheral Application Note 038

## How to Increase the Speed of the PSD5XX Counter/Timers

By Mohan Maghera

---

### Introduction

The PSD5XX family is presently the most capable programmable peripheral family that WSI produces. Among the standard features, such as EPROM, SRAM, I/O port expansion, Decode PLD (DPLD) and General Purpose PLD (GPLD), it also offers the designer a third PLD area known as the Peripheral PLD, four 16-bit counter/timer units (CTUs), and an 8-bit Interrupt Control Unit (ICU).

The PSD5XX has four 16-bit counter/timer units (CTUs) and this application note will examine the CTU block with respect to enhancing the speed of its operation (up to 28 MHz).

---

### CTU Block

All four CTUs work off the same clock source: CLKIN, the system clock. Before this clock goes to the CTUs, it passes through a pre-scaler that divides the system clock by a programmable value between 4 and 280. The maximum frequency of the CLKIN input to the pre-scaler is 28MHz. If the pre-scaler is set to divide by the minimum value of 4, the maximum frequency of operation of the CTU is 7MHz. However, there are many applications where it is required to count at much higher frequencies.

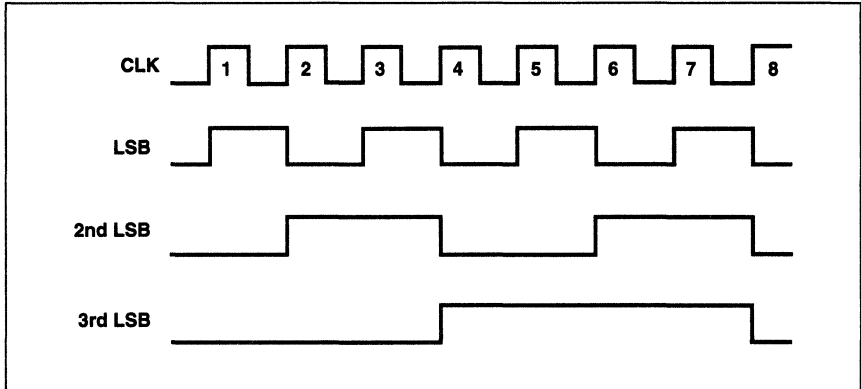
The GPLD on the PSD5XX-90 (90 nanosecond device), when used in synchronous clock mode (i.e., CLKIN is used as the clock input for the macrocell flip-flops), is capable of supporting internal feedback signals at frequencies up to 37.3 MHz, and when it is used in asynchronous clock mode (i.e., a product term clock is used for the macrocell flip-flops), the GPLD is capable of supporting internal feedback signals at frequencies up to 28.5 MHz.

If the design requires counter sizes of 5-bits or less (where 5-bits is the maximum size of a pre-loadable counter with count enable which can be realised in the GPLD Port-B macrocells without resorting to product term expansion), then it is possible to achieve counter frequencies of 37.5 MHz with CLKIN and 28.5 MHz with a product term clock. However, for those situations where the counter needs to be larger, it is possible to build such a counter from a CTU and the GPLD that operates at a much higher frequency than 7 MHz.

In any counter, the least significant bits are the ones that change the fastest and, therefore, need the faster clock. The least significant bit (LSB) changes state with every input clock cycle, the second LSB changes state with every second clock cycle, the third LSB with every fourth clock cycle, etc. (See Figure 1).

**CTU Block  
(Cont.)**

**Figure 1. Counter Output Waveforms**



If we examine the waveform produced by the outputs of the counter, we see the LSB produces a waveform at half the counter clock frequency ( $F_{cnt}$ ), the second LSB at a quarter of  $F_{cnt}$ , the third LSB at an eighth of  $F_{cnt}$ , etc. The relationship being:

$$\frac{F_{cnt}}{2^n}$$

where  $n = 1, 2, 3$ , etc., i.e., the position of the bit.

From this relationship, if we were to realise the least significant part of the counter in the GPLD, and from this generate a terminal count that could be used to gate one of the 16-bit CTUs, then the pre-scaled clock to the CTU need only be a fraction of the frequency used for the GPLD part of the counter. (See Figure 2.)

In order for the CTU to function correctly, it needs to be configured to run in the EVENT counter mode. This means that when the GPLD counter generates a terminal count, the positive going edge of this signal is latched as an event, and the CTU will be updated at the next CTU clock. For another event to be counted by the CTU, the terminal count of the GPLD counter must generate another rising edge, i.e., it must go low and back high again. Thus, the CTU clock must operate at a frequency above the events that are occurring to ensure that no events are missed and still satisfy the requirement that it remain below 7MHz., i.e.,

$$CTU\ clock = (CLKIN/pre-scaler\ value) \leq 7\ MHz$$

The relationship between CLKIN,  $F_{cnt}$ , pre-scaler value and the size of the GPLD counter ( $2^n$ ) is given by:

$$\frac{F_{cnt}}{(2^n)} < (CLKIN/pre-scaler\ value) \leq 7MHz$$

This is true for all  $F_{cnt}$  frequencies up to 28.5 MHz and CLKIN frequencies up to 28 MHz.

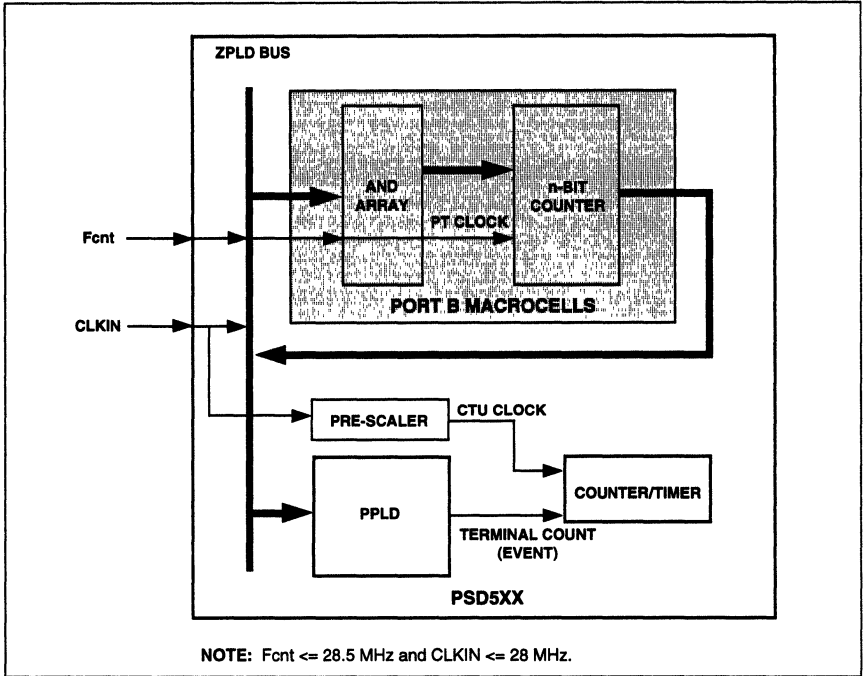
When  $F_{cnt}$  is the same clock as CLKIN (see Figure 3.), this relationship can be expressed as:

$$\frac{CLKIN}{(2^n)} < (CLKIN/pre-scaler\ value) \leq 7MHz$$

This is true for CLKIN frequencies up to 28MHz.

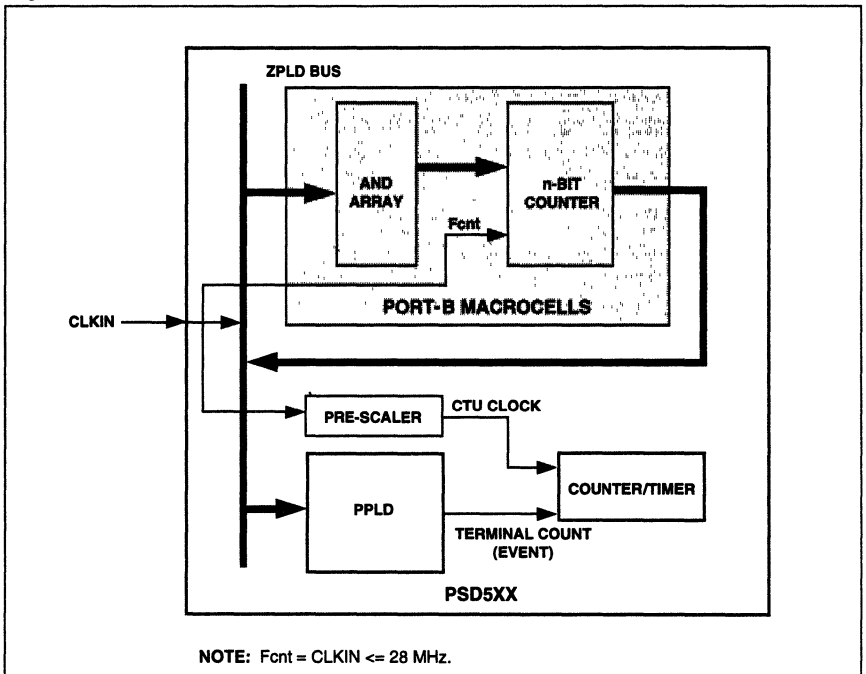
**CTU Block  
(Cont.)**

**Figure 2**



3

**Figure 3**



**CTU Block**  
(Cont.)

Below are sample ABEL and Verilog stimulus files for a design that needs a counter greater than 5 bits to run at a clock frequency up to 28 MHz. The Fcnt and CLKIN sources are the same, therefore, the relationship needed to be satisfied is:

$$\frac{\text{CLKIN}}{(2^n)} < (\text{CLKIN/pre-scaler value}) \leq 7 \text{ MHz}$$

$$\text{i.e., } \frac{28 \text{ MHz}}{(2^n)} < (28 \text{ MHz/pre-scaler value}) \leq 7 \text{ MHz}$$

This requires  $n$  to be at least 3 and the pre-scaler value to be between 4 and 7. In order to reduce the A.C. power consumption, it is best to use the biggest pre-scaler value possible (taking into account the clock frequency needs of the other three CTUs), i.e., 7 in this case.

The GPLD counter will be a 3-bit counter, whose terminal count is used to generate events to one of the 16-bit CTUs (CNTR0). The events (terminal counts) will occur at a frequency of  $(28/8) = 3.5$  MHz, and the CTUs will be clocked by  $(28/7) = 4$  MHz, which ensures that all events will be captured and counted.

The GPLD counter is cleared at power-up or with reset and will start to count only if the cnt\_en pin is held active (HIGH in this case). CNTR0, however, will need to be cleared by software by writing zero to it before it is enabled in the Command Register, CMD0, and in the Global Command Register (see stimulus file).

The GPLD counter in this example is made pre-loadable (cnt\_ld and din0-2) so that this, together with CNTR0, provides a 19-bit pre-loadable counter (CNTR0 is pre-loaded by writing the required upper 16-bit value to it before it is enabled in the CMD0 and Global Command Registers).

The terminal counts of the CTUs are available on Port-E and are also routed to the ICU to allow an interrupt to occur when a CTU reaches terminal count.

In order to read back the value of the complete 19-bit counter, the GPLD 3-bit counter outputs are available to the MCU via the Macrocell Output register. Assuming that all of the counter is realised in the Port-B macrocells, then a read of the Port-B Macrocell Output Register would access the counter bits. In order to ensure that the value does not change during the read cycle, it will be necessary to disable the counter (cnt\_en = LOW) before reading. To read back the 16-bit value of CNTR0, it is necessary to freeze the counter value by setting bit-0 in the Freeze Register and then polling bit-0 in the Freeze Acknowledge register until it reads 1. At this point CNTR0's value is transferred to the Image Register, IMG0. The value is then read from IMG0 while CNTR0 can continue counting (if cnt\_en is active). After IMG0 has been read, bit-0 in the Freeze Register should be reset to 0.

The CTUs in the PSD5XX can be used either in pulse or waveform modes, or in event count and time capture modes. Selection of these modes and the enabling of the CTUs to count are set via the command registers CMD0-3 and the Global Command Register. The pre-scaler value is set via a 5-bit value in the DLCY Register and a "scale bit" in the Global Command register. The order in which the various registers must be initialised and the values required for this example are given in the stimulus file.

**ABEL  
File****ABEL file:**

MODULE ctu\_spd  
title 'How to increase speed of CTU operation...';

## "INPUTS and OUTPUTS

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;  
wr, rd pin;  
psen, ale pin 38,37; "PE0-1  
clkln, reset pin;  
csi pin;

"base address for the PSD's internal I/O ports and  
"configuration registers  
csiop node;

## "GPLD 3-bit up counter signals

cnt\_en pin; "count enable input  
din2, din1, din0 pin; "counter data input pins  
cnt\_ld pin; "enable input for loading of  
"counter input data (din2-1)

"counter outputs  
cnt2, cnt1, cnt0 node istype 'reg';

"macrocell event (terminal count from GPLD counter  
"to CNTR0 – the CTU to be used for event counting  
mc2tmr0 node;

" \*\*\*\*\*

## "Definitions

X = .x;  
CK = .c;  
addr = [a15,a14,a13,a12,a11,a10,a9,a8,  
X,X,X,X,X,X,a1,a0];

din = [din2, din1, din0];  
cnt = [cnt2, cnt1, cnt0];

" \*\*\*\*\*

**ABEL  
File  
(Cont.)**

EQUATIONS

“DPLD equations

“I/O base address for PSD internal register — defined

“to be a 256 address block starting at ^hC000

csiop = (addr >= ^h0C000) & (addr <= ^h0C0FF);

“GPLD equations

cnt.clk = clkin;

cnt.re = reset;

WHEN (cnt\_ld) THEN cnt := din; “pre-load counter

ELSE WHEN (lcnt\_en) THEN cnt := cnt.fb; “counting is not enabled

ELSE cnt := (cnt.fb + 1) “increment count

“PPLD equations

“generate event for CNTR0 using the CTU macrocells in PPLD

“the terminal count is gated by the LOW part of clkin to

“ensure that no decoding spikes (after the rising edge of

“clkin) generate any false events

mc2tmr0 = (cnt.fb == ^h7) & !clkin;

END ctu\_spd

**VERILOG  
Stimulus  
File****VERILOG stimulus file:**

```
//Stimulus file for setting up the timer/counter, CNTR0,
//in event count mode, and for testing the 3-bit GPLD
//counter used to increase the speed of the PSD's
//Counter/timer units (CTUs).
```

```
reg [7:0] dat_val; //Used to hold data read from PSD
reg [7:0] din; //pre-load value for GPLD counter
assign {din2, din1, din0} = din;
```

```
//+++++
// User-Defined parameters
//+++++
```

```
parameter pb_mc_out='hC00D;
parameter cntr0L='hC098, cntr0H='hC099, img0L='hC090, img0H='hC091;
parameter cmd0='hC0A0;
parameter g_cmd='hC0A8;
parameter dicy='hC0A6;
parameter freeze='hC0A4, status='hC0A9;
```

```
//+++++
// Defining tasks to simplify writing the stimulus file
//+++++
```

```
task write (addr_bus,data_in);
```

```
input [15:0] addr_bus;
input [7:0] data_in;
```

```
begin
#20 ale = 1; //High true ale
#20 adio = addr_bus; //Set-up the right address
#20 ale = 0; //Latches address
#20 adio = data_in; //Write operation
#40 wr = 0; //Write pulse
#100 wr = 1; //Write ends
#10 adio = Z16;
end
```

```
endtask
```



**VERILOG**  
**Stimulus**  
**File**  
**(Cont.)**

```
task read (addr_bus);
input [15:0] addr_bus;

begin
#20 ale = 1; //Active high ale
#20 adio = addr_bus; //Set-up the right address
#20 ale = 0; //Latches address
#20 adio = Z16; //Float Address bus
#40 rd = 0; //Read starts
#50 dat_val = `adiol; //Store low byte of adio
#50 rd = 1; //Read ends
end
```

endtask

```
task psen (addr_bus);
input [15:0] addr_bus;

begin
#20 ale = 1; //Active high ale
#20 adio = addr_bus; //Set-up the right address
#20 ale = 0; //Latches address
#20 adio = Z16; //Float Address bus
#40 psen = 0; //Read starts
#100 psen = 1; //Read ends
end
```

endtask

//\*\*\*\*\* Begin stimulus \*\*\*\*\*

```
initial
begin
wr = 1; rd = 1; psen = 1;
ale = 0;
clkln = 0;
reset = 0;
csi = 0;
adio = `h0000;
din = `h0;
cnt_en = 0; //GPLD counter disabled
cnt_ld = 0;
```

```
#500 reset = 1;
```

//Clear timer/counter-0

```
#10 write(cntr0L, `h00);
write(cntr0H, `h00);
```

//Clear image register-0

```
write(img0L, `h00);
write(img0H, `h00);
```

**VERILOG**  
**Stimulus**  
**File**  
**(Cont.)**

```
//Set delay cycle register to 3, so that clkln is pre-scaled
//by 7 (pre-scale value = K(delay reg. + 4), where the
//scale bit, K, in the Global Command register is
//set to 0, i.e., scale factor is 1 – when set to 1 the scale
//factor would be 8)
write(dlcy, 'h03);

//Set command register, CMD0, to configure CNTR0 for event mode
//with the event coming from the macrocell, mc2tmr0

//LSB 0 Event count mode (if set to 1 = time capture mode)
// 1 Increment mode
// 1 Select (enable) CNTR0
// X No timer output in this mode
// X Pin-input polarity is not needed since the
// event is macrocell driven
// 0 Input command from macrocell
// (if set to 1 then from pin)
// 0 Load/Store command from Pin/Macrocell (in this
// case macrocell) allowed through
//MSB 0 Enable/Disable by Pin/Macrocell

write(cmd0,'h1E);

//Set the Global command register to enable the Counter/timers
//in event/time-capture mode

//LSB 0 Scale bit (0 = scale factor, k, is 1, 1 = scale factor is 8)
// 1 Counter start bit - enables all the selected counters
// 1 Global mode bit - set for event/time capture mode
// (if set to 0 then pulse/waveform mode is selected)
// 0 Watchdog disabled
//Bit4-7 are reserved and set to 0

write(g_cmd,'h06);

//Pre-load GPLD counter with 5
#10 din = 'h5;
 cnt_ld = 1;
#40 cnt_ld = 0; //Disable load after 1 clkln cycle
#10 din = 'h0;

//Enable counting

#10 cnt_en = 1;
```

**VERILOG  
Stimulus  
File  
(Cont.)**

```
//Disable GPLD counter and perform a
//freeze/freeze acknowledge cycle on CNTR0
 #2000 cnt_en = 0;
 write(freeze,'h01);

//Wait for freeze acknowledge flag to be set in
//status register (status = 'h01)
 dat_val = 'h00; //Clear temporary storage
 //register for read data
 while ((dat_val & 'h01) != 'h01) //Mask off CNTR0 Freeze
 //Acknowledge bit and test if set
 begin
 read(status); //Read Freeze Acknowledge
 //Status Register into dat_val.
 end

//Read GPLD counter outputs and CNTR0 value stored in IMG0
 read (pb_mc_out);
 read (img0L); //Low byte of Image Register
 read (img0H); // High byte of Image Register

//Reset freeze bit and enable GPLD counter
 #1000 write(freeze, 'h00);
 cnt_en = 1;

end

//***** Continuous signals *****

//Generate a continuous clock signal
always
 #18 clk_in = ~clk_in; // approximately 28MHz
```



# **Programmable Peripheral Application Note 039 Encoder for Shaft Direction and Position Recognition Using the PSD5XX**

*By Mohan Maghera*

---

## **Introduction**

In many applications the designer is provided with two input signals where one signal leads the other by some phase difference (perhaps 90 degrees, or even some variable amount). It is necessary to recognize which signal is leading and then either generate a pulse count from one of the signals (or some multiple, e.g., a pulse for each edge of the two signals – 4X clock), or be able to measure the phase difference between the two signals.

In a typical application the two signals are provided by a shaft encoder. These signals (A & B) are always 90 degrees out of phase. Depending on which signal is leading, it is possible to determine if the shaft is rotating clockwise or counterclockwise. By using a counter set to zero when the shaft is at the reference point and then counting up pulses (A and/or B) when the shaft is rotating clockwise and down pulses when rotating counterclockwise, it is possible to know the exact position of the shaft at any time from the value present in the counter. Integrated circuits are available on the market that input the two signals, perform the phase detection, and generate a direction signal (up/down), a 4x clock, and a 12- or 16-bit count value. These devices tend to be rather expensive and the designer is forced to integrate this function into an ASIC, or realize it in an EPLD, or some mixture of EPLD + discrete logic. The WSI, Inc. PSD5XX programmable MCU peripheral provides a space effective and optimal cost alternative to the designer – and at the same time providing EPROM, SRAM, interrupt control, chip selects and five I/O ports in one device.

Figure 1 shows the PSD5XX and the resources that are taken up by the inputs, outputs, state machine and counters.

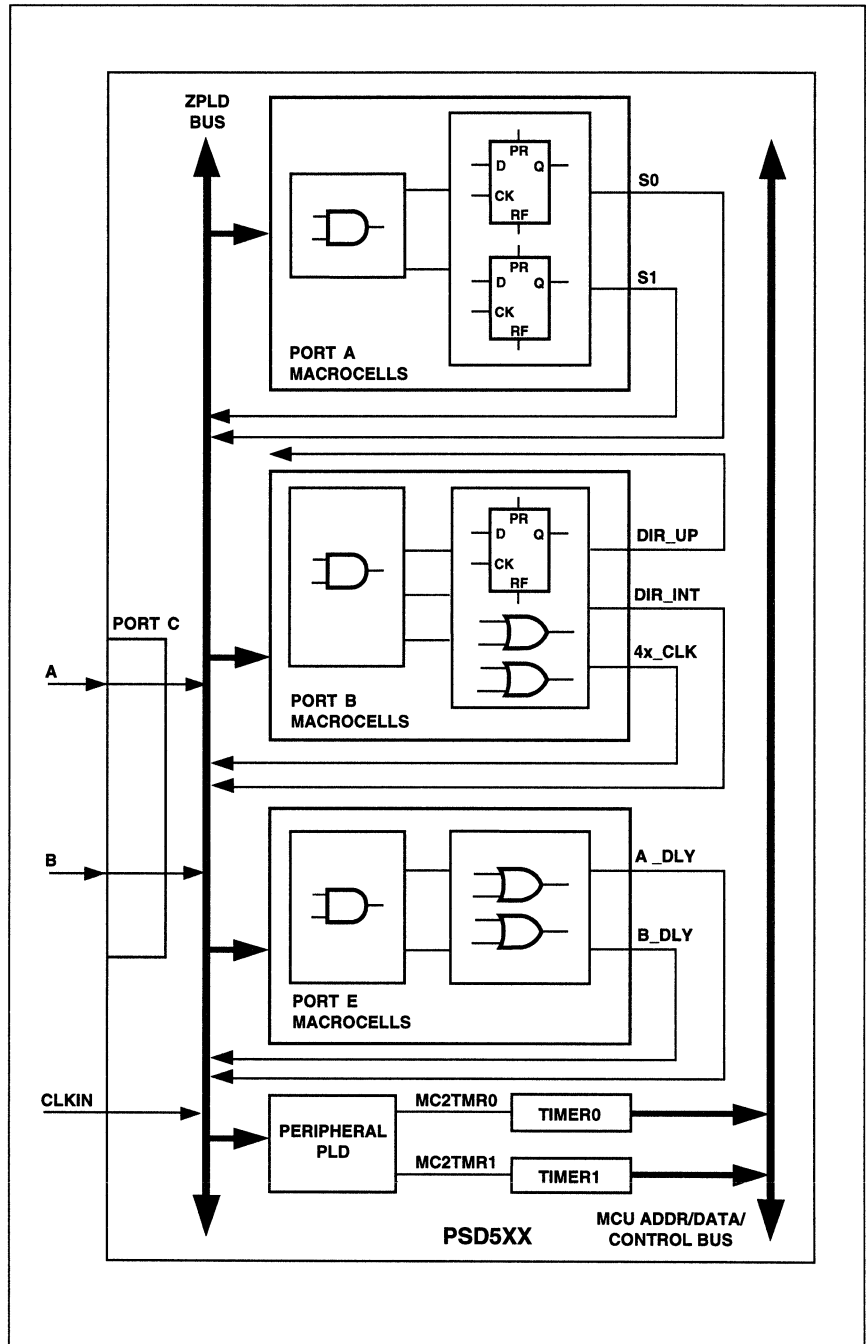
The DIR\_UP signal needs eight product terms (PTs) and the PSD5XX Port-B macrocells can handle a maximum of six PTs. It is necessary to break this down into two smaller PT groups: DIR\_INT (4PTs) and DIR\_UP (5PTs including DIR\_INT).

