ARM6 PIE User Guide

ARM

Advanced RISC Machines

ARM Ltd, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, UK

Credits:	Carol Atack, Dave Flynn, Dave Jaggar, Pete Magowan, Stuart Avery Design
Document number:	ARMDUI-0001D
Issue date:	April 1995
Change Log:	July 1994: Minor modifications to text. April 95: Updates to schematics.

Contents

Overview 1

1.1 Welcome to the PIE

Welcome to the Advanced RISC Machines' Platform Independent Evaluation (PIE) card.

The PIE card is the first step in developing an embedded system and a convenient means of evaluating the Advanced RISC Machines' (ARM) family of RISC processors.

1.2 What the PIE is

The PIE card is a compact card which serves as a target board for the development of a RISC processor based embedded system. The PIE is also the evaluation vehicle for the ARM CPU macrocell for chip-level customer specific designs (ASICs). The board is based around the ARM60 processor which has been designed specifically for embedded applications.

The ARM PIE card includes :

- 512 Kbytes of SRAM, for data and program storage
- 128 Kbyte EPROM with monitor and self-test firmware
- 20 MHz clock supply (scalable for reduced power)
- Serial RS232 interface
- JTAG boundary scan test port, simplifying debugging
- Interface for logic analyser/system expansion
- User configurable bi-endian operation

The PIE card incorporates a simple industry standard RS232 interface which allows it to be connected to any computer which has a serial interface, giving access to a wide variety of host machines including :

- SUN
- IBM compatible PCs
- Apple Macintosh computers

1.3 Other requirements

To use the PIE you will also need a power supply unit, an appropriate serial cable, the ARM Software Development Toolkit. The PIE board is designed to be used in conjunction with the ARM Software Development Toolkit which, when running on a host

machine, fully supports code design and debugging. You may wish to refer to a copy of the ARM60 datasheet; please contact your local supplier.

The PIE card, in addition to allowing evaluation of the ARM processor, forms the basis of a custom embedded board. Throughout the design, standard commercially available parts have been used and a 0.1 inch grid pitch has been adopted.

This means that the PIE hardware can be easily expanded to suit the requirements of your specific application. To support this comprehensive documentation is provided.

1.4About this manual

This manual explains everything you need to know in order to get your PIE board up and running.

Chapter 2 describes the installation and setup procedure.

Chapter 3 describes the PIE hardware in detail and explains how to connect a logic analyser to the PIE.

Chapter 4 covers the Demon debugger contained in its EPROM.

Chapter 5 covers developing your custom system.

In addition, comprehensive technical appendices provide detailed timing diagrams, circuit schematics, GAL listings and mechanical dimensions.

Conventions

This manual employs several typographic conventions intended to improve its ease of use. Computer code which you need to enter or which is provided as an example is presented in a monospaced typewriter font to aid recognition.

State diagrams in chapters three and five employ their own conventions to depict logical states. **!** is used to mean NOT, **&** indicates a logical AND and the vertical bar **|** indicates a logical OR.

Like most electronic devices the PIE can be damaged by inappropriate handling or use. Such actions are warned against in the text and an arrow like the one on the left is used to draw your attention to them.

Getting started 2

2.1 In this chapter

This chapter explains how to set up the ARM60 PIE Card – either when demonstrating the ARM60 processor or when evaluating its suitability for embedded controller development.

Topics include unpacking and installing the card, establishing a communications link with a host computer, downloading and running software, and configuring the PIE.

The EPROM debug monitor, and how to install software on the host to communicate with the card, is described fully in the ARM Software Development Toolkit documentation.

2.2 System setup

To try out the ARM60 processor in its evaluation card, you need to obtain the relevant ARM Software Development Toolkit for your host system.

You also need a cable to connect the PIE to the host system. See Appendix A for details on how to wire the cable correctly. You will also need a +5 volt DC power supply, unless you intend to draw power from a PC motherboard.

2.3 Protecting your card

The card is shipped in a protective black box, where it sits between two layers of conductive plastic foam.

Break the quality seal to open the box. Note the alternative decode GAL chip for big-endian operation.

Take care not to subject the evaluation card to high electrostatic potentials. You may prefer to wear a grounding strap or similar protective device while handling the card.