The state machine needs two bits (S0, S1) to cover the four possible states that can exist for the shaft encoder, and a 4X clock is generated (a pulse for every edge of A and B) in order to realize a finer position resolution. This clock is used to generate the events that the counter/timers count. The value of the DIR\_UP direction signal is used to gate the 4X clock to either the “Up” counter (TIMER0) or the “Down” counter (TIMER1).

The reason for using two counters is that when used in the event count mode, the counter/timers can only be used in the increment mode (up counting only). This requires that one counter is used for counting “Up Pulses” and a second counter used for counting “Down Pulses”. The actual position of the shaft is then the difference between the two counter values.

Below are the .abl and the .stl files that show how this design is realized. The software configuration necessary for the counters to operate in the event count mode is included, where the event is input via the macrocells (MC2TMR0 and MC2TMR1).

Figure 1.



**Abel File**

ABEL file:

**MODULE motr\_dir**  
**title 'Shaft encoder for motor direction and position**  
**recognition';**

“INPUTS and OUTPUTS

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;  
 wr, rd pin;  
 psen, ale pin 38,37; “PE0-1  
 clkln, reset pin;  
 csi pin;

“base address for i/o chip selects  
 csioip node;

“inputs for frequency quadrupler and phase discriminator  
 “a and b are always 90 degrees out of phase  
 a, b, pin;

“internal signals for frequency quadrupler delayed versions  
 “of a and b  
 a\_dly, b\_dly node;

“output from phase discriminator used to gate UP and DOWN  
 “event counters: dir\_up=1 ==> UP, dir\_up=0 ==> DOWN  
 dir\_up node istype 'reg\_D';  
 dir\_int node; “partial PT's for dir\_up

“output at quadruple frequency used to generate events —  
 “gated by dir\_up  
 a\_b\_x4 node;

“internal signals from phase discriminator state counter  
 s0, s1 node istype 'reg\_D';

“macrocell events to UP (timer0) and DOWN (timer1) counters  
 mc2tmr0, mc2tmr1 node;

“ \*\*\*\*\*

“Definitions

X = .x;  
 CK = .c.;

addr = [a15,a14,a13,a12,a11,a10,a9,a8,  
 X,X,X,X,X,X,a1,a0];

“state values for state machine  
 ss = [s1, s0];

“ \*\*\*\*\*

**Abel File**  
(Cont.)**EQUATIONS**

“DPLD equations

“i/o base address for PSD internal register — defined  
 “to be a 256 address block starting at ^hC000  

$$\text{csiop} = (\text{addr} \geq \text{^h0C000}) \ \& \ (\text{addr} \leq \text{^h0C0FF});$$

“GPLD equations

“phase discriminator equations

$$\text{a\_dly} = \text{a}; \quad \text{“delay a and b by macrocell delay}$$

$$\text{b\_dly} = \text{b};$$

“generate a pulse for each edge transition of a and b — pulse  
 “width is equal to the macrocell delay of a\_dly and b\_dly

$$\text{a\_b\_x4} = (\text{a} \ \$ \ \text{a\_dly}) \ \# \ (\text{b} \ \$ \ \text{b\_dly});$$

“generate events for the UP and DOWN counters using the  
 “counter/timer macrocells

$$\text{mc2tmr0} = \text{dir\_up.fb} \ \& \ \text{a\_b\_x4.fb};$$

$$\text{mc2tmr1} = \text{!dir\_up.fb} \ \& \ \text{a\_b\_x4.fb};$$

“state machine for detecting motor direction

“dir\_up as a complete equation needs 8 PTs, but since the PSD  
 “supports a maximum of 6 PTs (Port-B macrocells), it is necessary  
 “to split this in two: dir\_int and then dir\_up

$$\text{dir\_int} = (\text{!a} \ \& \ \text{b} \ \& \ \text{s0.fb} \ \& \ \text{s1.fb}$$

$$\ \# \ \text{!a} \ \& \ \text{!b} \ \& \ \text{!s0.fb} \ \& \ \text{s1.fb}$$

$$\ \# \ \text{a} \ \& \ \text{b} \ \& \ \text{s0.fb} \ \& \ \text{!s1.fb}$$

$$\ \# \ \text{a} \ \& \ \text{!b} \ \& \ \text{!s0.fb} \ \& \ \text{!s1.fb});$$

$$\text{dir\_up} := (\text{dir\_int.fb}$$

$$\ \# \ \text{b} \ \& \ \text{s0.fb} \ \& \ \text{s1.fb} \ \& \ \text{dir\_up.FB}$$

$$\ \# \ \text{!a} \ \& \ \text{!s0.fb} \ \& \ \text{s1.fb} \ \& \ \text{dir\_up.FB}$$

$$\ \# \ \text{a} \ \& \ \text{s0.fb} \ \& \ \text{!s1.fb} \ \& \ \text{dir\_up.FB}$$

$$\ \# \ \text{!b} \ \& \ \text{!s0.fb} \ \& \ \text{!s1.fb} \ \& \ \text{dir\_up.FB});$$

$$\text{dir\_up.C} = (\text{clk});$$

$$\text{dir\_up.RE} = (\text{!reset});$$

“the state counter comprises s0 and s1

$$\text{s0.D} = (\text{a} \ \& \ \text{s0.fb}$$

$$\ \# \ \text{a} \ \& \ \text{b} \ \& \ \text{s1.fb}$$

$$\ \# \ \text{a} \ \& \ \text{!b} \ \& \ \text{!s1.fb});$$

$$\text{s0.C} = (\text{clk});$$

$$\text{s0.RE} = (\text{!reset});$$

$$\text{s1.D} = (\text{a} \ \& \ \text{b} \ \& \ \text{s0.fb}$$

$$\ \# \ \text{!a} \ \& \ \text{b} \ \& \ \text{!s0.fb}$$

$$\ \# \ \text{b} \ \& \ \text{s1.fb});$$

$$\text{s1.C} = (\text{clk});$$

$$\text{s1.RE} = (\text{!reset});$$

## Abel File (Cont.)

```

"Below is the original method used for entering the state machine
"description. The above method was cut-&-pasted from the .eq2 file
"generated after running the ABEL compiler and optimizer.

"state machine for detecting motor direction
"
" ss.clk = clkln;
" ss.re = !reset;
" dir_up.clk = clkln;
" dir_up.re = !reset;
"
"state_diagram ss
" state 0: if ((a==0) & (b==0))
" then 0 with dir_up := dir_up.fb;
" endwith;
"
" else if ((a==1) & (b==0))
" then 1 with dir_up := 1;
" endwith;
"
" else if ((a==0) & (b==1))
" then 2 with dir_up := 0;
" endwith;
"
" state 1: if ((a==1) & (b==0))
" then 1 with dir_up := dir_up.fb;
" endwith;
"
" else if ((a==1) & (b==1))
" then 3 with dir_up := 1;
" endwith;
"
" else if ((a==0) & (b==0))
" then 0 with dir_up := 0;
" endwith;
"
" state 2: if ((a==0) & (b==1))
" then 2 with dir_up := dir_up.fb;
" endwith;
"
" else if ((a==1) & (b==1))
" then 3 with dir_up := 0;
" endwith;
"
" else if ((a==0) & (b==0))
" then 0 with dir_up := 1;
" endwith;
"
" state 3: if ((a==1) & (b==1))
" then 3 with dir_up := dir_up.fb;
" endwith;
"
" else if ((a==0) & (b==1))
" then 2 with dir_up := 1;
" endwith;
"
" else if ((a==1) & (b==0))
" then 1 with dir_up := 0;
" endwith;
"
end motr_dir

```



**Stimulus File****STIMULUS file:**

```

//Stimulus file for setting up of the counter/timers in
//event count mode, and for testing the direction
//recognition state machine.

reg [7:0] dat_val; //used to hold data read from PSD

//+++++
//
// User-Defined parameters
//+++++

parameter cntr0L='hC098, cntr0H='hC099, img0L='hC090,
 img0H='hC091;
parameter cntr1L='hC09A, cntr1H='hC09B, img1L='hC092,
 img1H='hC093;
parameter cmd0='hC0A0, cmd1='hC0A1;
parameter g_cmd='hC0A8;
parameter dlcY='hC0A6;
parameter freeze='hC0A4, status='hC0A9;

//+++++
//
// Defining tasks to simplify writing the stimulus file
//+++++

task write (addr_bus,data_in);

input [15:0] addr_bus;
input [7:0] data_in;

 begin

 #20 ale = 1; //high true ale
 #20 adio = addr_bus; //set-up the right address
 #20 ale = 0; //latches address
 #20 adio = data_in; //write operation
 #40 wr = 0; //write starts
 #100 wr = 1; //write ends
 #10 adio = Z16;

 end
endtask

task read (addr_bus);

input [15:0] addr_bus;

 begin

 #20 ale = 1; //active high ale
 dat_val = 'h00; //clear dat_val register
 #20 adio = addr_bus; //set-up the right address
 #20 ale = 0; //latches address
 #20 adio = Z16; //float address bus
 #40 rd = 0; //read starts
 #50 dat_val = `adio; //store low byte of adio
 #50 rd = 1; //read ends

 end
endtask

```

**Stimulus File**  
**(Cont.)**

```

task psen (addr_bus);

input [15:0] addr_bus;

begin

#20 ale = 1; //active high ale
#20 adio = addr_bus; //set-up the right address
#20 ale = 0; //latches address
#20 adio = Z16; //float address bus
#40 psen = 0; //PSEN read starts
#100 psen = 1; //PSEN read ends

end

endtask

//***** Begin stimulus *****

initial
begin

wr = 1; rd = 1; psen = 1;
ale = 0;
clkln = 0;
reset = 0;
csi = 0;
adio = 'h0000;
a = 0; b = 0;

#500 reset = 1; //end reset cycle

//Clear counter/timers-0 and -1

#300 write(cntr0L, 'h00);
 write(cntr0H, 'h00);

 write(cntr1L, 'h00);
 write(cntr1H, 'h00);

//Clear image registers-0 and -1

write(img0L, 'h00);
write(img0H, 'h00);

write(img1L, 'h00);
write(img1H, 'h00);

//Clear delay cycle register so that clkln is scaled by 4
//(i.e. when the scale bit in the Global Command register
//is set to 0)
write(dlcy, 'h00);

```

**Stimulus File  
(Cont.)**

```

//Set command register, CMD0, to configure timer0 for event mode
//with the event coming from the macrocell, mc2tmr0

//LSB 0 Event count mode
// 1 Increment mode
// 1 Select (enable) timer0
// X No timer output in this mode
// X Pin-input polarity is not needed since the
// event is macrocell driven
// 0 Input command from macrocell
// (if set to 1 then from pin)
// 0 Load/Store command from Pin/Macrocell (in this
// case macrocell) allowed through
//MSB 0 Enable/Disable by Pin/Macrocell

 write (cmd0,'h1E);

//Set command register, CMD1, to configure timer1 for event mode
//with the event coming from the macrocell, mc2tmr1

//LSB 0 Event count mode
// 1 Increment mode
// 1 Select (enable) timer0
// X No timer output in this mode
// X Pin-input polarity is not needed since the
// event is macrocell driven
// 0 Input command from macrocell
// (if set to 1 then from pin)
// 0 Load/Store command from Pin/Macrocell (in this
// case macrocell) allowed through
//MSB 0 Enable/Disable by Pin/Macrocell

 write (cmd1,'h1E);

//Set the Global command register to enable the counter/timers
//in event/time-capture mode

//LSB 0 Scale bit (0 = divide by 1, 1 = divide by 8)
// 1 Counter start bit - enables all the selected
// counters
// 1 Global mode bit - set for event/time capture mode
// 0 Watchdog disabled
//Bit4-7 are reserved and set to 0

 write (g_cmd,'h06);

//perform a freeze/freeze acknowledge cycle on both timer0 and -1
#2000 write(freeze,'h03);

//wait for both freeze acknowledge flags to be set in
//status register (status = 'h3)
 while (dat_val <= 'h2)
 begin
 read(status);
 end
end

```

**Stimulus File**  
**(Cont.)**

```

//reset freeze bits
#1000 write(freeze, 'h00);

//perform a second freeze/freeze acknowledge cycle on both
//timer0 and -1
#10000 write(freeze, 'h03);

//wait for both freeze acknowledge flags to be set in
//status register (status = 'h3)
while (dat_val <= 'h2)
begin
read(status);
end

//reset freeze bits
#1000 write(freeze, 'h00);

end

//***** Continuous signals *****

//Generate A and B pulse streams

always
begin

//Produce "a" and "b" input pulses with "a" leading "b" by 90 degrees
repeat (6)
begin
#500 a = ~a;
#500 b = ~b;
end

//Produce pulses with "b" leading "a" by 90 degrees
repeat (3)
begin
#500 b = ~b;
#500 a = ~a;
end

end

//Generate a continuous clock signal
always
#25 clkIn = ~clkIn; //20 Mhz

```





# Programmable Peripheral Application Note 042

## Four Axis Stepper Motor Control Using a Programmable PSD5XX MCU Peripheral from WSI, Inc.

By Nasser Pooladian, Data Card Corp.

### Introduction

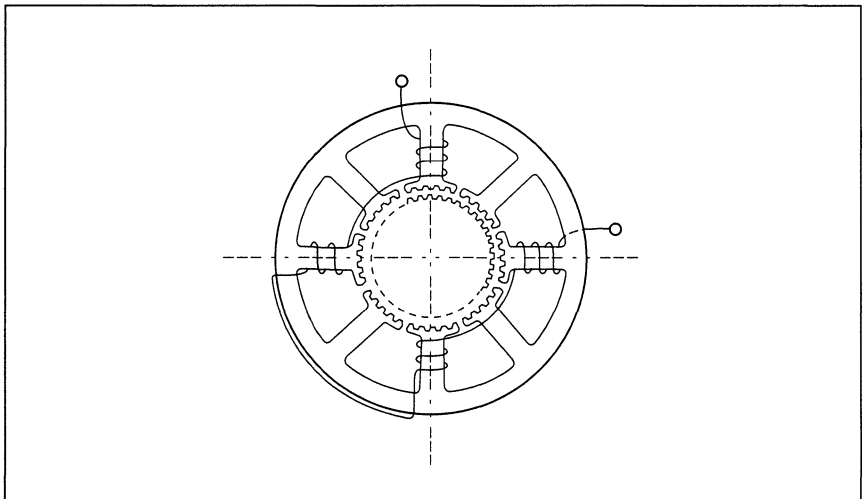
The design of a stepper motor control requires various timers and electronic controls. This application note explains the basic operation of a stepper motor. It also presents the theory, implementation and electronic control of a four axis stepper motor control using the PSD5XX family of products from the WSI Inc. The PSD5XX, as a field programmable microcontroller peripheral device, provides a high degree of integration on the embedded controller design. Configuration of the memory, ease of interface to various different microcontroller buses, interrupt handling, I/O ports, and four sixteen bit counter/timers make this device a great candidate for embedded applications.

### Stepper Motor Operation

A stepper motor is basically a rotational actuator which rotates a fixed angle when excited. A stepper motor can be directly controlled electronically without the need for a feedback element (encoder, tachometer feedback, etc.) as required in servo applications. The simpler drive and control electronics needed by a stepper motor makes it a good candidate for a positioning actuator in many different motion control applications. Several different types of stepper motors are used in the industry.

A hybrid stepping motor is used in this application. The rotor and stator are multi-toothed in a hybrid stepping motor and the rotor is magnetized in the axis of the rotor shaft. When properly driven, a hybrid stepping motor will step 1.8 degrees in the full step mode and 0.9 degrees in the half step mode. Figure 1 shows a typical hybrid motor.

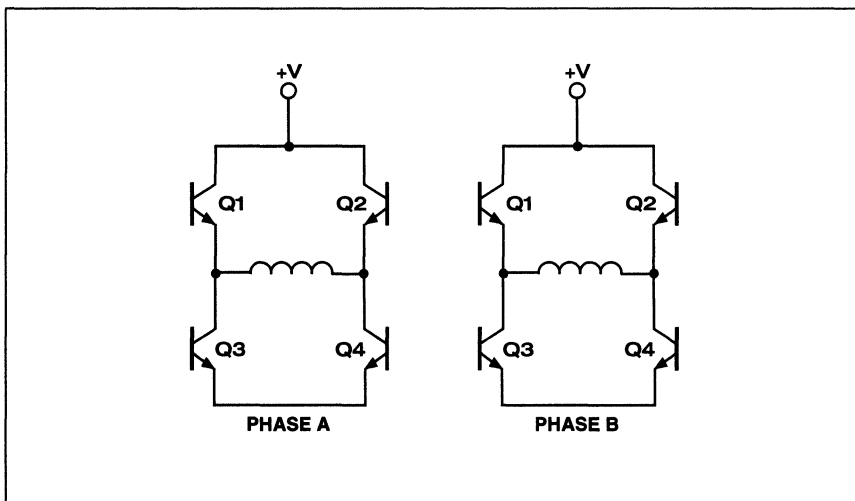
Figure 1. Hybrid Stepping Motor



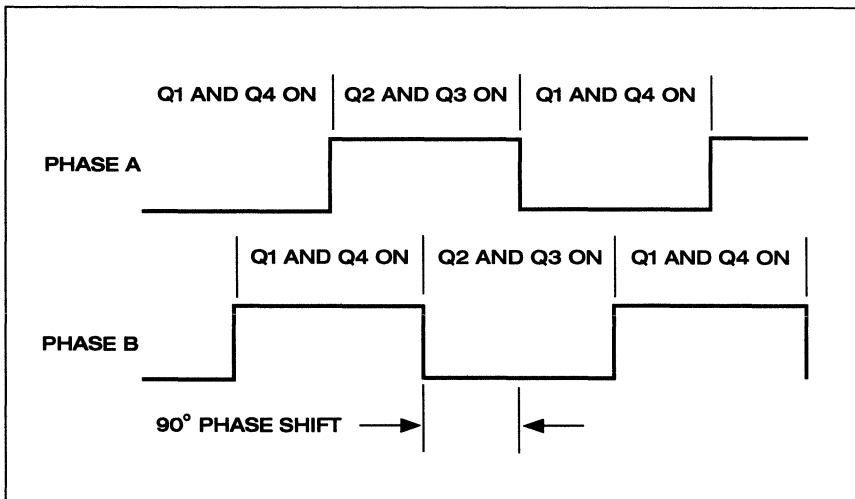
**Stepper Motor  
Operation**  
(Cont.)

The stator windings in a Hybrid stepper motor are distributed in 90-degree quadrants around the motor case. See Figure 1 for the phase winding distribution of the hybrid motor. Different methods are used for the excitation of a stepper motor. In this application a bipolar drive circuit is used for the power stage. The motor windings are connected 90 degrees apart such that the stepper motor looks like a two-phase motor. In this case there are four motor leads to be powered from the amplifier stage. Each phase of this stepper motor is powered by an H bridge. Figure 2 shows a typical H bridge that drives a stepper motor and Figure 3 illustrates the driving waveforms.

**Figure 2. Two H Bridges for Driving a Two Phase Stepper Motor.**



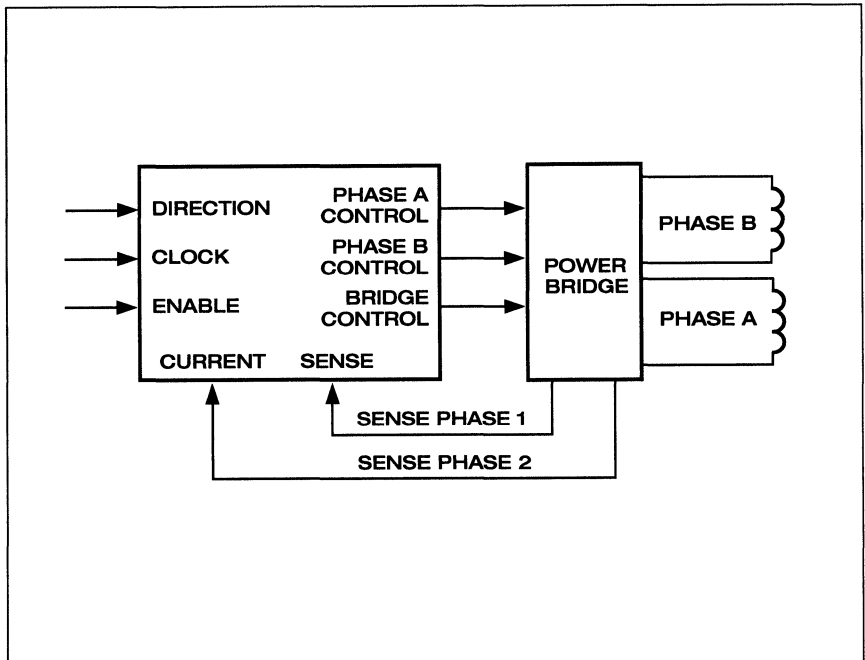
**Figure 3. Phase Excitation in a Bipolar Stepping Motor.**



**Stepper Motor  
Operation  
(Cont.)**

Phase timing for a stepper motor could be designed by either a combination of logic and linear electronics or by some stepper motor control IC's such as the L297 stepper motor controller. Figure 4 shows a block diagram of a stepper motor control and the L297 is used as the stepper motor control IC. The L297 provides control to an amplifier in the current mode. The chop frequency for the L297 is set to 20KHz. Chop frequency is used to regulate the amount of current in the motor windings. The current reference to the motor windings is set by a pair of resistors. The L297 is configured to FULL STEP mode. The ENABLE/DISABLE and axis DIRECTION control are controlled from PORT B of the PSD503B1. An electrical schematic using the L297 is given in Figure 13.

**Figure 4. Simplified Block Diagram for a Stepper Motor Control**



3

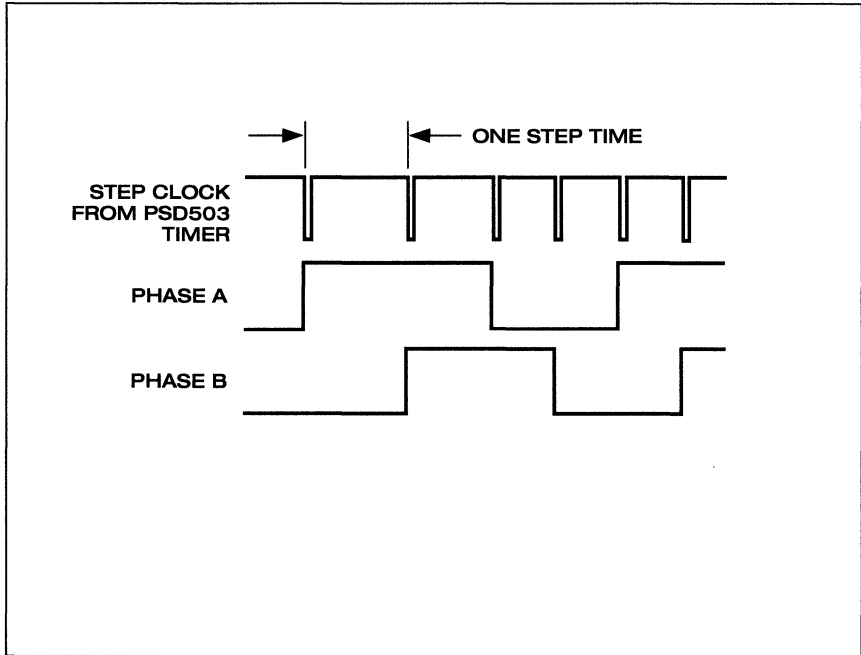


**Stepper Motor  
Clock  
Generation  
by Using a  
PSD5XX**

Figure 5 shows a timing diagram for the control of the phases in a stepper motor control where the steps and the step rate are controlled by clocks. The variation of the clock rate or the variation of the time between the two clock pulses determines the step rate. Change in the step rate determines the acceleration, deceleration, and the slew rate in a given motion profile.

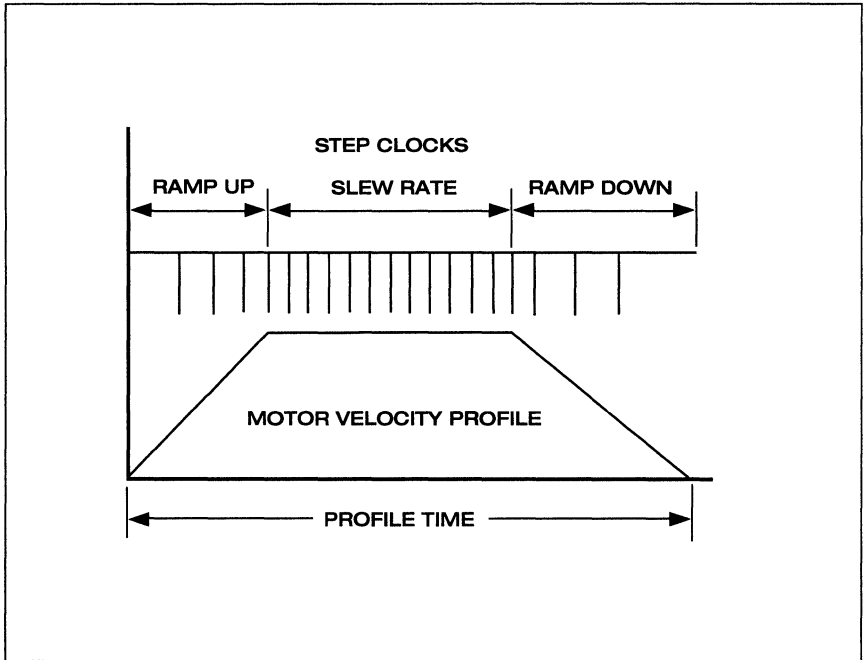
Figure 6 shows a typical trapezoidal motion profile. In the acceleration mode the step rate starts slowly and as the motion progresses the step rate increases according to a step rate table until it reaches the slew rate. At the slew rate the step rate is fixed and the period of the step clocks is constant. At the end of the slew rate the deceleration starts. In this part of the profile the step rate decreases according to a step rate table until the last step. The repeatability and accuracy of the step clocks in a stepper motor plays a major role in the stepper motor performance.

**Figure 5. Timing Diagram for a Stepper Motor Control**



**Stepper Motor  
Clock  
Generation  
by Using a  
PSD5XX  
(Cont.)**

**Figure 6. Typical Trapezoidal Speed Profile**

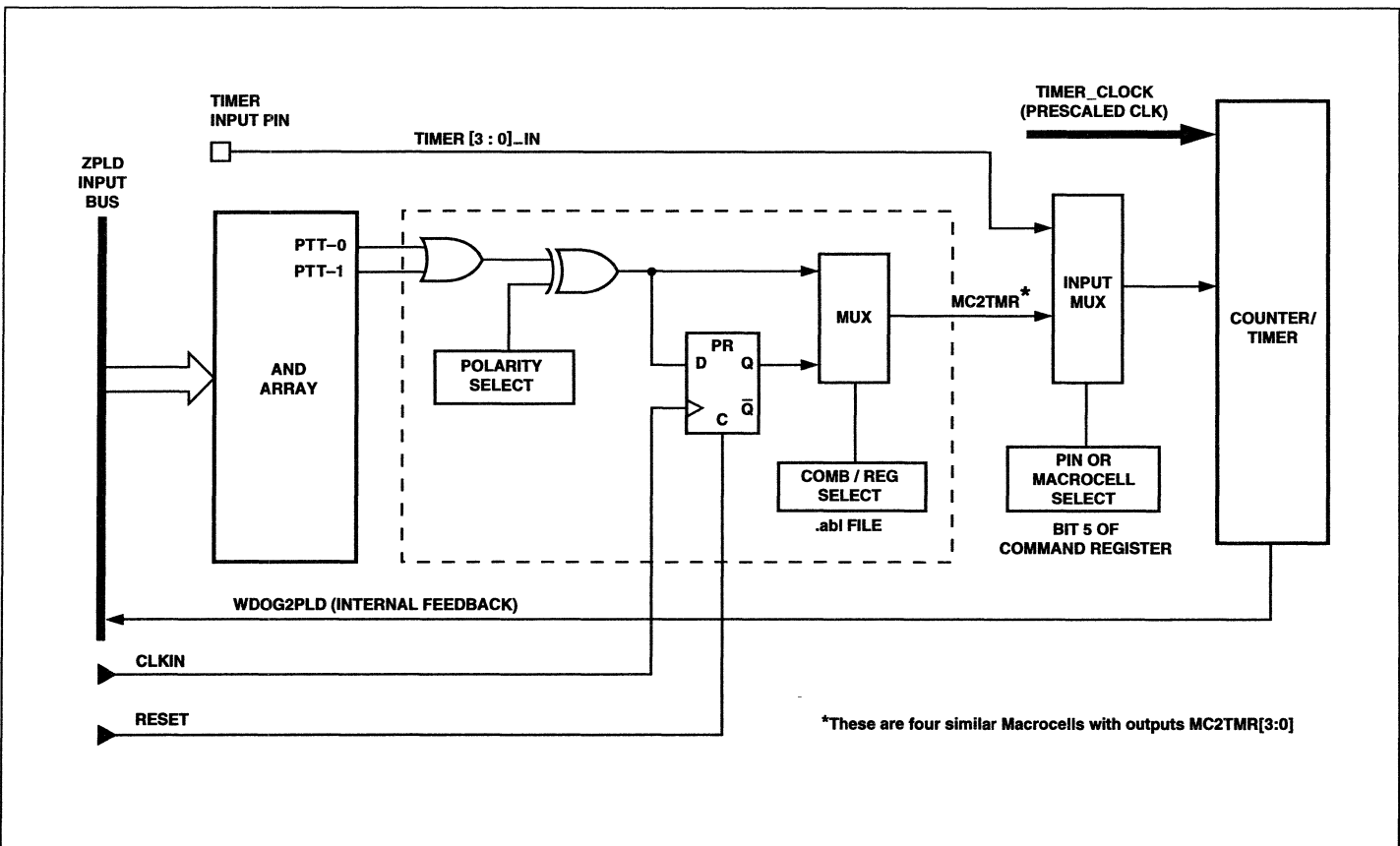


3

Figure 7 shows the programmable PLD (PPLD) macrocell for each counter/timer block diagram in the PSD5XX. In this design the four 16 bit timers on the PSD5XX are used to control a four axis stepper motor under microprocessor control. The four 16-bit timers in the PSD5XX are configured in the pulse mode. The Timers are loaded with a given step count for the duration of a pulse. When the pulse duration has expired, the logic on the PSD5XX is programmed such that the respective timer is preloaded with the count from the Image Registers. By preloading the timer, the step pulse duration will be exact with respect to the applied clock frequency. The timer clocks are configured to run at 1-MHz. In this case the preloading time on this system is based on a "one step ahead" stepper motor control. On the ramp up and ramp down mode each step clock will be preloaded in the image register because of the step rate changes. When the time for each step has expired the respective timer automatically preloads the image register in the count register and continues the new count. In this design the terminal count outputs (TC0 – TC3) of the timers are routed to the four inputs (INT0– INT3) of the interrupt controller on the PSD5XX device. The timer outputs are inverted and connected to the timer macrocell outputs MC2TMRx (x = 0 – 3 for three timers) in the PPLD logic. Figure 8 shows a simplified block diagram for the four axis stepper motor control.

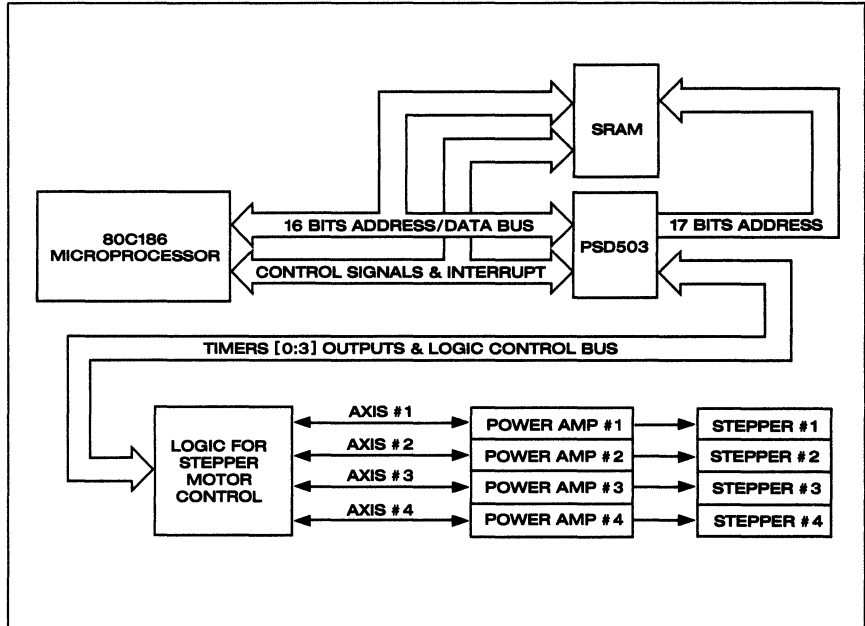
**Stepper Motor  
Clock  
Generation  
by Using a  
PSD5XX  
(Cont.)**

**Figure 7. PPLD Macrocell for Each Counter/Timer**



**Stepper Motor  
Clock  
Generation  
by Using a  
PSD5XX  
(Cont.)**

**Figure 8. Simplified Block Diagram for the System**

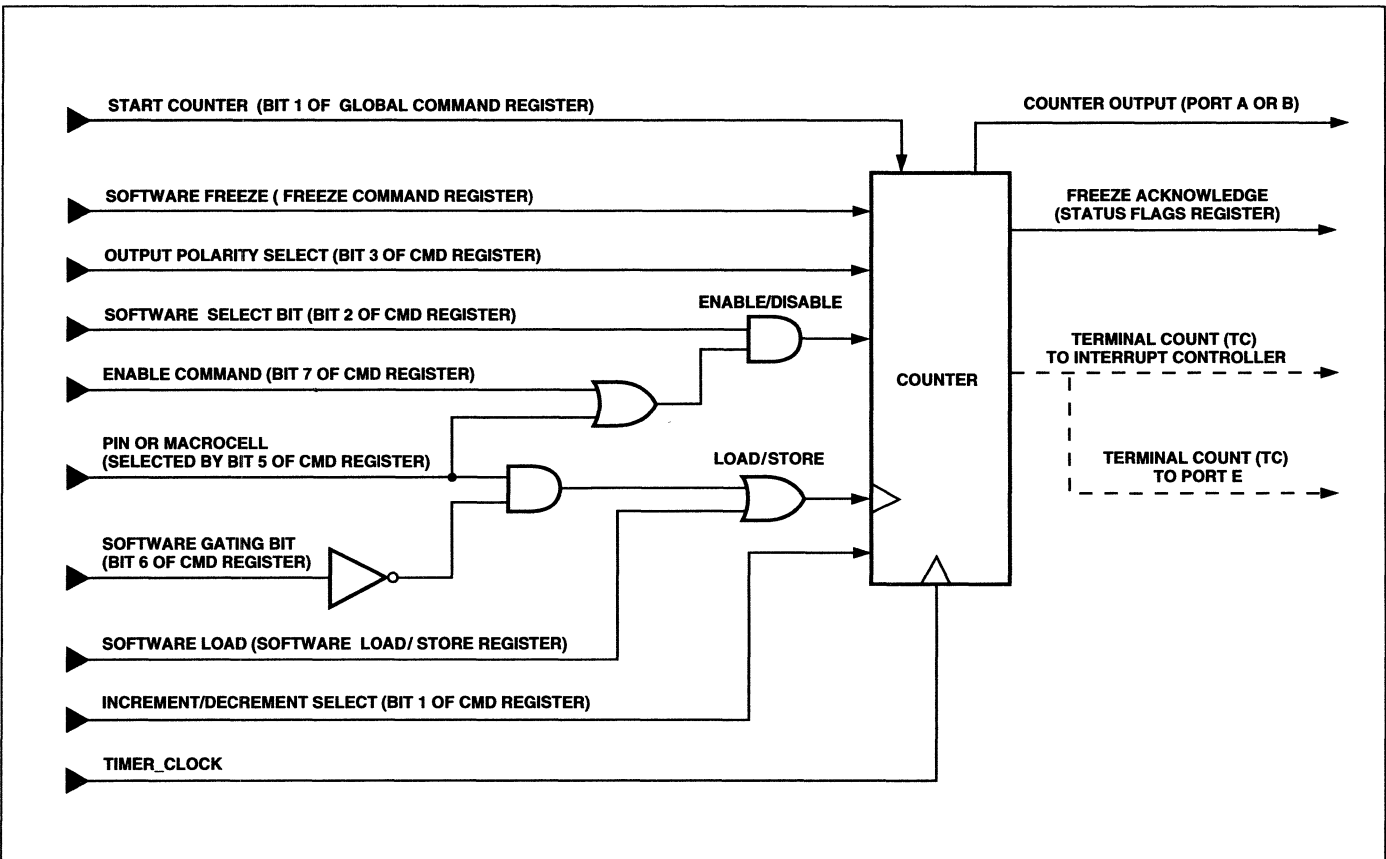


3

The output of the PSD5XX interrupt controller is connected to one of the interrupt inputs on the 80C186 microprocessor. The PSD5XX interrupt controller interrupts the microprocessor in response to the timer underflow. In response to this interrupt, the microprocessor reads the INTERRUPT PRIORITY STATUS REGISTER and updates the respective timer image register. The output of a timer makes a high to low transition when a timer count expires. The high to low transition of the timer is inverted and is used to preload the respective timer from the last image register. In the slewing mode the IMAGE REGISTER for a timer does not need to be preloaded on each step interrupt. As the timer count expires the old count will be pre-loaded automatically. Figure 9 shows the logic configuration for a given axis and Figure 10 shows the \*.abl file listing for the preloading capability of the timers.

**Stepper Motor  
Clock  
Generation  
by Using a  
PSD5XX  
(Cont.)**

**Figure 9. Logic Configuration for PSD5XX in Pulse Mode**



**Stepper Motor  
Clock  
Generation  
by Using a  
PSD5XX  
(Cont.)**

**Figure 10. A Sample PPLD Configuration in an \*.abl File for the PSD5XX**

“PPLD Equation for the Timer to Preload

```
mc2tmr0 = (!timerout0);
mc2tmr1 = (!timerout1);
mc2tmr2 = (!timerout2);
mc2tmr3 = (!timerout3);
```

**80C186  
Interface to the  
PSD503**

Figure 11 shows a block diagram of a PSD5XX family product. In this design the PSD503 is used. The PSD503 is configured to 64K x 16 EPROM in MUX mode. The address and data on the 80C186 are multiplexed so the PSD503 latches the address internally. The address lines A16 and A17 are internally latched using PA6 and PA5 from the PSD503 ports. Ports PC0 – PC7, PD0 – PD7, PE3 and PE4 on the PSD503 are used to output the address A0 – A17 externally to be used by the 128K x 16 SRAM external to the PSD503 device. PA0 through PA3 are used as timer outputs to provide clocks for the stepper motor control. Figure 12 shows the schematic for the processor connection to the PSD503 and Figure 13 shows a schematic for a typical stepper motor control unit interface to the PSD503. The stepper motor interface control uses PB0 – PB5 to control the four L297 stepper motor control chips. PB0 and PB1 are used to enable and disable the four axis of the motion. PB2 through PB5 are used to control the direction of the motor motion. PB0 through PB7 are configured in the software. Figure 14 shows the \*.ABL file used in this design.

Figure 11. PSD5XX Block Diagram

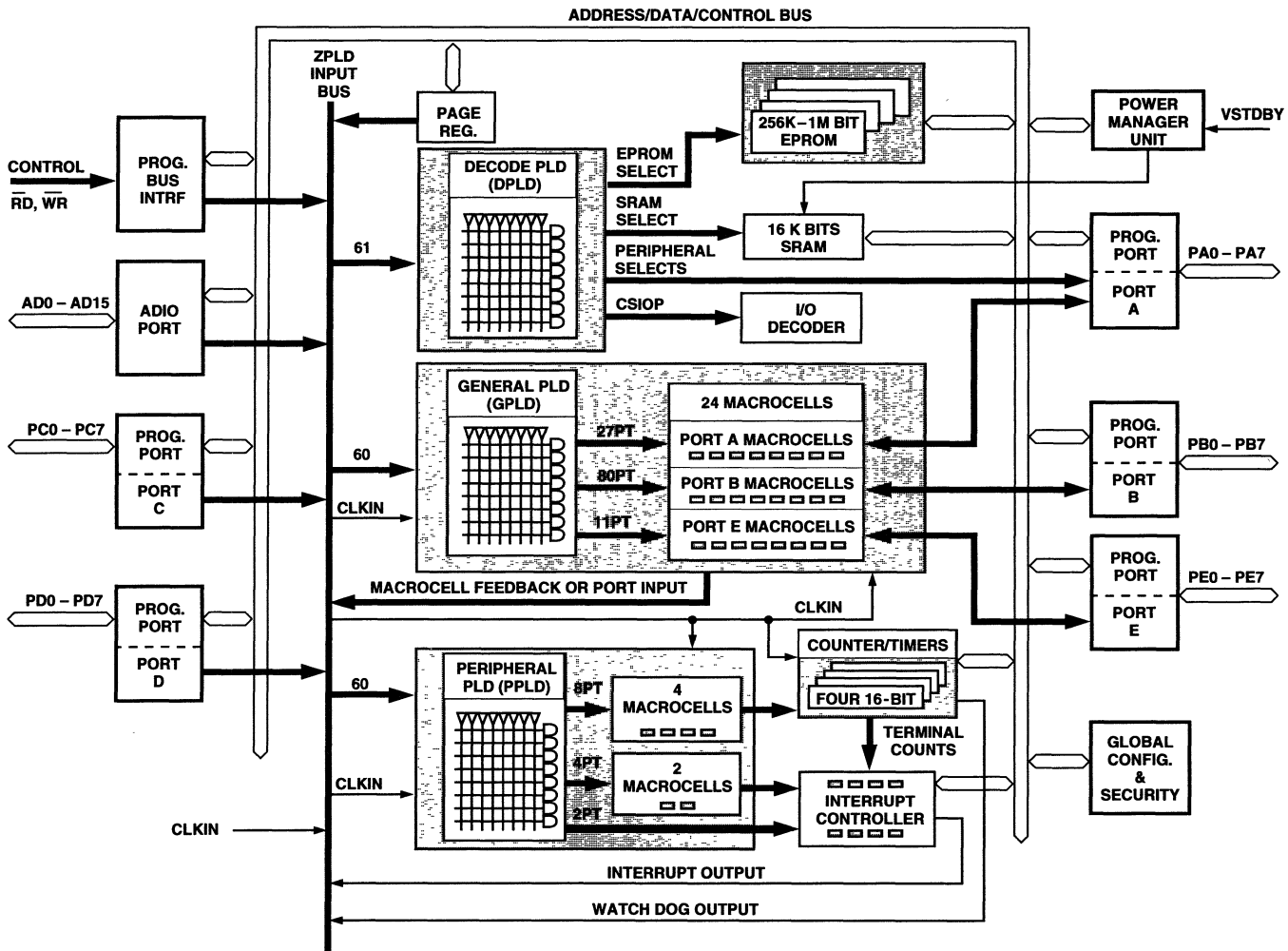


Figure 12. Schematic for the PSD503 Interface to a 80C186 Processor

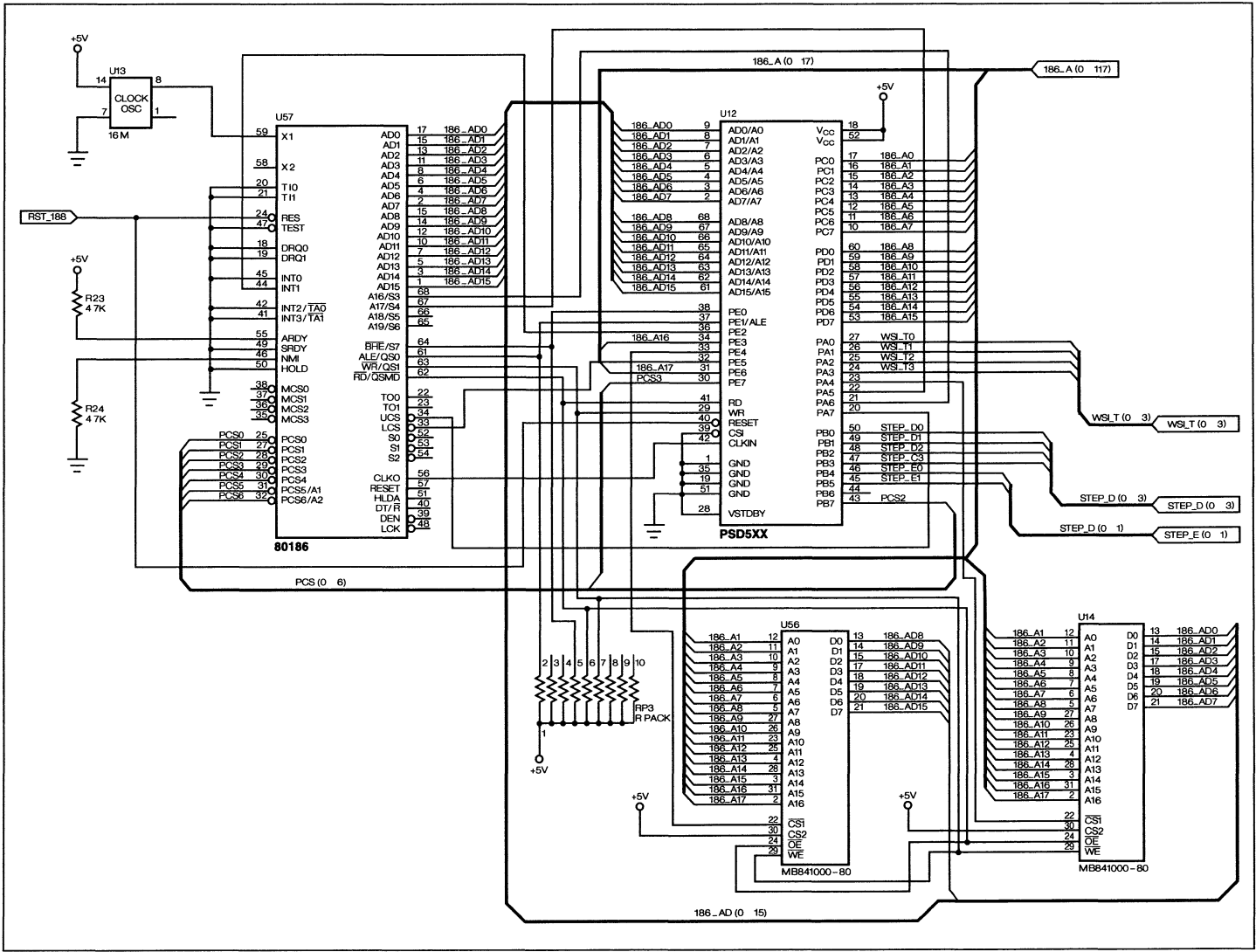
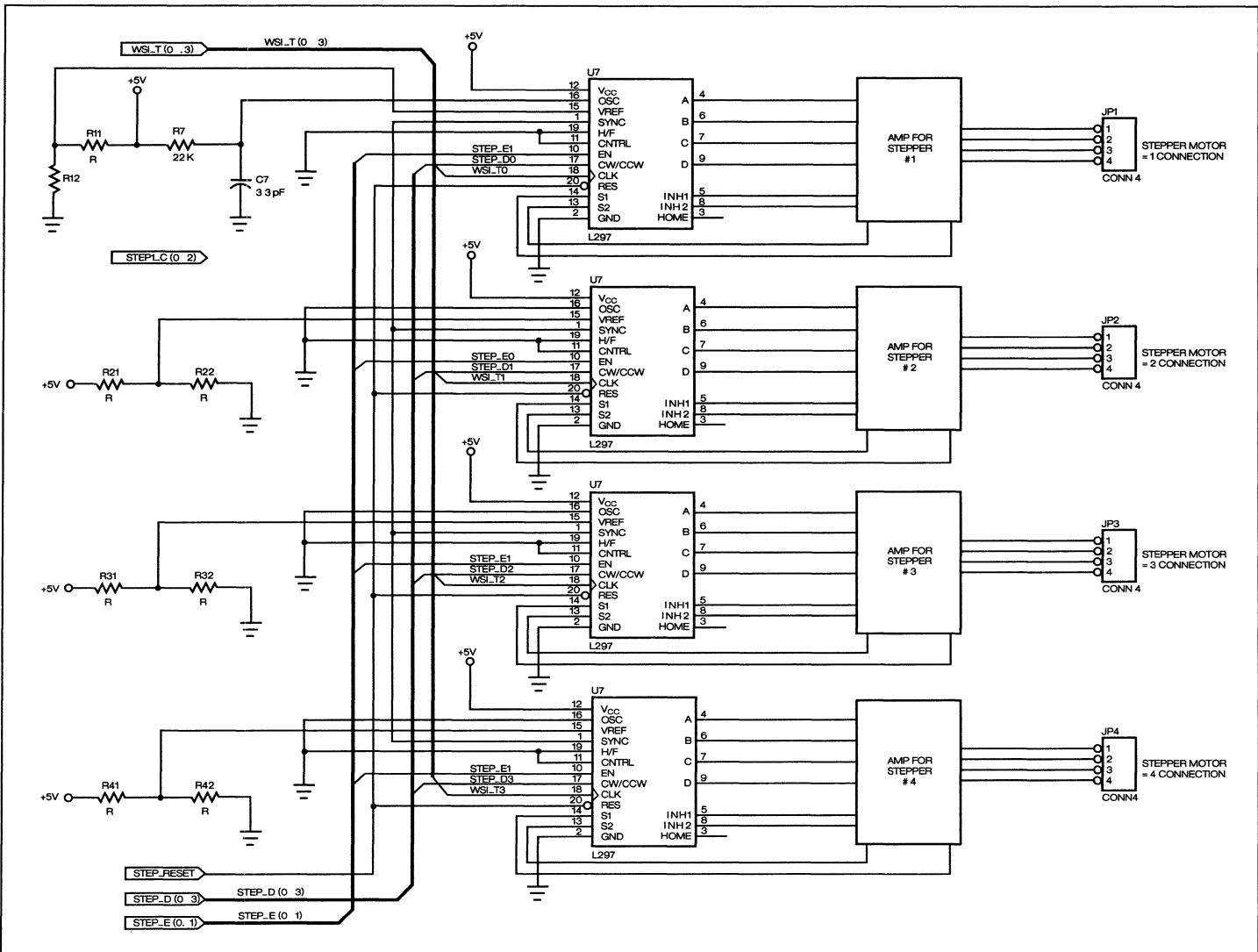




Figure 13. Schematic Interface Between PSD503 and the L297 Stepper Controller



**80C186**  
**Interface to the**  
**PSD503**  
 (Cont.)

**Figure 14. Program Listing for the ABEL File Used in this Design**

```
module mfhs_16
title 'DESIGN FOR PSD503 ABEL source file to interface with 80C186';
```

"Input signals

"Address lines, using reserved names.

```
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
wr pin;
rd pin;
bhe pin;
```

```
a16 pin 21; "high order address input
a17 pin 22; "high order address input
add_16 pin 34; "address 16 latched output
add_17 pin 31; "address 17 latched output
```

" PINS DEDFINED BY NPK "

```
umcs pin; " Upper Memory Chip Select
lmcs pin; " Lower Memory Chip Select
```

```
emcs pin; " even memory chip select for external SRAM
omcs pin; " odd memory chip select for external SRAM
```

```
pcs3 pin; " PSD Upper 256 bytes address chip select space
pcs2 pin; " PSD Lower 256 bytes address chip select space
```

```
pb0, pb1, pb2, pb3, pb4, pb5 pin; " Stepper motor Control Port
```

" Timer Contol Pins "

```
timerout0 pin; " Stepper 1 Clock 1
timerout1 pin; " Stepper 2 Clock 2
timerout2 pin; " Stepper 3 Clock 3
timerout3 pin; " Stepper 4 Clock 4
```

" Port Control "

```
pd0, pd1, pd2, pd3, pd4, pd5, pd6, pd7 pin; " Upper Address Output "
```

```
pc0, pc1, pc2, pc3, pc4, pc5, pc6, pc7 pin; " Lower Address Output "
```

```
clkin, reset pin; "using the reserved names.
```

"Output signals

```
csiop, rs0, es0, es1, es2, es3 node; "DPLD output chip selects
mc2tmr0 node; " PPLD Output To Timer 0 "
mc2tmr1 node; " PPLD Output To Timer 1 "
mc2tmr2 node; " PPLD Output To Timer 2 "
mc2tmr3 node; " PPLD Output To Timer 3 "
```

**80C186  
Interface to the  
PSD503  
(Cont.)**

**Figure 14. Program Listing for the ABEL File Used in this Design (Cont.)**

“General outputs

“DEFINITIONS

```

“page = [pgr3,pgr2,pgr1,pgr0];”
CK = .c. ; “ Clock pulse definition
X = .x. ; “ Don't care
Address = [a16,a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,a1,a0];

Add = [pc7,pc6,pc5,pc4,pc3,pc2,pc1,pc0];

```

equations

“DPLD EQUATIONS

```

csiop = ((Address >= ^h00100) & (Address <= ^h001ff));
rs0 = 0; “ Disable The 2k On Board SRAM “
es0 = (Address >= ^h00000) & (Address <= ^h07fff) & (!mcs); “ 32k block 0
es1 = (Address >= ^h08000) & (Address <= ^h0ffff) & (!mcs); “ 32k block 1
es2 = (Address >= ^h10000) & (Address <= ^h17fff) & (!mcs); “ 32k block 2
es3 = (Address >= ^h18000) & (Address <= ^h1ffff) & (!mcs); “ 32k block 3

```

```

add_16 = a16; “ Address 16 latched output “
add_17 = a17; “ Address 17 latched output “

```

```

emcs = (!mcs & bhe & !a0) + (!mcs & !bhe & !a0); “ even address SRAM chip select
omcs = (!mcs & !bhe & a0) + (!mcs & !bhe & !a0); “ odd address SRAM chip select

```

“PPLD Equations

```

mc2tmr0 = (!timerout0); “ Pre Load Timer 0 “
mc2tmr1 = (!timerout1); “ Pre Load Timer 1 “
mc2tmr2 = (!timerout2); “ Pre Load Timer 2 “
mc2tmr3 = (!timerout3); “ Pre Load Timer 3 “

```

```

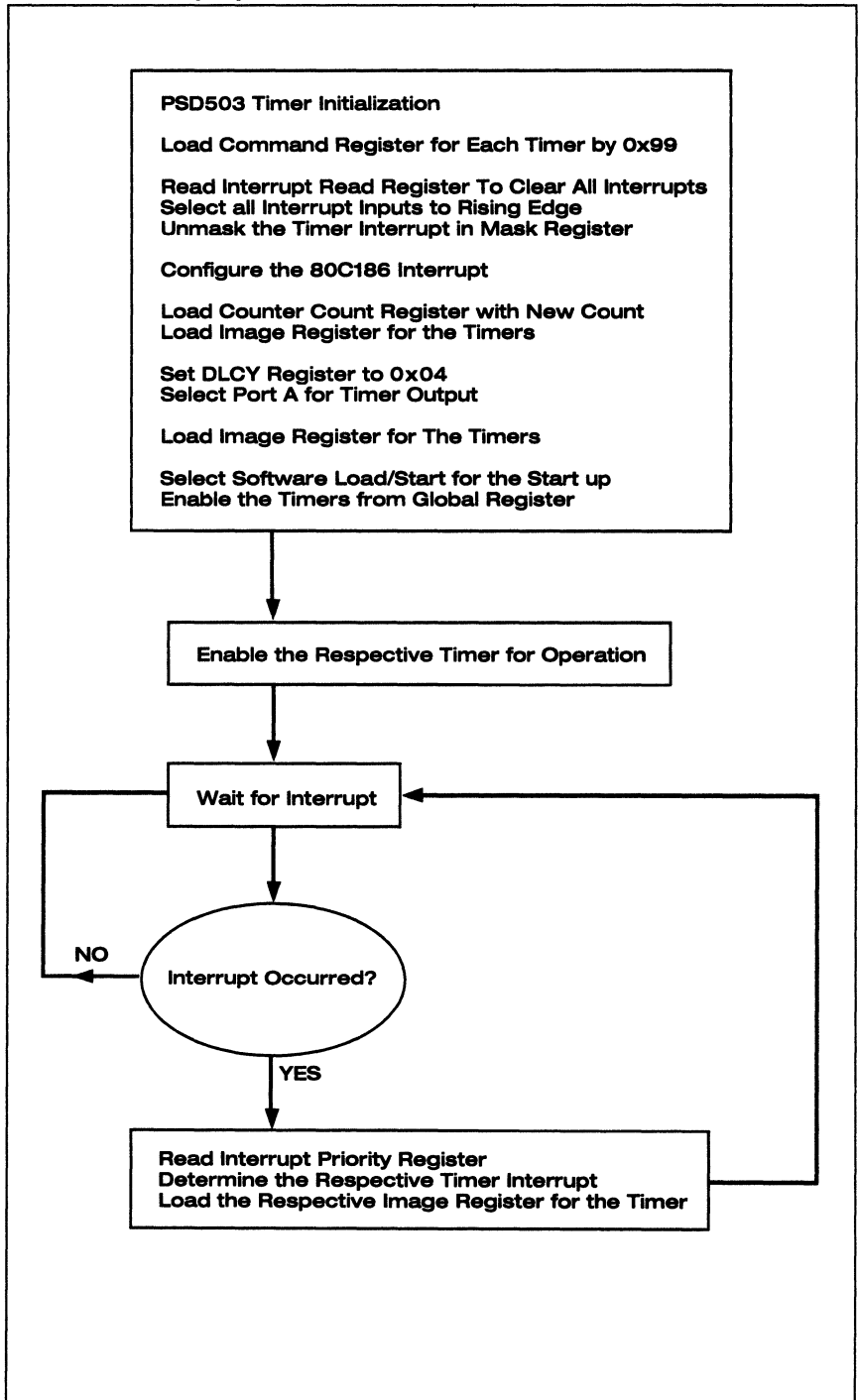
“ *****
“ TEST VECTORS
“ *****

```

END mfs\_16

**80C186  
Interface to the  
PSD503  
(Cont.)**

**Figure 15. Block Diagram for the Register Configuration and Interrupt Operation**



## 80C186 Interface to the PSD503 (Cont.)

**Figure 16. A Sample \*.C Program for this Application (Cont.)**

```

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

typedef unsigned short USHORT;
typedef short SHORT;
typedef unsigned long* PULONG;

#include "x_step.dat" /* Stepper Profile Table */

#define PCS0 0x000
#define PCS1 0x080
#define PCS2 0x100
#define PCS3 0x100
#define PCS4 0x200
#define PCS5 0x280

/* WSi Registers
*/

#define WSiINTRREAD PCS2+0xD4 /* Interrupt Read Clear */
#define WSiINTRMASK PCS2+0xD3 /* Interrupt Mask Register */
#define WSiINTRMODE PCS2+0xD2 /* Interrupt Edge/Level */
#define WSiINTRREQ PCS2+0xD1 /* Interrupt Request Latch */
#define WSiINTRPRI PCS2+0xD0 /* Interrupt Priority */

#define WSiSLR PCS2+0xa5 /* Software Load/Stor */

#define WSiCNTR0 PCS2+0x98 /* Timer 0 control */
#define WSiCNTR1 PCS2+0x9A /* Timer 1 control */
#define WSiCNTR2 PCS2+0x9C /* Timer 2 control */
#define WSiCNTR3 PCS2+0x9E /* Timer 3 control */

#define WSiCMD0 PCS2+0xa0 /* Timer 0 Control Register */
#define WSiCMD1 PCS2+0xa1 /* Timer 1 Control Register */
#define WSiCMD2 PCS2+0xA2
#define WSiCMD3 PCS2+0xA3
#define WSiDLCY PCS2+0xa6 /* Scale Factor Control Of Timers */
#define WSiIMG0 PCS2+0x90 /* Timer 0 Image Register*/
#define WSiIMG1 PCS2+0x92 /* Timer 1 Image Register*/
#define WSiIMG2 PCS2+0x94 /* Timer 0 Image Register*/
#define WSiIMG3 PCS2+0x96 /* Timer 1 Image Register*/

#define WSiGLBREG PCS2+0xa8 /* Timers Global Register */
#define WSiSFR PCS2+0x08 /* Special function register for port A */
#define WSiFREEZ PCS2+0xA4

#define WSiPORTB_CNTR PCS2+0x03 /* Port B Configuration */
#define WSiPORTB_DIR PCS2+0x07

#define WSiPORTC_CNTR PCS2+0x12 /* Port C Configuration */
#define WSiPORTC_DIR PCS2+0x16

#define WSiPORTD_CFG PCS2+0x13 /* Port D Configuration */
#define WSiPORTD_DIR PCS2+0x17

#define WSiPORTE_SFR PCS2+0x28 /* Port E Configuration */
#define WSiPORTE_DIR PCS2+0x26

```

**80C186**  
**Interface to the**  
**PSD503**  
**(Cont.)**

**Figure 16. A Sample \*.C Program for this Application (Cont.)**

```
#define WSiPORTE_OUT PCS2+0x24
#define WSiPORTE_IN PCS2+0x20
#define WSiPORTE_CFG PCS2+0x22

#define PULSE_MODE_DISABLED 0x99 /* was 99 */
#define ENABLE 0x04 /* was 00*/
/* 188 Registers
*/
#define I0CON 0xFF38
#define I1CON 0xFF3A
#define IMASK 0xFF28
#define EOI 0xFF22

/*
 Global Variables Here
*/

int Stepper_1_Total_Step_Count;
int Step_1_Max_Slew_Count;
int Step_1_Motion_Index;
int Step_1_Motion_Stat;
int Step_1_Slew_Count;
int Stepper_1_Ramp_Up_Count;
int Stepper_1_Ramp_Down_Count;
int Stepper_1_Step_Count;
int Stepper_1_Step_Time;
int Step_1_Time;
int Step_1_Count_Old;
int Stepper_1_Profile[20];

}

USHORT Image;
USHORT MotorNum;
/*
 INTERRUPT 1 ROUTINE
*/

USHORT Port = WSiINTRREAD;
USHORT iw;
void _interrupt WSiHandler(void)
{
 static USHORT Read;
 Image = 0x200;
 switch(inportb(WSiINTRPRI) & 0x07)
 {
```

**80C186**  
**Interface to the**  
**PSD503**  
**(Cont.)**

**Figure 16. A Sample \*.C Program for this Application (Cont.)**

```

 case 0:
 output(WSiIMG0, Step_1_Time); /* Load Timer 0 for step pulse */
 Stepper_1_Step_Count++;
 break;
 case 1:
 output(WSiIMG1, Image);
 break;
 case 2:
 output(WSiIMG2, Image);
 break;
 case 3:
 output(WSiIMG3, Image);
 break;
 }
 output(EOI,0x8000);
 _enable();
}

void Init_Timers(void)
{
 PULONG pIVT=NULL;
 _disable();
 Image = 100;

 /* Pulse Mode Timer 0-3
 */
 outputb(WSiCMD0, 0x99); /* Program Command Register For The Counters */
 outputb(WSiCMD1, 0x99); /* All Counters to Pulse Mode */
 outputb(WSiCMD2, 0x99); /* All Counters Disabled */
 outputb(WSiCMD3, 0x99);

 /* Interrupt WSi Setup
 */
 inportb(WSiINTRREAD); /* Clear All The Interrupts */
 outputb(WSiINTRMODE,0x00);
 outputb(WSiINTRMASK,0x0f); /* Unmask Timers Interrupt */

 outputb(WSiPORTE_SFR,0x0d); /* Configure Port E For Special Function */

 /* Interrupt 188 Setup
 */
 pIVT[12] = (PULONG)SensorInt0; /* Sensor Interrupt */
 outputb(I0CON,0x012); /* Disable sensor interrupt */
 pIVT[13] = (PULONG)WSiHandler; /* Interrupt # 1 Initialization */
 outputb(I1CON,0x010); /* Was level sensitive 0x07 */
 outputb(IMASK,0xDD);

 Image = 0;

```

**80C186**  
**Interface to the**  
**PSD503**  
**(Cont.)**

**Figure 16. A Sample \*.C Program for this Application (Cont.)**

```

outport(WSiCNTR0, 0x00);
outport(WSiCNTR1, 0x00);
outport(WSiCNTR2, 0x00);
outport(WSiCNTR3, 0x00);

outportb(WSiDLCY, 0x04);
outportb(WSiSFR, 0x0f);

outport(WSiIMG0, Image+0x340);
outport(WSiIMG1, Image+0x300);
outport(WSiIMG2, Image+0x260);
outport(WSiIMG3, Image+0x220);

outportb(WSiSLR,0x0F);
outportb(WSiGLBREG,0x02); /* Configure The Wsi Global Register */

_enable(); /* Enable Interrupt */

}
/*
This Routine Sets up timer 0 for the start up profile
*/
void Step_1_Init(void)
{
static SHORT s_en1,s1_c;
static SHORT c_r1,c_r2,m_c;

Stepper_1_Step_Count = 0;
Step_1_Count_Old = 0;

Stepper_1_Step_Count = 0;
Stepper_1_Total_Step_Count =1000;
Step_1_Max_Slew_Count = 998;
Step_1_Motion_Index = 0;
Step_1_Slew_Count = 0;
Stepper_1_Ramp_Up_Count = 5;
Stepper_1_Ramp_Down_Count = 7;
Step_1_Motion_Stat = 0; /* Set Up For Ramp Up */
Step_1_Time = 0x3000;
outportb(Step_Motor1_Control,s_en1); /* Reset Motor State */

outportb(WSiCMD0, 0x9d); /* Enable Timer 0 for stepper 1 */

}
/*
This routin is used to update the profile table for motor 1
*/
void Stepper_1_Move(void)
{
if(Stepper_1_Step_Count > Step_1_Count_Old)
{

```



**80C186**  
**Interface to the**  
**PSD503**  
 (Cont.)

**Figure 16. A Sample \*.C Program for this Application (Cont.)**

```

if(Step_1_Motion_Stat == 0)
{
 Step_1_Motion_Index++; /* ***** RAMP UP STEPPER 1 ***** */
 Step_1_Time = x_axis[Step_1_Motion_Index];
 if(Step_1_Motion_Index == 132)
 {
 Step_1_Motion_Stat = 1; /* Set Status For Slew */
 }
}

if(Step_1_Motion_Stat == 1)
{
 Step_1_Slew_Count++; /* ***** SLEW FOR STEPPER 1 ***** */
 if(Step_1_Slew_Count == Step_1_Max_Slew_Count)
 {
 Step_1_Motion_Stat = 2; /* Set Status For Ramp Down */
 Step_1_Motion_Index = 132;
 }
}

if(Step_1_Motion_Stat == 2)
{
 Step_1_Motion_Index--; /* ***** RAMP DOWN FOR STEPPER 1 ***** */
 if(Step_1_Motion_Index != 0)
 {
 Step_1_Time = x_axis[Step_1_Motion_Index];
 }
 if(Step_1_Motion_Index == 0)
 {
 Stepper_1_Step_Count = 0;
 Step_1_Count_Old = 0;
 outportb(WSiCMD0, 0x99); /* Disable Motor */
 outportb(WSiCMD0, 0x99); /* Disable Motor. This is Just For Ice */
 }
}

Step_1_Count_Old = Stepper_1_Step_Count;
}
}
}

```

**80C186**  
**Interface to the**  
**PSD503**  
**(Cont.)****Figure 16. A Sample \*.C Program for this Application (Cont.)**

```
main()
{
 static USHORT Read, y, d1=0xAA,d2=0xAA;
 static USHORT key;

 Init_Timers();

 Step_1_Init();

 key = 1;
 while(1)
 {
 switch(key)
 {
 case 1:
 Stepper_1_Move();
 break;
 case 2:
 stp_2();
 break;
 case 3:
 dcm_1();
 break;
 case 4:
 dcm_2();
 break;
 case 5:
 cres_12();
 break;
 case 6:
 C188_152();
 break;
 }
 }
 return 0;
}
```

**Software Configuration of the PSD503**

Figure 15 shows a block diagram of the steps needed to configure the registers of the PSD503 for this application. Figure 16 shows a sample software program written in C that is used in this application to configure the PSD503. This software programs the special function register of Port A to be used as the timer outputs. Figure 17 shows the PSDSOFT configuration of the timers. The PSD503 must be configured through PSDSOFT for the BUS type, WR, RD, INTR and PORT operation.

The timer clock frequency is configured through the DLCY register to 1MHz. As the step rate increases the step rate accuracy deteriorates due to the quantization effect. The quantization effect is not a problem in this application. The output pulse width of each timer is one microsecond which is sufficient for this application.

**Figure 17. PSDsoft Configuration of the Timers**

|                                                |                      |
|------------------------------------------------|----------------------|
| Counter / Timer 0:                             | Waveform/Pulse Mode. |
| Counter / Timer 1:                             | Pulse Output.        |
| Counter / Timer 2:                             | Waveform/PulseMode.  |
| Counter / Timer 3:                             | Pulse Output         |
| Do you need Automatic Power Down Clock Input ? | NO                   |
| Do you want to set the security bit ?          | NO                   |
| Do you need the Intr output signal ?           | YES                  |

**Conclusion**

In this application the PSD503 provided a very useful integrated means of design. The following were benefited from this design:

- 64 K x 16 EPROM
- Eighteen bits of latched output for demultiplexing ADDRESS from DATA.
- An 8-bit Interrupt Controller Equivalent to an 8259.
- Four 16-bit preloadable timers with a prescaler for the timer clocks.
- Logic for decoding.
- Programmable external PORTS.

The board space reduction and the amount of noise reduction that resulted from this design is immeasurable.

**Design for  
PSD503 ABEL  
Source File to  
Interface with  
80C186**

\*\*\*\*\*  
**W S I - PSDsoft Version 1.05B**  
**Output of PSD Fitter**  
\*\*\*\*\*

TITLE : DESIGN FOR PSD503 ABEL source file to interface with 80C186  
PROJECT : mfhs\_16 DATE : 04/07/1995  
DEVICE : PSD503B1 TIME : 09:31:05  
FIT OPTION : Keep Current  
\*\*\*\*\*

**Pin Assignment**

|                              |           |            |                                |
|------------------------------|-----------|------------|--------------------------------|
|                              | 1 ] GND   | GND [35    |                                |
| Address/Data Bus ADIO_7      | 2 ] adio7 | pe2 [36    | introut                        |
| Address/Data Bus ADIO_6      | 3 ] adio6 | pe1 [37    | ale                            |
| Address/Data Bus ADIO_5      | 4 ] adio5 | pe0 [38    | bhe                            |
| Address/Data Bus ADIO_4      | 5 ] adio4 | csi [39    | csi                            |
| Address/Data Bus ADIO_3      | 6 ] adio3 | reset [40  | reset                          |
| Address/Data Bus ADIO_2      | 7 ] adio2 | rd [41     | rd                             |
| Address/Data Bus ADIO_1 (a1) | 8 ] adio1 | clkln [42  | clkln                          |
| Address/Data Bus ADIO_0 (a0) | 9 ] adio0 | pb7 [43    | pcs2                           |
| pc7                          | 10] pc7   | pb6 [44    | (Not Used)                     |
| pc6                          | 11] pc6   | pb5 [45    | pb5                            |
| pc5                          | 12] pc5   | pb4 [46    | pb4                            |
| pc4                          | 13] pc4   | pb3 [47    | pb3                            |
| pc3                          | 14] pc3   | pb2 [48    | pb2                            |
| pc2                          | 15] pc2   | pb1 [49    | pb1                            |
| pc1                          | 16] pc1   | pb0 [50    | pb0                            |
| pc0                          | 17] pc0   | GND [51    |                                |
|                              | 18] VCC   | VCC [52    |                                |
|                              | 19] GND   | pd7 [53    | pd7                            |
| umcs                         | 20] pa7   | pd6 [54    | pd6                            |
| a16                          | 21] pa6   | pd5 [55    | pd5                            |
| a17                          | 22] pa5   | pd4 [56    | pd4                            |
| omcs                         | 23] pa4   | pd3 [57    | pd3                            |
| timerout3                    | 24] pa3   | pd2 [58    | pd2                            |
| timerout2                    | 25] pa2   | pd1 [59    | pd1                            |
| timerout1                    | 26] pa1   | pd0 [60    | pd0                            |
| timerout0                    | 27] pa0   | adio15 [61 | Address/Data Bus ADIO_15 (a15) |
|                              | 28] VSTBY | adio14 [62 | Address/Data Bus ADIO_14 (a14) |
| wr                           | 29] wr    | adio13 [63 | Address/Data Bus ADIO_13 (a13) |
| pcs3                         | 30] pe7   | adio12 [64 | Address/Data Bus ADIO_12 (a12) |
| add_17                       | 31] pe6   | adio11 [65 | Address/Data Bus ADIO_11 (a11) |
| lmcs                         | 32] pe5   | adio10 [66 | Address/Data Bus ADIO_10 (a10) |
| emcs                         | 33] pe4   | adio9 [67  | Address/Data Bus ADIO_9 (a9)   |
| add_16                       | 34] pe3   | adio8 [68  | Address/Data Bus ADIO_8 (a8)   |

**Global Configuration**

Data Bus : 16 Multiplexed  
ALE/AS Signal : Active High  
WatchDog Mode : Off  
Security Protection : Off

**Address & Data Bus Assignment**

**Stimulus Bus Name Signal Description**

`adiol = adio[7:0] = Address/Data Bus ADIO\_7 – ADIO\_0  
`adioh = adio[15:8] = Address/Data Bus ADIO\_15 – ADIO\_8  
adio = adio[15:0] = Address/Data Bus ADIO\_15 – ADIO\_0



**Design for  
PSD503 ABEL  
Source File to  
Interface with  
80C186  
(Cont.)**

**Resource Usage Summary**

| Device Resources                           | Used/Total | Percentage |
|--------------------------------------------|------------|------------|
| Port A: (pin 20 – pin 27)                  |            |            |
| I/O pins                                   | 8 / 8      | 100 %      |
| MCU I/O or Address Out                     | 0 / 8      | 0 %        |
| Peripheral I/O                             | 0 / 8      | 0 %        |
| ZPLD Inputs                                | 3 / 8      | 37 %       |
| ZPLD Combinatorial Outputs                 | 1 / 8      | 12 %       |
| ZPLD Registered Outputs                    | 0 / 8      | 0 %        |
| Other Information                          |            |            |
| Buried Macrocells                          | 0 / 7      | 0 %        |
| Product Terms                              | 1 / 27     | 3 %        |
| Timer Outputs                              | 4 / 4      | 100 %      |
| Port B: (pin 43 - pin 50)                  |            |            |
| I/O pins                                   | 7 / 8      | 87 %       |
| MCU I/O or Address Out                     | 7 / 8      | 87 %       |
| ZPLD Inputs                                | 0 / 8      | 0 %        |
| ZPLD Combinatorial Outputs                 | 0 / 8      | 0 %        |
| ZPLD Registered Outputs                    | 0 / 8      | 0 %        |
| Other Information                          |            |            |
| Buried Macrocells                          | 0 / 8      | 0 %        |
| Product Terms                              | 0 / 80     | 0 %        |
| Timer Outputs                              | 0 / 4      | 0 %        |
| Port C: (pin 10 - pin 17)                  |            |            |
| I/O Pins                                   | 8 / 8      | 100 %      |
| MCU I/O or Address Out                     | 8 / 8      | 100 %      |
| ZPLD Input Pins                            | 0 / 8      | 0 %        |
| Data Port (Non-Mux Bus)                    | 0 / 8      | 0 %        |
| Port D: (pin 53 - pin 60)                  |            |            |
| I/O Pins                                   | 8 / 8      | 100 %      |
| MCU I/O or Address Out                     | 8 / 8      | 100 %      |
| ZPLD Input Pins                            | 0 / 8      | 0 %        |
| Data Port (16-Bit Non-Mux Bus)             | 0 / 8      | 0 %        |
| Port E: (pin 30 - pin 34, pin 36 - pin 38) |            |            |
| I/O pins                                   | 8 / 8      | 100 %      |
| MCU I/O or Address Out                     | 1 / 8      | 12 %       |
| ZPLD Inputs                                | 1 / 8      | 12 %       |
| ZPLD Combinatorial Outputs                 | 3 / 8      | 37 %       |
| ZPLD Registered Outputs                    | 0 / 8      | 0 %        |
| Control Signal Inputs                      | 2 / 2      | 100 %      |
| Timer Control Inputs                       | 0 / 4      | 0 %        |
| Interrupt Control Output                   | 1 / 1      | 100 %      |
| APD Clock Input                            | 0 / 1      | 0 %        |
| Terminal Counts (TC)                       | 0 / 4      | 0 %        |
| Other Information                          |            |            |
| Buried Macrocells                          | 0 / 5      | 0 %        |
| Product Terms                              | 3 / 11     | 27 %       |
| Counter/Timer: Embedded Nodes              |            |            |
| Product Terms                              | 4 / 8      | 50 %       |
| Interrupt: Embedded Nodes                  |            |            |
| Product Terms                              | 0 / 4      | 0 %        |

**Design for  
PSD503 ABEL  
Source File to  
Interface with  
80C186**

*(Cont.)*

**OMC Resource Assignment**

| Resources Used  | User Name |                           |
|-----------------|-----------|---------------------------|
| <b>Port A :</b> |           |                           |
| macro cell 4    | omcs      | (mc_pa4) => Combinatorial |
| <b>Port B:</b>  |           |                           |
| <b>Port E:</b>  |           |                           |
| macro cell 3    | add_16    | (mc_pe3) => Combinatorial |
| macro cell 4    | emcs      | (mc_pe4) => Combinatorial |
| macro cell 6    | add_17    | (mc_pe6) => Combinatorial |

**EQUATIONS**

**DPLD EQUATIONS:**

```

es0 = !a15 & !a16 & !umcs;
es1 = a15 & !a16 & !umcs;
es2 = !a15 & a16 & !umcs;
es3 = a15 & a16 & !umcs;
rs0 = 0;
csiop = !a15 & !a14 & !a13 & !a12 & !a11 & !a10 & !a9 & a8 & !a16;

```

**TIMER EQUATIONS:**

```

mc2tmr0 = !timerout0;
mc2tmr1 = !timerout1;
mc2tmr2 = !timerout2;
mc2tmr3 = !timerout3;

```

**INTERRUPT EQUATIONS:**

**PORT A EQUATIONS:**

```

omcs = !bhe & !lmcs;
[omcs].OE = 1;

```

**PORT B EQUATIONS:**

**PORT E EQUATIONS:**

```

add_16 = a16;
emcs = !a0 & !lmcs;
add_17 = a17;
[add_16, emcs, add_17].OE = 1;

```





---

***PSD3XX Family***

**1**

---

***ZPSD3XX Family***

**2**

---

***PSD4XX/5XX Family***

**3**

---

***Motorola Application Notes***

**4**

---

***Sales Representatives  
and Distributors***

**5**



# Section Index

---

## **Motorola Application Notes**

*The following are Motorola Application Notes and known as Application Notes 043 and 044 at WSI, Inc.*

- Application Note 043 Using M68HC11 Microcontrollers  
with WSI Programmable Peripheral Devices .....4-1
- Application Note 044 High Performance M68HC11 System Design  
Using The WSI PSD4XX and PSD5XX Families .....4-9

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

---

**MOTOROLA**  
**SEMICONDUCTOR**  
**APPLICATION NOTE**

# Using M68HC11 Microcontrollers with WSI Programmable Peripheral Devices

by Steve Torp - Motorola Semiconductor  
Karen Spesard - WSI

## INTRODUCTION

Following system development using M68HC711 microcontroller (MCU) devices with EPROM or one time programmable ROM (OTPROM), a final design is often implemented using an equivalent mask-programmed M68HC11 device. However, there is a quick, cost-effective alternative to this method of going to production.

WSI manufactures a complete line of PSD programmable MCU peripherals that make it possible to use a ROM-less M68HC11 derivative instead of a mask-programmed device. PSD devices combine EPROM, SRAM, programmable logic for memory map decoding, programmable I/O ports, an address latch, power management, and other capabilities on a single chip. A "twin chip" solution can increase flexibility, provide expanded memory and enhanced I/O, lower power consumption, and lower cost — all with a minimum of software and hardware modifications.

This application note describes the process of converting from a prototype design that uses an M68HC711 device to a production design that uses a low-cost M68HC11 derivative and a WSI PSD.

4

## CONVERSION PROCEDURES

There are eight steps in the conversion process. Each is discussed in detail in the following text.

1. Choose the M68HC11 and PSD
2. Add the PSD to the design
3. Configure the M68HC11 for expanded mode operation
4. Configure the PSD
5. Make memory map and I/O port selections
6. Modify M68HC11 code to address memory and I/O
7. Integrate M68HC11 code with PSD configuration data
8. Program the PSD

### CHOOSE THE M68HC11 AND PSD

The M68HC11 family offers a wide range of operating voltage and frequency selections. **Table 1** shows M68HC11 Family devices, including M68L11 low-power devices, that can be used in two-chip systems. EPROM/OTPROM devices are shown in bold. WSI PSDs are available in a variety of configurations. PSDs provide a larger memory size, I/O port expansion, and programmable logic to an M68HC11 system. Low-power PSDs are a perfect complement to M68L11 MCUs.



**Table 1 Motorola M68HC11 Devices**

| Motorola Part Number     | ROM        | RAM        | EEPROM     | I/O       | A/D        |            |
|--------------------------|------------|------------|------------|-----------|------------|------------|
| MC68HC11A0<br>MC68L11A0  | 0          | 256        | 0          | 22        | Yes        |            |
| MC68HC11A1<br>MC68L11A1  | 0          | 256        | 512        | 22        |            |            |
| MC68HC11A7<br>MC68L11A7  | 8K         | 256        | 0          | 38        |            |            |
| MC68HC11A8<br>MC68L11A8  | 8K         | 256        | 512        | 38        |            |            |
| MC68HC11C0               | 0          | 256        | 0          | 35        |            | Yes        |
| <b>MC68HC711D3</b>       | <b>4K</b>  | <b>192</b> | <b>0</b>   | <b>32</b> |            | <b>No</b>  |
| MC68HC11D0<br>MC68HC11D3 | 0<br>4K    | 192<br>192 | 0<br>0     | 14<br>32  | No         |            |
| <b>MC68HC711E9</b>       | <b>12K</b> | <b>512</b> | <b>512</b> | <b>38</b> | <b>Yes</b> |            |
| MC68HC11E0<br>MC68L11E0  | 0          | 512        | 0          | 22        | Yes        |            |
| MC68HC11E1<br>MC68L11E1  | 0          | 512        | 512        | 22        |            |            |
| MC68HC11E8<br>MC68L11E8  | 12K        | 512        | 0          | 38        |            |            |
| MC68HC11E9<br>MC68L11E9  | 12K        | 512        | 512        | 38        |            |            |
| <b>MC68HC711E20</b>      | <b>20K</b> | <b>768</b> | <b>512</b> | <b>38</b> |            | <b>Yes</b> |
| MC68HC11E20              | 20K        | 768        | 512        | 38        |            | Yes        |
| MC68HC811E2              | 0          | 256        | 2K         | 38        | Yes        |            |
| MC68HC11F1<br>MC68L11F1  | 0          | 1K         | 512        | 30        | Yes        |            |
| <b>MC68HC711K4</b>       | <b>24K</b> | <b>768</b> | <b>640</b> | <b>62</b> | <b>Yes</b> |            |
| MC68HC11K0<br>MC68L11K0  | 0          | 768        | 0          | 37        | Yes        |            |
| MC68HC11K1<br>MC68L11K1  | 0          | 768        | 640        | 37        |            |            |
| MC68HC11K3<br>MC68L11K3  | 24K        | 768        | 0          | 62        |            |            |
| MC68HC11K4<br>MC68L11K4  | 24K        | 768        | 640        | 62        |            |            |
| <b>MC68HC711L6</b>       | <b>16K</b> | <b>512</b> | <b>512</b> | <b>46</b> |            | <b>Yes</b> |
| MC68HC11L0<br>MC68L11L0  | 0          | 512        | 0          | 30        |            | Yes        |
| MC68HC11L1<br>MC68L11L1  | 0          | 512        | 512        | 30        |            |            |
| MC68HC11L5<br>MC68L11L5  | 16K        | 512        | 0          | 46        |            |            |
| MC68HC11L6<br>MC68L11L6  | 16K        | 512        | 512        | 46        |            |            |
| <b>MC68HC711P2</b>       | <b>32K</b> | <b>1K</b>  | <b>640</b> | <b>62</b> | <b>Yes</b> |            |
| MC68HC11P2               | 32K        | 1K         | 640        | 62        | Yes        |            |

**Table 2** shows PSD devices that are recommended for use with M68HC11 and M68L11 family members.

**Table 2 Recommended Devices**

| Device   | EPROM | RAM  | I/O |
|----------|-------|------|-----|
| PSD311C1 | 32K   | —    | 19  |
| PSD311   | 32K   | 2048 | 19  |
| PSD411A1 | 32K   | 2048 | 40  |

**Table 3** shows typical twin-chip alternatives to particular M68HC711 or M68L711 systems.

**Table 3 Alternative System Configurations**

|                    | Device                  | ROM            | RAM            | EEPROM     | I/O       | A/D        |
|--------------------|-------------------------|----------------|----------------|------------|-----------|------------|
| <b>Single Chip</b> | <b>MC68HC711D3</b>      | <b>4K</b>      | <b>192</b>     | <b>0</b>   | <b>32</b> | <b>No</b>  |
| Twin Chip          | MC68HC11D0 + PSD311C1   | 32K            | 192            | 0          | 33        | No         |
| <b>Single Chip</b> | <b>MC68HC711E9/20</b>   | <b>12K/20K</b> | <b>512/768</b> | <b>512</b> | <b>38</b> | <b>Yes</b> |
| Twin Chip          | MC68HC11A0/1 + PSD311C1 | 32K            | 256            | 0/512      | 41        | Yes        |
| Twin Chip          | MC68HC11A0/1 + PSD311   | 32K            | 2304           | 0/512      | 41        | Yes        |
| Twin Chip          | MC68HC11D0 + PSD311C1   | 32K            | 192            | 0          | 33        | No         |
| Twin Chip          | MC68HC11E0/1 + PSD311C1 | 32K            | 512            | 0/512      | 41        | Yes        |
| Twin Chip          | MC68HC11ED0 + PSD311C1  | 32K            | 512            | 0          | 33        | No         |
| <b>Single Chip</b> | <b>MC68HC11K4</b>       | <b>24K</b>     | <b>768</b>     | <b>640</b> | <b>62</b> | <b>Yes</b> |
| Twin Chip          | MC68HC11K0/1 + PSD411A1 | 32K            | 2816           | 0/640      | 77        | Yes        |
| Twin Chip          | MC68HC11K0/1 + PSD311   | 32K            | 816            | 0/640      | 56        | Yes        |
| <b>Single Chip</b> | <b>MC68HC11L6</b>       | <b>16K</b>     | <b>512</b>     | <b>512</b> | <b>46</b> | <b>Yes</b> |
| Twin Chip          | MC68HC11L1 + PSD311C1   | 32K            | 512            | 0/512      | 49        | Yes        |

### ADD THE PSD TO THE DESIGN

Migration from an M68HC711 single-chip system to an M68HC11/PSD system can be accomplished in one of three ways.

1. By building a daughter board that plugs into the MCU socket on an existing printed circuit board. The board includes the M68HC11, the PSD, and system clock generation circuitry. Including the clock generator on the daughter board is important to minimize radiated EMI.
2. By placing an edge or row connector on an existing printed circuit board to allow access to a PSD device on a daughter board. The minimum signals needed include the address/data lines and control signals (R/W, E, AS, RESET). This requires changing the existing schematic.
3. By redesigning the existing printed circuit board to accommodate the PSD device.

**Figure 1** and **Figure 2** are examples of interfacing an M68HC11 to particular PSD devices. Please refer to the appropriate Motorola data book and to the WSI *PSD Design and Applications Handbook* for more information.



## CONFIGURE THE M68HC11 FOR EXPANDED MODE OPERATION

M68HC11 operating mode is determined by the logic state of the MODA and MODB pins during system reset or power up. To configure the MCU for expanded mode operation, make the reset state of the MODA pin HIGH by pulling it up to VDD through a pullup resistor.

## CONFIGURE THE PSD

PSD software must be used to configure the PSD. There are two different software packages available.

PSD-SILVER software supports the PSD3XX devices and includes the MAPLE and MAPPRO software modules which run under the DOS platform. MAPLE software is used to configure the PSD chip. It features simple menu driven commands for selecting different device configurations. It also provides mapping of the EPROM, SRAM, and chip select outputs into the user's address space, and locates the files to be programmed into the EPROM segments. MAPPRO enables the user to program PSDs on a WSI MagicPro III® programmer.

PSDsoft WS7001 or WS7002 software supports the PSD3XX, PSD4XX, and PSD5XX families and runs under MicroSoft® Windows® (PSD3XX support included in PSDsoft available Q295). It includes PSDlabel, PSD configuration, PSD compiler, PSDsilos III simulator, and PSD programming software. The PSDsoft environment allows design and simulation of the on-chip PLD logic under Data I/O ABEL®, PSD interface selections to any MCU, configuration of the I/O, and address mapping of the EPROM and SRAM memory, among other things.

PSD-to-M68HC11 interface configuration is simple and straightforward. Configuration is performed by selecting certain option bits in the PSD software package. For MC68HC11A, C, D, E, and L devices, the PSD is configured for multiplexed mode. For MC68HC11F, K, and P devices, the PSD is configured for non-multiplexed mode. For all versions of the M68HC11, the other option bits on the PSD device are set as follows: R/W and E mode, active high AS (ALE), active low  $\overline{\text{RESET}}$ , and combined memory mode. To complete the configuration process, PSD Ports A and B must be configured as general-purpose I/O, to replace M68HC11 Ports B and C, which are used for address and data lines when the MCU is operating in expanded mode.

For a better understanding of the M68HC11 to PSD interface configuration information, please refer to the pin descriptions section of the appropriate Motorola data book and to Table 5 and Figure 12 in WSI Applications Note 011 for PSD3XX devices, and the section beginning with Figure 12 in Applications Note 029 for PSD4XX/5XX devices.

## MAKE MEMORY MAP AND I/O PORT SELECTIONS

To convert from an M68HC711 system to a system that uses a ROM-less M68HC11 and a PSD, the content of the M68HC711 ROM must be transferred to PSD EPROM, and mapped externally. The default state of the ROMON bit in the CONFIG register of ROM-less M68HC11 devices is zero, so all accesses to the ROM address space automatically go external. Virtually no change in the MCU address map is required because PSD EPROM can be mapped anywhere on a block boundary using the address map menu in the PSD software.

For example, assume a PSD device with 32 Kbytes of EPROM is selected. The PSD311 has eight blocks of 4 Kbyte x 8 EPROM. Each block can be mapped on a 4-Kbyte block boundary in the address range as originally defined in the OTP application. The PSD411A1 has four blocks of 8 Kbyte x 8 EPROM, and each can be mapped to an 8-Kbyte block boundary. Please refer to the modes and memory section of the appropriate Motorola data book, to Figure 32 in WSI Applications Note 011 for the PSD3XX, and to Table 7 in WSI Applications Note 029 for the PSD4XX/PSD5XX devices.

For PSDs that include an additional 2 Kbyte x 8 SRAM, the SRAM can be mapped anywhere within the address space on a 2-Kbyte boundary to extend the SRAM already supplied on the M68HC11.

PSDs also offer from 19 to 40 configurable I/O pins that can replace I/O pins that are used for other purposes when the M68HC11 operates in expanded mode or enhance the function of the available ports. The programmable I/O is addressed via an offset from a base address that is selected in the PSD software. These

offsets are shown in Table 6 in the PSD3XX data sheets, Tables 21–23 in the PSD4XX data sheet, and in Tables 29–31 in the PSD5XX data sheet.

For example, the following steps must be performed to replace ports B and C on an M68HC711 with ports A and B on a PSD311 device.

1. Refer to the appropriate PSD data sheet to determine the correct offsets.
2. Set the CSIOPORT (CSP) base address of the PSD. The base address can be mapped to any boundary from 256 bytes to 2 Kbytes. In this example, the CSIOPORT base address starts at \$2000.
3. Port B on the M68HC11 is mapped to port A on the PSD311. For compatibility with port B on the M68HC11, which is an output-only port, port A on the PSD311 is set for output. This is accomplished by writing \$FF (output) to the PSD311 port A data direction register, located at \$2004 (offset four from base address).
4. Port C on the M68HC11 is mapped to port B on the PSD311. The direction of the individual I/O pins in PSD311 port B is determined by the definition in the original OTP application. The direction is set by writing to \$2005 (offset five from base address). To make a pin an input, the appropriate bit in the register must be cleared; to make a pin an output, the appropriate bit must be set.
5. To write data to PSD311 port A and port B pins, the data must be written to \$2006 for port A and to \$2007 for port B. Data from the PSD311 port A and port B pins must be read from \$2002 and \$2003, respectively.

Other M68HC11 resources, such as EEPROM, SRAM, vectors, and the control registers are mapped internally and do not require any memory map redirection.

#### **MODIFY M68HC11 CODE TO ADDRESS PSD MEMORY AND I/O**

Change M68HC11 I/O port addresses to match the port address at the appropriate offset from the specified PSD I/O port base address (CSIOPORT).

#### **INTEGRATE M68HC711 CODE WITH PSD CONFIGURATION DATA**

Code that would normally be programmed into M68HC711 EPROM must be merged with PSD configuration information to create one output file. This is done during the compile procedure in the PSD software. The single output file is then downloaded to an industry-standard programmer (or the WSI MagicPro III PC-compatible programmer) and used to program the PSD device.

#### **PROGRAM THE PSD**

The output file (filename.obj) generated from the PSD software compiler is now ready to be programmed into a device from one of the three PSD families (PSD3XX, PSD4XX, or PSD5XX). A list of programmer manufacturers that support the PSD devices can be obtained from a WSI sales office or sales representatives. Programmers which support the PSD devices are available from Data I/O, BP Microsystems, and Logical Devices.

## **CONCLUSION**


A single-chip Motorola M68HC711 control system can be quickly and easily converted to a system that uses a ROM-less M68HC11 and a WSI Programmable MCU peripheral. A small investment in hardware and software modification can provide an increase in system memory, expanded I/O, lower power consumption, greater design flexibility, and lower cost.

**NOTE:** This Motorola document is also known as Application Note 043 at WSI, Inc.

## NOTES



Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

**MOTOROLA** and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

**Literature Distribution Centers:**

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre, 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England

JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.

ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No.2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

# High Performance M68HC11 System Design Using The WSI PSD4XX and PSD5XX Families

by John Bodnar

## INTRODUCTION

This application note covers conversion from a single-chip MC68HC711K4 microcontroller (MCU) system to a two chip MC68HC11K1 + PSD412A1 combination. It is not intended to be a comprehensive guide to using Motorola M68HC11 microcontrollers with WSI PSD4XX or PSD5XX microcontroller peripherals. These flexible devices provide a wide array of features, many of which cannot be adequately discussed within the context of this note. Designers with a more general interest in this topic should examine published material available from both Motorola and WSI. These documents are listed under **REFERENCES**.

## GENERAL INFORMATION

M68HC11 K-series MCUs are highly integrated derivatives of the MC68HC11F1, the first member of the M68HC11 family with a non-multiplexed address and data bus. Features common to the K series include:

- M68HC11 CPU core capable of dc to 4 MHz operation
- Power-saving STOP and WAIT modes
- 768 bytes of SRAM, with separate standby power input for battery backup
- Four programmable chip selects with clock stretching for expanded mode interfacing
- On-chip memory paging logic to allow expansion of the address space to 1 Mbyte
- 16-bit timer with programmable prescaler that includes 3 input capture (IC) channels, 4 output compare (OC) channels, and a single switchable IC or OC channel
- 8-bit pulse accumulator (PAC)
- Four 8-bit pulse width modulation (PWM) timer channels, pairs of which can be concatenated into two 16-bit channels
- Real-time interrupt circuit (RTI)
- Computer operating properly (COP) watchdog and clock monitor circuits
- Eight channel 8-bit analog-to-digital converter (ADC)
- Enhanced asynchronous non-return to zero serial communications interface (SCI)
- Enhanced synchronous serial peripheral interface (SPI)
- Maximum of 54 bits of bi-directional I/O available in single-chip mode of operation
- 8-bit fixed input-only port

The MC68HC11K4 device provides 24 Kbytes of masked ROM, 768 bytes of SRAM, and 640 bytes of EEPROM. Cost-reduced versions of this device are available without EEPROM and/or masked ROM. The MC68HC711K4 is functionally equivalent to the MC68HC11K4 but has 24 Kbytes of one-time programmable or UV-erasable EPROM instead of masked ROM. This programmable memory is typically used for prototyping, just-in-time inventory management, and applications requiring small production quantities or frequent code updates.



The MC68HC711K4 provides a great deal of flexibility for single-chip applications, but in some instances, it may be necessary to find an alternate solution that provides equivalent functionality. These situations may arise because:

- Pins used by the on-chip peripherals are also used to implement digital I/O ports, so use of peripherals can limit the available discrete digital I/O.
- Addition of new features may increase object code size beyond the 24 Kbytes provided by the internal EPROM. None of the M68HC11 K-series devices provide more than 24 Kbytes of ROM or EPROM, and expanded operating mode uses 25 digital I/O pins for the address/data bus and read/write line.
- Reduced software maintenance costs due to source code development in a high-level language can initially be offset by greater object code size, causing a memory crunch and loss of I/O resources.
- Some applications require peripherals that are either not available on an M68HC11 derivative or not available with an SCI- or SPI-compatible interface. These devices usually have an address/data bus, and must be memory mapped, causing a loss of I/O resources.

Some of these problems may seem insurmountable without extensive hardware and/or software redesign, but there is a solution that offers both the flexibility of expanded operating mode and the I/O preservation of single-chip operating mode.

WSI PSD4XX and PSD5XX programmable system devices (PSDs) are peripherals with flexible bus interfaces that provide microcontroller system designers an integrated memory solution consisting of SRAM, EPROM, and programmable logic. PSDs can also provide up to 40 digital I/O lines to replace those occupied by the MCU address/data bus. PSD4XX and PSD5XX devices share the following features:

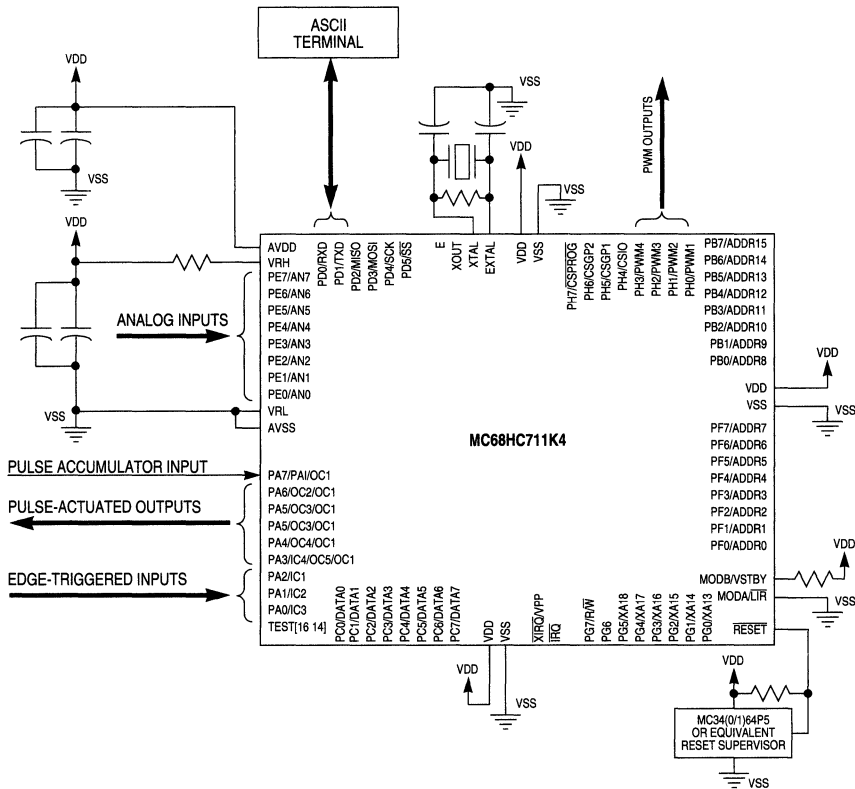
- Bus access speeds of 90, 120, 150, and 200 nanoseconds
- 37 (PSD4XXA1), 59 (PSD4XXA2), or 61 (PSD5XX) PLD inputs
- 113 (PSD4XXA1), 126 (PSD4XXA2), or 140 (PSD5XX) PLD product terms
- 8 (PSD4XXA1), 24 (PSD4XXA2), or 30 (PSD5XX) registered macrocells
- 40 bi-directional digital I/O pins
- Power management unit (PMU)
- 32 K x 8 (PSDX11), 64 K x 8 (PSDX12), or 128 K x 8 (PSDX13) of EPROM
- 2 K x 8 of SRAM

The PSD5XX family builds upon the PSD4XX family by adding a peripheral module that contains four 16-bit counters/timers, a watchdog timer, and an eight level interrupt controller. On all PSDs, some of the programmable logic is used to map the different memory blocks and control registers for the I/O ports, the PMU, and (on PSD5XX devices) peripheral control registers. Usually, sufficient programmable logic remains after memory decoding to implement chip-select signals for external memory mapped peripherals, other bus control signals, and even state machines to perform useful peripheral functions.

The discussion which follows covers the process of converting a hypothetical single-chip M68HC11 application to an equivalent or enhanced “two-chip solution” consisting of a non-multiplexed bus M68HC11 and a PSD4XX or PSD5XX device. Please refer to the **DEVICE REFERENCE TABLES**, at the end of this note, for a list of compatible devices. **Table 4** shows suitable non-multiplexed bus M68HC11 devices. **Table 5** shows PSD4XX and PSD5XX devices.

## THE PROBLEM

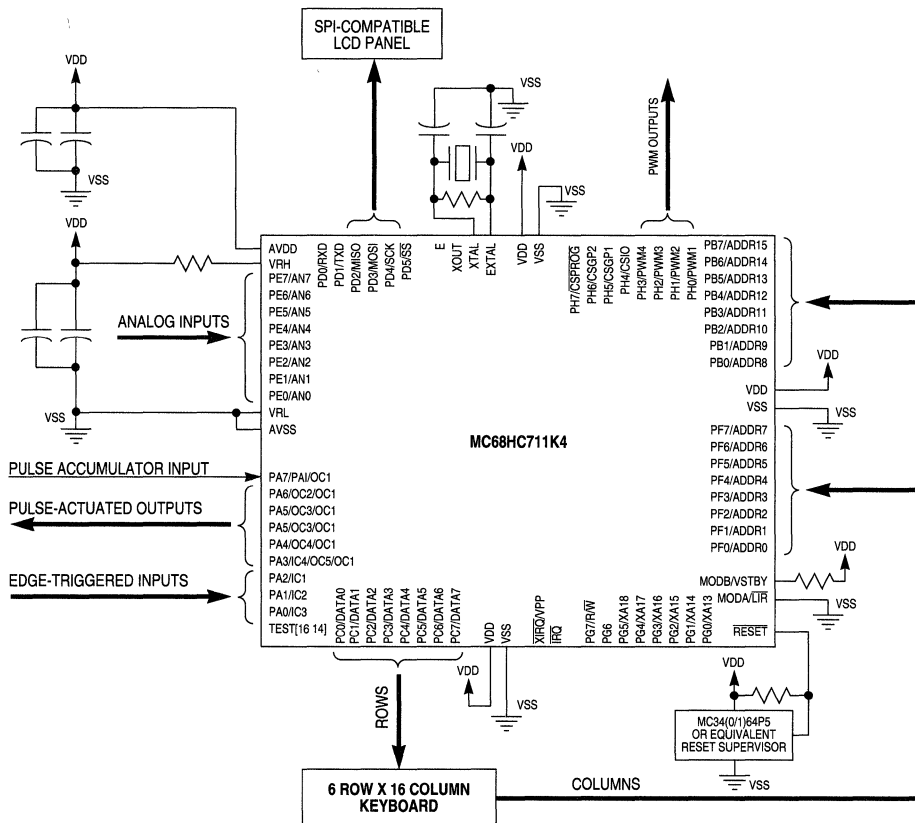
**Figure 1** shows a single-chip MC68HC711K4 application that makes extensive use of MCU on-chip peripheral and memory resources. An ASCII terminal interface that facilitates user interaction is an important feature of this design. However, the customer has requested that the next generation product be substantially smaller. This can best be achieved by eliminating the ASCII terminal.



AN1242 SCHEM 1

**Figure 1 Existing Single-Chip MC68HC711K4 Application**

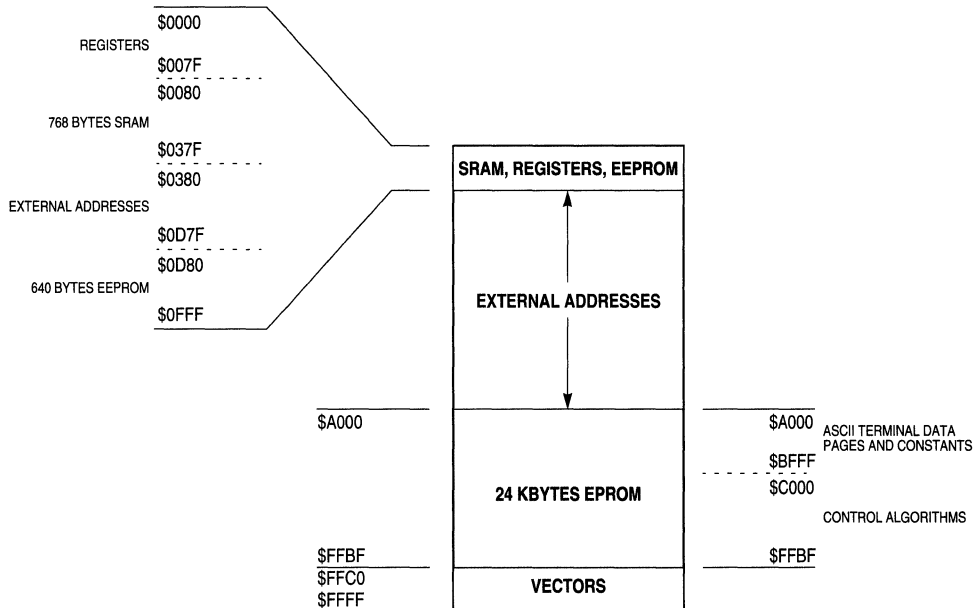
The redesign promising the greatest size reduction integrates a large LCD panel and keyboard with an M68HC11-based control unit. These changes meet customer requirements for a more compact, tightly integrated control solution, but as **Figure 2** indicates, the keyboard and LCD interfaces could exhaust the digital I/O resources of the MC68HC711K4.



AN1242 SCHEM 2

**Figure 2 Next Generation MC68HC711K4 Application With Keyboard and LCD**

To complicate matters further, these changes make greater demands of the MCU firmware. As shown in **Figure 3**, application control code and constant tables fit neatly into the 24 Kbytes of EPROM on the MC68HC711K4. The ASCII terminal connection to the SCI reduces the user interface to simple character I/O functions, but addition of a parallel input keyboard and a large LCD panel requires supplemental firmware support that increases permanent storage requirements to more than 24 Kbytes.

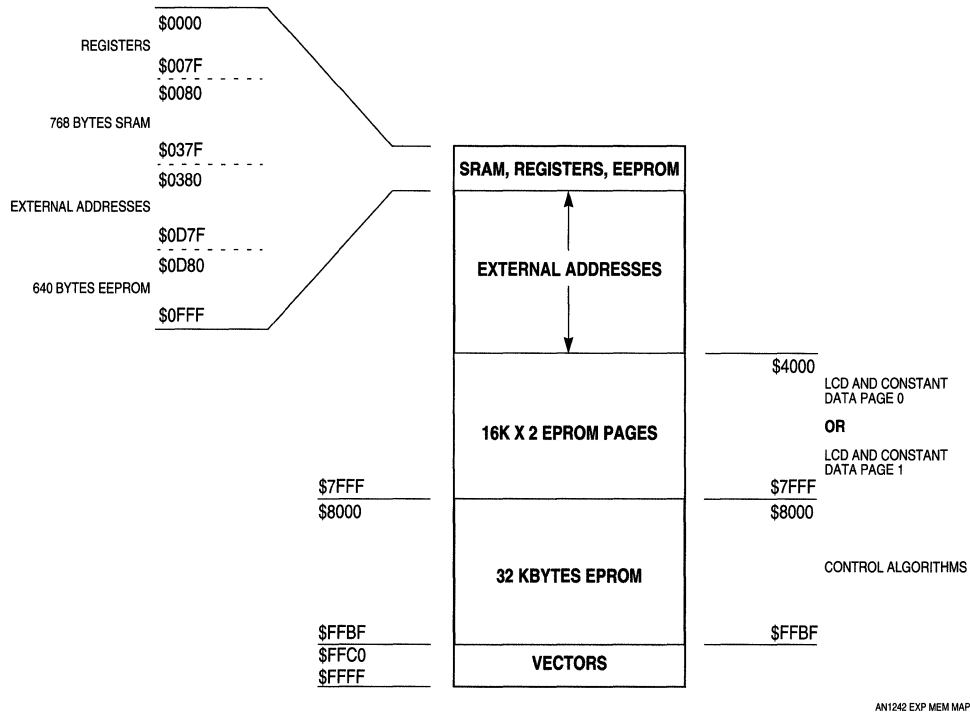


AN1242 SCM MEM MAP

**Figure 3 Initial Single-Chip Mode Memory Map**

Sophisticated control algorithms are required to support the increased functionality of the keyboard and LCD panel. To speed code development, reduce the cost of support, and provide for future enhancements, the firmware for the next generation product will be ported from M68HC11 assembly language to C. While high-level languages simplify the development of complex control applications, they do so at the expense of greater storage requirements. The reduction in object code size achieved by hand-tuning assembly language programs begins to disappear as application functionality and complexity increase.

One possible method of providing for increased storage demands would be to use the flexible memory expansion capabilities of an M68HC11 K-series device. This would lead to the simple expanded memory map shown in **Figure 4**. Estimates indicate that the control algorithms will require 32 Kbytes of EPROM and that the LCD data tables will require an additional 32 Kbytes. A 16-Kbyte memory paging window can be used to access the LCD data and still provide 12 Kbytes of contiguous address space for any other memory mapped peripherals that may be needed.



**Figure 4 Proposed Memory Map Expansion**

Unfortunately, access to the MC68HC711K4 address and data buses results in the complete loss of I/O ports B, C, and F, as well as bit 7 of port G. In addition, other port G bits would be used as expansion address lines and some port H bits may be used as chip selects. These lost I/O pins can be rebuilt with simple latches or more complex peripherals at the risk of decreased flexibility and more complicated circuit design and debugging.

## THE SOLUTION

Before proceeding with a design solution based on a WSI PSD4XX or PSD5XX device, it is instructive to review the problem as it now stands.

- The existing MC68HC711K4-based system makes extensive use of the integrated MCU peripheral resources. In particular, the SCI connects to an ASCII terminal that simplifies interactive user control.
- Size reductions specified for the next generation of this product are best achieved by replacing the ASCII terminal with a keyboard and LCD panel that are integrated with the control unit.
- The digital I/O requirements for the keyboard and LCD interfaces could exhaust the MC68HC711K4 I/O resources.
- Application storage demands increase for two reasons. The keyboard and LCD panel require additional interface code, and the firmware is to be ported from assembly language to C.
- To support the proposed expansion of storage capacity from 24 Kbytes to 64 Kbytes, the MCU must use expanded operating mode rather than single-chip operating mode.
- Expanded mode operation requires at least 25 pins for the non-multiplexed address/data bus and the read/write line. These pins are currently used for digital I/O. Other digital I/O pins must be used to support chip selects and memory expansion beyond 64 Kbytes.

It appears that the proposed system would require a 32 Kbyte EPROM for control routines, another 32 Kbyte EPROM for LCD data tables, several latches to rebuild lost I/O ports, and some programmable logic to map all of these devices into the MC68HC711K4 address space. This design clearly jeopardizes cost savings achieved by the existing implementation and future high-level language software development.

Some of the savings can be restored by switching MCUs. The 24 Kbyte EPROM on the MC68HC711K4 is not needed for the new design, so either the ROM-less MC68HC11K1 or the ROM-and-EEPROM-less MC68HC11K0 could be used. These devices retain the specialized peripherals available on the MC68HC(7)11K4, and are ideal for expanded mode applications where the I/O pins used by the bus interface are not required or are otherwise rebuilt.

Both cost reduction and increased flexibility can be achieved by using a WSI PSD4XX or PSD5XX programmable system device in place of the memory and logic components that would otherwise be needed to realize this design.

As shown in **Table 4** and **Table 5**, a PSD412A1 can easily provide the required additional memory, I/O, and logic resources. If subsequent specifications dictate increased memory, logic, or even peripheral functionality, other members of the PSD4XX and PSD5XX families could be used, while maintaining close compatibility with the PSD412A1.

**Table 1** compares the memory, I/O, and logic resources of both the initial MC68HC711K4 system and the proposed MC68HC11K(0/1) + PSD412A1 system.

**Table 1 M68HC11 Single-Chip vs M68HC11 + PSD4XX Resource Comparison**

|                              | MC68HC711K4 | MC68HC11K(1/0) + PSD412A1                     |
|------------------------------|-------------|-----------------------------------------------|
| EPROM                        | 24 Kbytes   | 16 Kbytes + 16 Kbytes + 32 Kbytes = 64 Kbytes |
| SRAM                         | 768 bytes   | 768 + 2048 = 2816 bytes                       |
| EEPROM                       | 640 bytes   | 640 bytes/0 bytes                             |
| Available bi-directional I/O | 54 lines    | 61 lines                                      |
| PLD input terms              | None        | 61                                            |
| PLD product terms            | None        | 113                                           |
| Registered macro cells       | None        | 8                                             |



In essence, the combination of a non-multiplexed bus M68HC11 MCU and a PSD4XX or PSD5XX device restores much of the functionality of a single-chip system. While not providing the ultimate size and power consumption features of such a design, the increased flexibility of this pairing and the freedom it provides to system designers is a competitive advantage.

## THE CONVERSION PROCESS

Converting a single-chip M68HC11 application to a two chip system consisting of a non-multiplexed bus M68HC11 and a PSD4XX or PSD5XX is a five step process:

1. Assess the memory, I/O, and logic requirements of the combined system
2. Select the appropriate M68HC11 and PSD combination.
3. Produce the two chip system memory map.
4. Determine which PSD I/O ports replace M68HC11 I/O ports used for expanded mode operation.
5. Generate a schematic for the combined system.

### 1. Assess the Memory, I/O, and Logic Requirements of the Combined System

This step has already been discussed. Key determinations to be made in this step include:

- How much I/O is required for the combined application?
  - Consider single-chip usage and any additional I/O that will be necessary for current and/or future product enhancements.
- How much memory is required for the combined application?
  - Consider potential firmware enhancements and the possibility of source code migration from assembly language to a high-level language like C or a visual application builder.
  - Also consider additional RAM requirements. PSD4XX and PSD5XX devices provide 2K x 8 of SRAM that can be powered from backup batteries, and future derivatives may eliminate the SRAM to reduce cost. If even more RAM is necessary, the PSD can provide the decode logic needed to memory map larger devices.
- How much logic will be required?
  - Any conversion to a two chip solution will use at least some of the PSD decode logic for memory, I/O port, and control register mapping. If the existing single-chip system uses PALs or 74HC family logic, consider using the PSD to replace as much of this as possible. The lower chip count reduces cost and use of the PSDsilos III™ simulation software can reduce system debug time.

### 2. Select the Appropriate M68HC11 and PSD Combination

Choose the M68HC11 and PSD combination carefully.

- When compatibility between the single-chip M68HC11 system and its PSD-based expanded mode counterpart is essential, use a ROM-less version of the single-chip MCU. In the example application, the MC68HC711K4 can be replaced with an MC68HC11K1 or MC68HC11K0 paired with the PSD412A1.
- In applications where maximum compatibility is not required, carefully selecting the M68HC11 MCU and PSD can realize considerable cost savings.
  - If the M68HC11 device is used because it has a large EPROM array, consider replacing it with a smaller ROM-less derivative. The PSD can be chosen to maximize available EPROM and I/O.
  - If the M68HC11 device is used to provide large amounts of I/O, choose the nearest equivalent ROM-less version and a PSD that will maximize available I/O.

- If the M68HC11 device is used for high-speed execution, consider using a smaller ROM-less derivative capable of the same performance. The PSD can be chosen to maximize available EPROM and I/O.
- If the M68HC11 device is used because it has a specific on-chip peripheral complement, choose the nearest equivalent ROM-less version and PSD that approximate the functionality of the single-chip device.
- Selection of an appropriate PSD is relatively straightforward. The device must meet the memory, I/O, and logic requirements determined in Step 1. If necessary, the MCU can be chosen to augment PSD resources, such as I/O and logic used for chip selects.

### 3. Produce the Two-Chip System Memory Map

This step is best explained by continuing with the example application. First, examine the memory map of the M68HC11 derivative to be used and locate areas not occupied by internal memory resources. These openings in the 16-bit address space are available for memory mapping external devices. The following ranges are externally addressable for MC68HC11K(0/1) devices:

\$0380 to \$0D7F

\$1000 to \$FFFF (\$2000 to \$FFFF if CSIO is used)

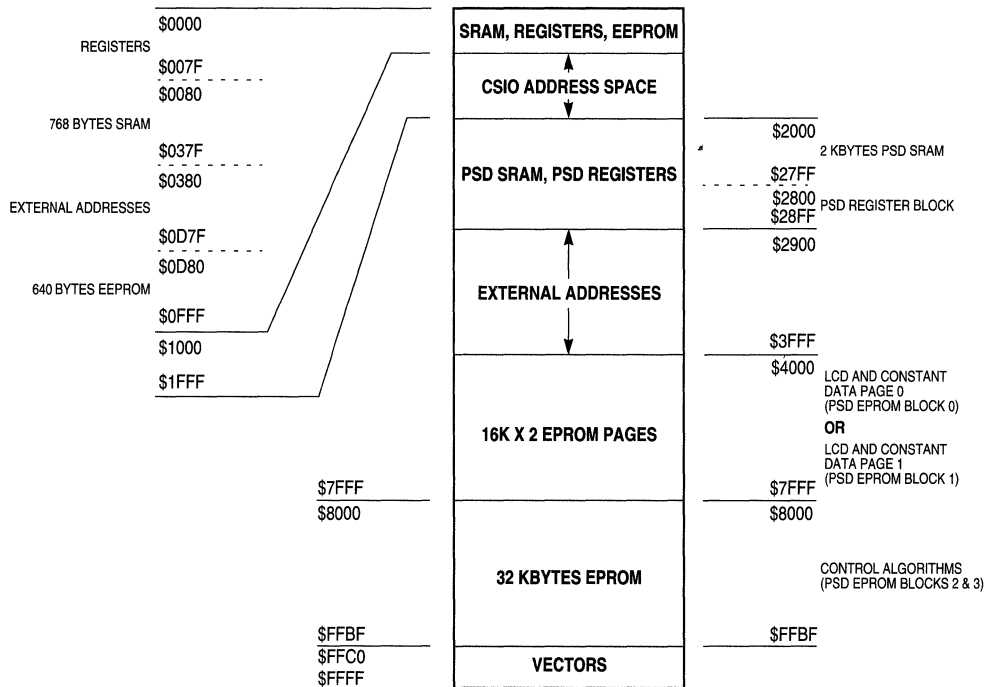
A 60-Kbyte block of space is available from \$1000 to \$FFFF in expanded operating mode. However, if the chip select I/O (CSIO) function implemented in M68HC11 K-series devices is used, this area is reduced to 56 Kbytes available from \$2000 to \$FFFF.

Allocating space to CSIO allows use of a memory-mapped display controller instead of a display controller with a serial or a parallel interface. A number of manufacturers provide a complete LCD solution that includes an intelligent display controller. The controller can be connected directly to a microcontroller address/data bus if slow access times can be managed. The CSIO signal is ideal for this purpose because it can be stretched by up to three E clock cycles. CSIO requires the fixed 4-Kbyte block of addresses from \$1000 to \$1FFF in order to operate.

Compile the memory map for the two chip system by listing the following address ranges:

- M68HC11 SRAM
- M68HC11 register block
- M68HC11 EEPROM, if used
- M68HC11 fixed chip-select address ranges, if used
- 256-byte PSD register block
- PSD SRAM, if used
- PSD EPROM blocks

**Figure 5** shows the combined memory map for the example application.



AN1242 PSD MEM MAP

**Figure 5 Combined MC68HC11K1 + PSD412A1 Memory Map**

The ultimate purpose of this memory map is to guide development of a PSDabel™ file. PSDabel is one component of WSI's comprehensive PSDsoft™ design package that also includes PSDcontrol™ (configuration, compilation, de-compilation, fitting, address translation, hex data file conversion, and device programming) and PSDsilos III™ (Verilog-based device simulation). PSDabel is based on Data I/O Corporation's ABEL Hardware Description Language. It is used to describe the logical operation of the PSD4XX and PSD5XX decode ZPLD (DPLD) and general-purpose ZPLD (GPLD).

A listing of the PSDabel file used to implement the memory map shown in **Figure 5** follows. The included comments provide a basic understanding of how a PSDabel file is constructed. Refer to the *PSDabel™ Manual* for further documentation and a tutorial.

```

module K4_TO_PSD_CONVERSION
title 'MC68HC711K4 to MC68HC11K1 + PSD412A1 Conversion'
"The following section lists the input signals.
"First come the address lines using their reserved names. Note that only those signals
"listed are routed to the DPLD.
a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;
"Next come the general purpose inputs used for the paging scheme. Uncomment the lines
"implementing the desired paging. For this application, the PSD page register will be used
"because it requires no additional I/O pins. The K1 memory expansion address lines may be
"used if additional address bits or the page register inputs are required for specific
"decoding purposes. Use of the page register will be discussed later.
pgr3,pgr2,pgr1,pgr0 pin; "These are the 4 input bits of the PSD page register.
xapage pin 20; "This is XA14 from the MC68HC11K1 and is used to select one of the 16K LCD
" data table pages.
"The M68HC11 non-multiplexed bus control signals are specified here.
rd_wr,e pin 29,41; "M68HC11 R/W* and E specified here as PSD pins 29 (WR) and 41 (RD).
"Now the DPLD chip select outputs are listed.
"CSIOP is the chip select for the PSD register block.
"RS0 is the chip select for the 2K PSD SRAM.
"ES[0:3] are the chip selects for PSD EPROM blocks 0, 1, 2, and 3.
csiop,rs0,es0,es1,es2,es3 node;
"Signal definitions and groupings now follow.
X = .x.; "This is how a don't care term is specified.
"This definition groups together the CPU address lines.
CPUaddress = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];
"This definition groups together the page register bits.
PAGE = [pgr3,pgr2,pgr1,pgr0];
"DPLD Chip Select Equations.
"This maps the PSD register block from $2800 to $28FF.
csiop = (CPUaddress >= ^h2800) & (CPUaddress <= ^h28FF);
"This maps the PSD 2K SRAM from $2000 to $27FF.
rs0 = (CPUaddress >= ^h2000) & (CPUaddress <= ^h27FF);
"This maps 16K PSD EPROM block 3 from $C000 to $FFFF.
es3 = (CPUaddress >= ^hc000) & (CPUaddress <= ^hFFFF);
"This maps 16K PSD EPROM block 2 from $8000 to $BFFF.
es2 = (CPUaddress >= ^h8000) & (CPUaddress <= ^hBFFF);
"This maps 16K PSD EPROM block 1 from $4000 to $7FFF when XA14 is logic one, i.e. this is LCD
" data table page 1. Use this equation when the K1 memory expansion is used in place of the
" PSD page register.
"es1 = xapage & (CPUaddress >= ^h4000) & (CPUaddress <= ^h7FFF);
"This maps 16K PSD EPROM block 1 from $4000 to $7FFF when PAGE = $1, i.e. this is LCD
" data table page 1. Do not use this equation if the K1 memory expansion is being used.
es1 = (PAGE == ^h1) & (CPUaddress >= ^h4000) & (CPUaddress <= ^h7FFF);
"This maps 16K PSD EPROM block 0 from $4000 to $7FFF when XA14 is logic zero, i.e. this is LCD
" data table page 0. Use this equation when the K1 memory expansion is used in place of the
" PSD page register.
"es0 = !xapage & (CPUaddress >= ^h4000) & (CPUaddress <= ^h7FFF);
"This maps 16K PSD EPROM block 0 from $4000 to $7FFF when PAGE = $0, i.e. this is LCD
" data table page 0. Do not use this equation if the K1 memory expansion is being used.
es0 = (PAGE == ^h0) & (CPUaddress >= ^h4000) & (CPUaddress <= ^h7FFF);
end K4_TO_PSD_CONVERSION

```

#### 4. Determine Which PSD I/O Ports Replace M68HC11 I/O Ports

PSD4XX and PSD5XX devices have five 8-bit I/O ports, labeled A, B, C, D, and E. When used with a non-multiplexed bus M68HC11, port C becomes the 8-bit data bus. Of the available 32 bits of general-purpose I/O, 24 are used to rebuild M68HC11 ports B, C, and F (the address/data bus), and the remaining eight can be used to rebuild port G bit 7 (the R/W line) and other port G or port H I/O pins used for expansion address lines or chip selects.

To modify existing single-chip M68HC11 software to take advantage of PSD I/O ports, simply substitute PSD register addresses for M68HC11 register addresses. A typical M68HC11 I/O port has both a data direction register and a port data register. Every PSD I/O port has a control register that determines port function, a data direction register, a data in register, and a data out register. Some PSD I/O ports also have registers to enable open drain operation, to determine if a pin is used as a PLD signal or I/O bit, and to read the outputs of the GPLD.

A good way to view the port relationships between a PSD and an M68HC11 is to construct a table that lists each port and its associated registers. On one side of the table, list the M68HC11 I/O port and its registers, and on the other side, list the equivalent PSD I/O port and its registers. Use this table as a guide when modifying single-chip firmware to support the two chip M68HC11/PSD system. **Table 2** is an I/O mapping table for the example application. Remember that CSIOP is mapped from \$2800 to \$28FF.

**Table 2 M68HC11 to PSD I/O Conversion Table**

| MC68HC711K4           |         |                  | PSD412A1 |            |        |
|-----------------------|---------|------------------|----------|------------|--------|
| Port B                | DDRB    | \$0002           | Port B   | PB_DDR     | \$2807 |
|                       | PORTB   | \$0004           |          | PB_INDATA  | \$2801 |
|                       |         |                  |          | PB_OUTDATA | \$2805 |
|                       |         | PB_CONTROL       |          | \$2803     |        |
|                       |         | PB_PLD_IO        |          | \$280B     |        |
|                       |         | PB_MAC_OUT       |          | \$280D     |        |
| Port C                | DDRC    | \$0007           | Port A   | PA_DDR     | \$2806 |
|                       | PORTC   | \$0006           |          | PA_INDATA  | \$2800 |
|                       |         |                  |          | PA_OUTDATA | \$2804 |
|                       |         | PA_CONTROL       |          | \$2802     |        |
|                       |         | PA_PLD_IO        |          | \$280A     |        |
|                       |         | PA_MAC_OUT       |          | \$280C     |        |
| Port F                | DDRF    | \$0003           | Port E   | PE_DDR     | \$2826 |
|                       | PORTF   | \$0005           |          | PE_INDATA  | \$2820 |
|                       |         |                  |          | PE_OUTDATA | \$2824 |
|                       |         | PE_CONTROL       |          | \$2822     |        |
|                       |         | PE_PLD_IO        |          | \$282A     |        |
|                       |         | PE_MAC_OUT       |          | \$282C     |        |
| Occupied Port G/H I/O | DDRG/H  | \$007F or \$007D | Port D   | PD_DDR     | \$2817 |
|                       | PORTG/H | \$007E or \$007D |          | PD_INDATA  | \$2811 |
|                       |         |                  |          | PD_OUTDATA | \$2815 |
|                       |         | PD_CONTROL       |          | \$2813     |        |
|                       |         | PD_OPN_DRN       |          | \$2819     |        |

As **Table 2** indicates, the location of each PSD register is specified as an 8-bit offset from the CSIOP base address specified in the PSDlabel file. The PSD4XX and PSD5XX documentation lists these 8-bit offsets.

A few small differences in I/O functionality should be noted:

- I/O ports on some M68HC11 devices have assignable pull-up resistors. For example, the PPAR register at \$002C on M68HC11 K-series MCUs can enable pull-up devices on ports G and H in all modes and on ports B and F only in single-chip mode. This feature is not available on PSD4XX or PSD5XX devices, so external pull-ups may be needed.
- I/O ports B, C, and F on M68HC11 K-series MCUs do not have any sort of control or alternate function registers, although port C can be placed in open drain mode with the CWOM bit in the OPT2 register at \$0038. If this functionality must be maintained, replace port C on the MC68HC711K4 with PSD4XX or PSD5XX port D. The open drain control register (PD\_OPN\_DRN in the table above) allows each PSD port D I/O pin to be configured for normal or open drain mode.
- The PGAR register at \$002D is used to enable the memory expansion address lines associated with port G bits 0 to 5. Setting bits in this register to one overrides the port G I/O functions and enables the associated XA lines. This register is cleared to \$00 after reset.
- Chip select control registers CCTL, GPCS1A, and GPCS2A, located respectively at \$005B, \$005C, and \$005E, override the I/O functions of port H bits 4 to 7. In expanded operating mode, GPCS1A and GPCS2A are set to \$00 after reset, thus disabling general-purpose chip selects 1 and 2 (CSGP1 and CSGP2). CCTL will be set to \$04 after reset, leaving the I/O chip select (CSIO) disabled and the program chip select (CSPROG) enabled. Write CCTL to \$00 to disable CSPROG and make the PH7/CSPROG pin available for I/O. CSPROG is not required for interfacing to the PSD, although it can be used in conjunction with the PSD power management unit (PMU) to reduce power consumption.

The code examples that follow demonstrate how the PSD I/O ports are accessed in comparison with M68HC11 I/O ports. Access to the other PSD control registers is achieved in the same straightforward fashion. Please refer to PSD4XX and PSD5XX documentation for more information.

**Single-Chip MC68HC711K4**

```

*
* port B, C, and F I/O
*
REGBASE equ $0000
DDRB equ $02
PORTB equ $04
DDRC equ $07
PORTC equ $06
DDRF equ $03
PORTF equ $05
*
* read port B[7:0]
*
 clr REGBASE + DDRB
 ldaa REGBASE + PORTB
*
* write pattern to port C[7:0]
*
 ldaa #$FF
 staa REGBASE + DDRC
 ldaa #$55
 staa REGBASE + PORTC
*
* configure PF[3:0] for inputs,
* PF[7:4] for outputs, poll until PF0
* is set to 1, then write pattern to
* PF[7:4].
*
 ldx #REGBASE
 ldaa #$F0
 staa DDRF,X
POLLPF0 brclr PORTF,X,$01,POLLPF0
 bset PORTF,X,$A0

```

**MC68HC11K(0/1) + PSD412A1**

```

*
* port B, A, and E I/O
*
REGBASE equ $2800
PB_DDR equ $07
PB_INDATA equ $01
PB_OUTDATA equ $05
PB_CONTROL equ $03
PA_DDR equ $06
PA_INDATA equ $00
PA_OUTDATA equ $04
PA_CONTROL equ $02
PE_DDR equ $26
PE_INDATA equ $20
PE_OUTDATA equ $24
PE_CONTROL equ $02
*
* make ports B, A, and E exclusively
* available for I/O
*
 ldaa #$FF
 staa REGBASE + PB_CONTROL
 staa REGBASE + PA_CONTROL
 staa REGBASE + PE_CONTROL
*
* read port B[7:0]
*
 clr REGBASE + PB_DDR
 ldaa REGBASE + PB_INDATA
*
* write pattern to port A[7:0]
*
 ldaa #$FF
 staa REGBASE + PA_DDR
 ldaa #$55
 staa REGBASE + PA_OUTDATA
*
* configure PE[3:0] for inputs,
* PE[7:4] for outputs, poll until PE0
* is set to 1, then write pattern to
* PE[7:4].
*
 ldx #REGBASE
 ldaa #$F0
 staa PE_DDR,X
POLLPE0 brclr PE_INDATA,X,$01,POLLPE0
 bset PE_OUTDATA,X,$A0

```

## 5. Generate a Schematic for the Combined System

**Table 3** shows the connections between a non-multiplexed bus M68HC11 and a PSD4XX or PSD5XX

**Table 3 M68HC11 to PSD Connections**

| M68HC11    | PSD4XX or PSD5XX |
|------------|------------------|
| ADDR[15:0] | ADIO[15:0]       |
| DATA[7:0]  | PC[7:0]          |
| E          | RD               |
| R/W        | WR               |

An expansion address line (XA14) could be connected to one of the PSD port A inputs, and used to select the 16-Kbyte LCD table EPROM pages. In the example application, however, it is easier to use the PSD page register. The four page register bits (PG[3:0]) can be used as inputs to the DPLD. In the example PSDabel listing, the ES0 and ES1 EPROM chip selects are decoded when the page register value is \$0 or \$1 and the CPU address is between \$4000 and \$7FFF.

The page register is accessed as follows.

```
REGBASE equ $2800
PSD_PAGE equ $E0
PAGE0 equ $00
PAGE1 equ $01
LCD_LINE1 equ $4000
LINE_LEN equ $F0
*
* select EPROM page 0/LCD table 0
*
 ldaa #PAGE0
 staa REGBASE + PSD_PAGE
*
* read data from selected page
*
 ldx #LCD_LINE1
 ldab #LINE_LEN
SEND_L1 ldaa 0,X
 jsr SEND_DATA
 inx
 decb
 bne SEND_L1
 .
 . etc.
 .
```

The page register bits are available as inputs to both the DPLD and the GPLD. In fact, the DPLD can generate two additional chip selects called PSEL0 and PSEL1 that can be used to connect other peripheral devices to the combined system. Using the page register, these devices could be mapped into the \$4000 to \$7FFF range used for EPROM blocks 0 and 1. If a more complex decoding function is needed, the GPLD and its associated macrocells can be used.



## CONCLUSION

**Figure 6** shows the newly-enhanced system, which has plenty of free general-purpose I/O to handle a large parallel interface keyboard. A number of different LCD solutions can be supported — the choices range from simple I/O driven devices to complete intelligent controller-based displays with synchronous serial or memory mapped interfaces. The system is capable of meeting next generation product specifications with room to spare for future expansion.

Highly integrated M68HC11 derivatives, such as the MC68HC711K4, can often serve as complete solutions for single-chip embedded control systems. Cost-effective designs with these devices make extensive use of on-board peripherals like the SCI, SPI, timer, and A/D converter. However, an application can outgrow the original design, and when this happens, it may be difficult to find an enhanced derivative to meet new peripheral and memory requirements.

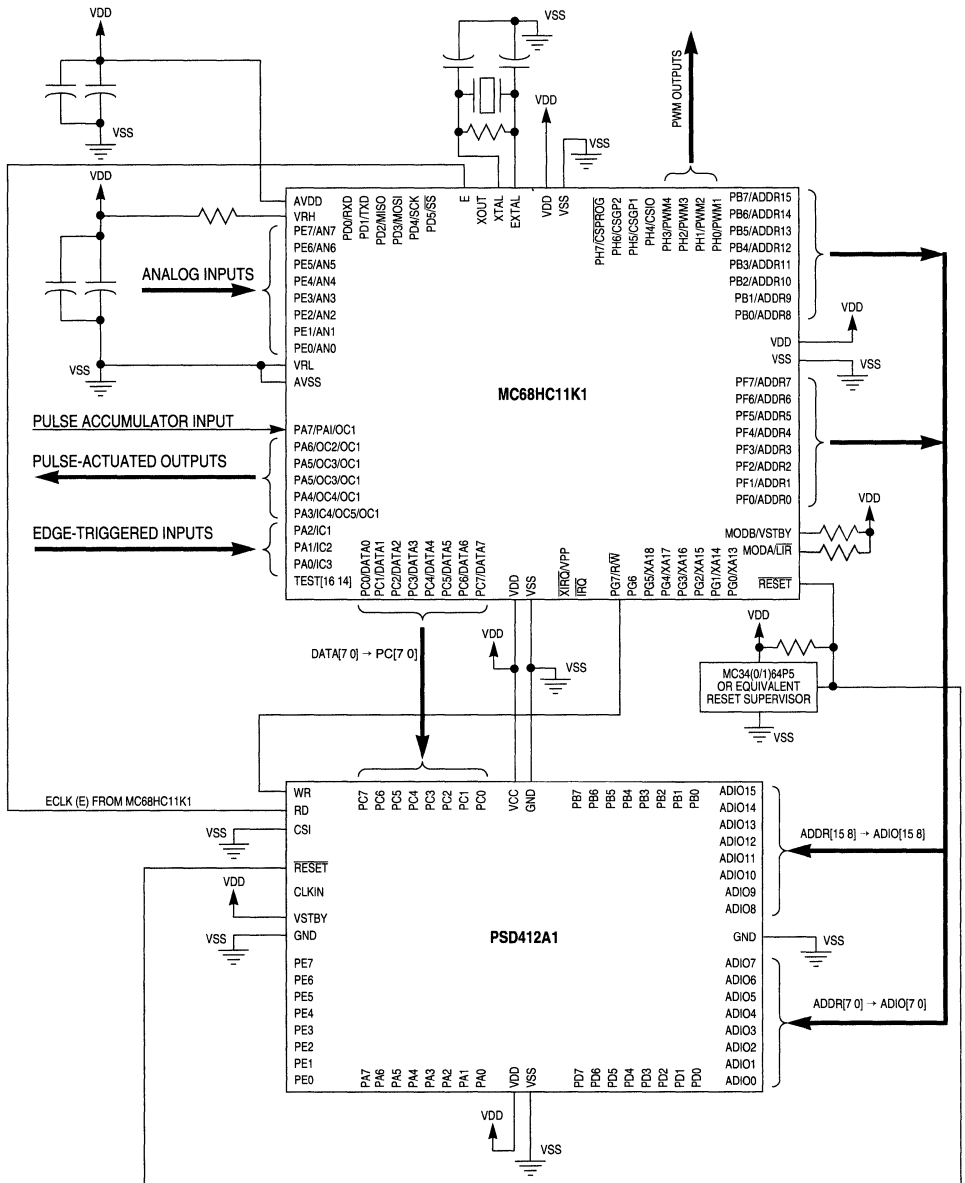
To solve this problem, users of high performance M68HC11 devices can pair a ROM-less M68HC11 derivative with a WSI PSD4XX or PSD5XX programmable system device. WSI's highly integrated microcontroller peripherals can deliver a cost-effective combination of EPROM, RAM, programmable logic, digital I/O, timer, and interrupt control modules. The M68HC11/PSD combination retains many advantages of the original single-chip MCU solution while providing a flexible resource complement for future application growth.

## REFERENCES

Motorola *MC68HC11K4 Technical Data Book* (MC68HC11K4/D)

Motorola *MC68HC11K4 Programming Reference Guide* (MC68HC11K4RG/D)

WSI *PSD Programmable Peripherals Design and Applications Handbook*.



AN1242 SCHEM 3

Figure 6 Keyboard and LCD Ready MC68HC11K1 + PSD412A1 System

## DEVICE REFERENCE TABLES

**Table 4 M68HC11 Derivatives with Non-Multiplexed Address/Data Bus**

| Motorola Part Number | ROM or EPROM | RAM (Bytes) | EEPROM (Bytes) | Total I/O | On-Chip Peripherals                                             | Technical Data  |
|----------------------|--------------|-------------|----------------|-----------|-----------------------------------------------------------------|-----------------|
| MC68HC11F1           | 0            | 1024        | 512            | 30        | Standard <sup>1</sup><br>+ 4 chip selects                       | MC68HC11F1/D    |
| MC68HC11G5           | 16K          | 512         | 0              | 66        | Standard<br>+ 10-bit ADC<br>+ event counter                     | MC68HC11G5/D    |
| MC68HC711G5          | 16K          | 512         | 0              | 66        | Standard<br>+ 10-bit ADC<br>+ event counter                     | MC68HC11G5/D    |
| MC68HC11G7           | 24K          | 512         | 0              | 66        | Standard<br>+ 10-bit ADC<br>+ event counter                     | MC68HC11G5/D    |
| MC68HC11K0           | 0            | 768         | 0              | 37        | Enhanced <sup>2</sup><br>+ 4 chip selects<br>+ memory expansion | MC68HC11K4/D    |
| MC68HC11K1           | 0            | 768         | 640            | 37        | Enhanced<br>+ 4 chip selects<br>+ memory expansion              | MC68HC11K4/D    |
| MC68HC11K4           | 24K          | 768         | 640            | 62        | Enhanced<br>+ 4 chip selects<br>+ memory expansion              | MC68HC11K4/D    |
| MC68HC711K4          | 24K          | 768         | 640            | 62        | Enhanced<br>+ 4 chip selects<br>+ memory expansion              | MC68HC11K4/D    |
| MC68HC11KA4          | 24K          | 768         | 640            | 51        | Enhanced                                                        | MC68HC11KA4TS/D |
| MC68HC711KA4         | 24K          | 768         | 640            | 51        | Enhanced                                                        | MC68HC11KA4TS/D |
| MC68HC11P2           | 32K          | 1024        | 640            | 62        | Enhanced<br>+ 2 SCI+                                            | MC68HC11P2/D    |
| MC68HC711P2          | 32K          | 1024        | 640            | 62        | Enhanced<br>+ 2 SCI+                                            | MC68HC11P2/D    |

**NOTES:**

1. The standard peripheral complement consists of an 8-bit, 8 channel A/D converter (ADC), serial communications interface (SCI), serial peripheral interface (SPI), 16-bit timer with 3 or 4 input captures (ICs), 4 or 5 output compares (OCs), pulse accumulator, real-time interrupt, and computer operating properly (COP) watchdog monitor.
2. The enhanced peripheral complement improves on the standard peripheral complement with an SCI+ (enhanced SCI with parity generation and more flexible baud rate generator) and an enhanced SPI (additional baud rates and selectable bit shifting order) and four pulse width modulation (PWM) timers.


**Table 5 PSD4XX and PSD5XX Derivatives**

| WSI Part Number | Bus Width (Bits) | Inputs | Product Terms | Registered Macrocells | EPROM Density        |
|-----------------|------------------|--------|---------------|-----------------------|----------------------|
| PSD401A1        | x 8 or x 16      | 37     | 113           | 8                     | 32K x 8 or 16K x 16  |
| PSD411A1        | x 8              | 37     | 113           | 8                     | 32K x 8              |
| PSD402A1        | x 8 or x 16      | 37     | 113           | 8                     | 64K x 8 or 32K x 16  |
| PSD412A1        | x 8              | 37     | 113           | 8                     | 64K x 8              |
| PSD403A1        | x 8 or x 16      | 37     | 113           | 8                     | 128K x 8 or 64K x 16 |
| PSD413A1        | x 8              | 37     | 113           | 8                     | 128K x 8             |
| PSD401A2        | x 8 or x 16      | 59     | 126           | 24                    | 32K x 8 or 64K x 16  |
| PSD411A2        | x 8              | 59     | 126           | 24                    | 32K x 8              |
| PSD402A2        | x 8 or x 16      | 59     | 126           | 24                    | 64K x 8 or 32K x 16  |
| PSD412A2        | x 8              | 59     | 126           | 24                    | 64K x 8              |
| PSD403A2        | x 8 or x 16      | 59     | 126           | 24                    | 128K x 8 or 64K x 16 |
| PSD413A2        | x 8              | 59     | 126           | 24                    | 128K x 8             |
| PSD501B1        | x 8 or x 16      | 61     | 140           | 30                    | 32K x 8 or 16K x 16  |
| PSD511B1        | x 8              | 61     | 140           | 30                    | 32K x 8              |
| PSD502B1        | x 8 or x 16      | 61     | 140           | 30                    | 64K x 8 or 32K x 16  |
| PSD512B1        | x 8              | 61     | 140           | 30                    | 64K x 8              |
| PSD503B1        | x 8 or x 16      | 61     | 140           | 30                    | 128K x 8 or 64K x 16 |
| PSD513B1        | x 8              | 61     | 140           | 30                    | 128K x 8             |

PSD4XX and PSD5XX devices have SRAM that can be configured as 2K x 8 or 1K x 16, 40 I/O pins, and a power management unit (PMU). PSD5XX devices have a peripheral unit consisting of four 16-bit counters/timers, a watchdog timer, an eight-level interrupt controller, and programmable logic for memory mapping. All PSDs are available with access speeds of 90, 120, 150, or 200 nanoseconds.

**NOTE:** This Motorola document is also known as Application Note 044 at WSI, Inc.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

**MOTOROLA** and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.

ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No.2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.





---

***PSD3XX Family***

**1**

---

***ZPSD3XX Family***

**2**

---

***PSD4XX/5XX Family***

**3**

---

***Motorola Application Notes***

**4**

---

***Sales Representatives  
and Distributors***

**5**

# ***Section Index***

---

***Sales  
Representatives  
and Distributors***

|                                     |     |
|-------------------------------------|-----|
| Domestic<br>Representatives .....   | 5-1 |
| Domestic<br>Distributors .....      | 5-1 |
| International<br>Distributors ..... | 5-1 |
| WSI Direct<br>Sales Offices .....   | 5-1 |

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

---

# WSI Worldwide Sales, Service and Technical Support

## REPRESENTATIVES

### ALABAMA

Rep. Inc.  
Tel: (205) 881-9270  
Fax: (205) 882-6692

### ARIZONA

Summit Sales  
Tel: (602) 998-4850  
Fax: (602) 998-5274

### CALIFORNIA

Bager Electronics, Inc  
Tel: (714) 957-3367  
Fax: (714) 546-2654

Tel: (818) 712-0011  
Fax: (818) 712-0160

Earle Assoc., Inc  
Tel: (619) 278-5441  
Fax: (619) 278-5443

I Squared, Inc  
Tel: (408) 988-3400  
Fax: (408) 988-2079

Tel: (916) 989-0843  
Fax: (916) 989-2841

### CANADA

Inteltech, Inc  
Tel: (905) 629-0082  
Fax: (905) 629-1795

### COLORADO

Waugaman Associates, Inc  
Tel: (303) 423-1020  
Fax: (303) 467-3095

### CONNECTICUT

Advanced Tech Sales  
Tel: (508) 664-0888  
Fax: (508) 664-5503

### FLORIDA

QXI of Florida, Inc.  
Tel: (305) 341-1440  
Fax: (305) 341-1430

Tel: (407) 831-8131  
Fax: (407) 831-8112

Tel: (813) 894-4556  
Fax: (813) 894-3989

### GEORGIA

Rep. Inc  
Tel: (770) 938-4358  
Fax: (770) 938-0194

### IDAHO

Bager Electronics  
Tel: (801) 582-0501  
Fax: (801) 582-1850

### ILLINOIS

Victory Sales  
Tel: (847) 490-0300  
Fax: (847) 490-1499

### INDIANA

Victory Sales  
Tel: (317) 581-0880  
Fax: (317) 581-0882

### IOWA

Gassner & Clark Co  
Tel: (319) 393-5763  
Fax: (319) 393-5799

### KANSAS/NEBRASKA

Rush & West Associates  
Tel: (913) 764-2700  
Fax: (913) 764-0096

### KENTUCKY

Victory Sales  
Tel: (513) 436-1222  
Fax: (513) 436-1224

### MARYLAND/VIRGINIA

Robert Electronic Sales  
Tel: (410) 995-1900  
Fax: (410) 964-3364

### MASSACHUSETTS

Advanced Tech Sales, Inc  
Tel: (508) 664-0888  
Fax: (508) 664-5503

### MICHIGAN

Victory Sales  
Tel: (313) 432-3145  
Fax: (313) 432-3146

### MINNESOTA

OHMS Technology, Inc.  
Tel: (612) 932-2920  
Fax: (612) 932-2918

### MISSOURI

Rush & West Associates  
Tel: (314) 965-3322  
Fax: (314) 965-3529

### NEW JERSEY

Strategic Sales, Inc  
Tel: (201) 842-8960  
Fax: (201) 842-0906

### NEW MEXICO

BGR WYCK  
Tel: (609) 727-1070  
Fax: (609) 727-9633

### NEW MEXICO

S & S Technologies  
Tel: (602) 438-7424  
Fax: (602) 414-1125

### NEW YORK

Strategic Sales, Inc  
Tel: (201) 842-8960  
Fax: (201) 842-0906

Tri-Tech Electronics, Inc  
Tel: (716) 385-6500  
Fax: (716) 385-7655

Tel: (607) 722-3580  
Fax: (607) 722-3774

### NORTH CAROLINA

Rep. Inc  
Tel: (919) 469-9997  
Fax: (919) 481-3879

### OHIO

Victory Sales  
Tel: (216) 498-7570  
Fax: (216) 498-7574

Tel: (513) 436-1222  
Fax: (513) 436-1224

### OKLAHOMA

Bravo Sales, Inc  
Tel: (214) 250-2900  
Fax: (214) 250-2905

### OREGON

Electra Technical Sales  
Tel: (503) 643-5074  
Fax: (503) 526-2055

### PENNSYLVANIA

Victory Sales  
Tel: (216) 498-7570  
Fax: (216) 498-7574

BGR WYCK  
Tel: (609) 727-1070  
Fax: (609) 727-9633

### PUERTO RICO

QXI of Florida, Inc  
Tel: (305) 978-0120  
Fax: (305) 972-1408

### TENNESSEE

Rep. Inc.  
Tel: (423) 475-9012  
Fax: (423) 475-6340

### TEXAS

Bravo Sales, Inc  
Tel: (512) 328-7550  
Fax: (512) 328-7426

Tel: (214) 250-2900  
Fax: (214) 250-2905

Tel: (713) 955-6996  
Fax: (713) 955-7446

### UTAH

Bager Electronics  
Tel: (801) 582-0501  
Fax: (801) 582-1850

### WASHINGTON

Electra Technical Sales  
Tel: (206) 821-7442  
Fax: (206) 821-7289

### WISCONSIN

Victory Sales  
Tel: (414) 789-5770  
Fax: (414) 789-5760

OHMS Technology, Inc.  
Tel: (612) 932-2920  
Fax: (612) 932-2918

## DISTRIBUTORS

Arrow Electronics  
Avnet Electronics  
Marsh Electronics  
Port Electronics  
Time Electronics  
Vantage Components  
Wyle Laboratories  
Zeus Electronics

## WORLDWIDE

### AUSTRALIA

Zatek Components  
Tel: 61-2-744-5711  
Fax: 61-2-744-5527

Tel: 61-3-9574-9644  
Fax: 61-3-9574-9661

### BELGIUM, LUX

Alcom Electronics nv/sa  
Tel: 32-3-458-3033  
Fax: 32-3-458-3126

### BRAZIL/ARGENTINA

Colgil, Inc.  
Tel: 55-11-663285  
Fax: 55-11-663285

### CHINA

Comex Technology  
Tel: (86-10) 849-9430/8888  
Fax: (86-10) 849-9430

Tel: (86-811) 531-5258  
Fax: (86-811) 531-5258

Tel: (86-20) 380-7307/5688  
Fax: (86-20) 380-7307

Tel: (86-25) 449-1384  
Fax: (86-25) 449-1384

Microlink Intl' Co  
Tel: (602) 276-7808  
Fax: (602) 276-8211

Wuhan Liyuan Computer Ltd  
Tel: 86-27-7802986  
Fax: 86-27-7802985

### DENMARK

Jakob Hatteland A/S  
Tel: (45) 42-571000  
Fax: (45) 45-166199

### ENGLAND

Micro Call, Ltd.  
Tel: 44-184-426-1939  
Fax: 44-184-426-2998

Silicon Concepts, Ltd  
Tel: 44-1428-751-617  
Fax: 44-1428-751-603

### FINLAND

Avnet Nortec OY  
Tel: 358-0613181  
Fax: 358-06922326

### FRANCE

ASAP Composants  
Tel: 33 (1) 30-12-20-20  
Fax: 33 (1) 30-57-07-19

Microel  
Tel: 33 (1) 69-07-08-24  
Fax: 33 (1) 69-07-17-23

### GERMANY

Jermyn GmbH  
Tel: 49 (06) 431-5080  
Fax: 49 (06) 431-508289

Scantec GmbH  
Tel: 49 (089) 899-1430  
Fax: 49 (089) 857-6574

Topas Electronic GmbH  
Tel: 49 (0511) 968640  
Fax: 49 (0511) 9686464

### HONG KONG

Comex Technology Ltd  
Tel: 852-2735-0325  
Fax: 852-2730-7538

### INDIA/PAKISTAN

Pamir Electronics Corp.  
Tel: 610-594-8337  
Fax: 610-594-8559

### ISRAEL

Star-Tronics, Ltd  
Tel: 972-3-6960148  
Fax: 972-3-6960255

### ITALY

Compres SPA  
Tel: 39-3625781  
Fax: 39-362553967

Silverstar  
Tel: 39 2661251  
Fax: 39 266101359

### JAPAN

Internix, Inc.  
Tel: 813-3-369-1105  
Fax: 813-3-363-8486

Kyocera Corporation  
Tel: 813-3-708-3111  
Fax: 813-3-708-3372

Nippon Imex Corporation  
Tel: 813-3-321-8000  
Fax: 813-3-325-0021

### KOREA

Woo Young Tech Co., Ltd.  
Tel: 82-2-369-7099  
Fax: 82-2-369-7091

### NETHERLANDS

Alcom Electronics bv  
Tel: 31-10-451-9533  
Fax: 31-10-458-6482

### NEW ZEALAND

Apex Electronics  
Tel: 644-3853404  
Fax: 644-3853483

### NORWAY

Henaco A/S  
Tel: 47-22-16-21-10  
Fax: 47-22-25-77-80

### REPUBLIC OF SOUTH AFRICA

Sames (Pty) Ltd  
Tel: 2712-3336021  
Fax: 2712-3333158

### SINGAPORE

Technology Distribution(s) Pte. Ltd  
Tel: 65-299-7811  
Fax: 65-294-1518

### SPAIN, PORTUGAL

Matrx Electronica SL  
Tel: 34 1 5602737  
Fax: 34 1 5652865

### SWEDEN

DipCom Electronics  
Tel: 46-8-7522480  
Fax: 46-8-7513649

### SWITZERLAND

Elbatex  
Tel: (41) 56-43-75-11-11  
Fax: (41) 56-26-14-86

Laser & Electronic Equipment  
Tel: 41-1-4223330  
Fax: 41-1-4223458

### TAIWAN

Ally, Inc  
Tel: 886-2-768-6399  
Fax: 886-2-768-6390



## Corporate Headquarters

47280 Kato Road  
Fremont, California 94538-7333  
Tel: 510-656-5400 Fax: 510-657-5916  
800-TEAM-WSI (800-832-6974)  
In California 800-562-6363  
Web Site: <http://www.wsipsd.com>

## REGIONAL SALES

### Midwest

Hoffman Estates, IL  
Tel: (847) 215-2560  
Fax: (847) 215-2702

### Western Area

Irvine, CA  
Tel: (714) 753-1180  
Fax: (714) 753-1179

### Northeast

Trevose, PA  
Tel: (215) 638-9617  
Fax: (215) 638-7326

### Southeast

Dallas, TX  
Tel: (214) 418-2970  
Fax: (214) 418-2971

## EUROPE SALES

WSI - France  
2 Voie La Cardon  
91126 Palaiseau  
Cedex, France  
Tel: 33 (1) 69-32-01-20  
Fax: 33 (1) 69-32-02-19

## ASIA SALES

WSI - Asia, Ltd  
1006 C.C. Wu Bldg  
302-308 Hennessy Rd  
Wan Chai, Hong Kong  
Tel: 852-2575-0112  
Fax: 852-2893-0678  
Korea Branch  
Tel: 82-2-761-1281/2  
Fax: 82-2-761-1283







---

WaferScale Integration, Inc. (WSI) reserves the right to make changes without further notice to any products herein. WSI makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does WSI assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. WSI does not convey any license under its patent rights nor the rights of others. WSI products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the WSI product could create a situation where personal injury or death may occur. Should Buyer purchase or use WSI products for any such unintended or unauthorized application, Buyer shall indemnify and hold WSI and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that WSI was negligent regarding the design or manufacture of the part.

---

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.  
U.S.A: 4,328,565; 4,361,847; 4,409,723; 4,639,893; 4,649,520; 4,795,719; 4,763,184; 4,758,869; 5,006,974; 5,016,216; 5,014,097; 5,021,847; 5,034,786; 5,136,186; 4,939,392; 4,961,172

West Germany: 3,103,160  
Japan: 1,279,100  
England: 2,073,484; 2,073,487

---

PSDsoft is a trademark of WaferScale Integration, Inc.  
MagicPro and PSD301 are registered trademarks of WaferScale Integration, Inc.  
ABEL, ABEL-HDL, and ABEL-PLA are trademarks of Data I/O Corporation.  
Data I/O is a registered trademark of Data I/O Corporation.  
SIMUCAD and SILOS III are trademarks of SIMUCAD, Inc.  
IBM and IBM Personal Computer are registered trademarks of International Business Machines Corporation.  
PAL is a registered trademark of Advanced Micro Devices, Inc.

Copyright © 1996 WaferScale Integration, Inc. All Rights Reserved.



*47280 Kato Road  
Fremont, California 94538-7333  
Phone: 510/656-5400  
Fax: 510/657-5916  
800/ TEAM-WSI (800/832-6974)  
In California 800/562-6363*

*Printed in U.S.A. 6/96*