Generally avoid touching the underneath (solder side) of the card, the component pins or any other metallic element. If passing to another person, place on foam in shipping box or other suitably antistatic material.

2.4 Inspecting the card

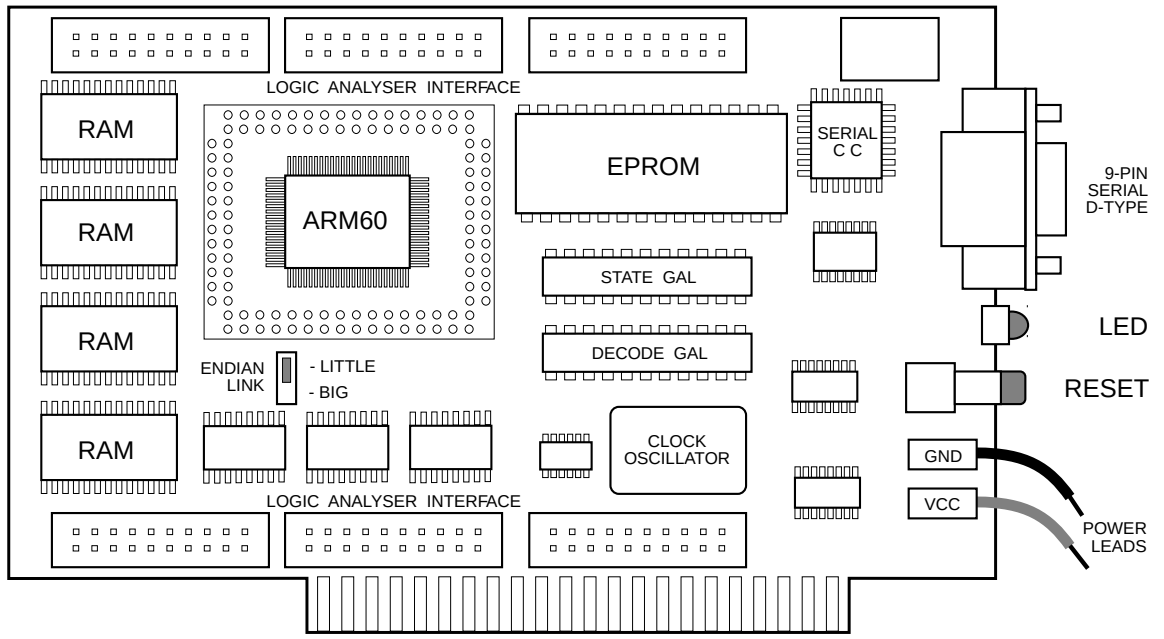
The arrangement of major components on the card is shown in Figure 2.1.

The following system components are clearly labelled on the card by name, and all components are identified by their code number shown on the schematic at the back of this manual:

- ARM60 Microprocessor
- EPROM
- RAM
- State GAL
- Decode GAL (little-endian)

Figure 2.1 shows also the six 20-pin connectors providing the logic analyser interface, the oscillator, serial communications controller and serial port, reset button and test LED, and the spade connectors to which you should attach power leads to an external power supply if used.

Figure 2.1 **Evaluation card**



2.5 Powering up

You can power your evaluation card by plugging it into a PC XT 8-bit expansion slot in your intended host system. (This may be convenient even when you are developing on a Sun or other non-PC host.) Alternatively, you can attach a suitable independent power supply.

PC power

The ARM evaluation card is a short PC card and is easily accommodated in IBM PC or compatible machines with a free expansion slot. Turn off the machine while inserting the card.

If you choose this option you will not need to attach power leads to the spade connectors.

Independent power

Power can also be supplied to the card via the two spade connectors adjacent to the reset button.

The card has no diode protection and so any power supply must provide the correct voltage and polarity:

- +5 volts DC 10%, 250 mA or gr eater

Check that the polarity of the power supply leads is correct and that the power is attached to V_{CC}.

2.6 Activating self-test

Once the card is powered up, you can activate the self-test facility. Press the reset button and note that the red LED lights for approximately one second, goes off for one second and then relights and stays on.

If the LED fails to light or if it fails to stay on, then the card is either faulty or incorrectly powered.

Reset button

Pressing the reset button causes the card to take the following actions:

- Checks the RAM
- Checks the ROM and its contents
- Checks the serial controller chip

After making these checks, the card outputs a diagnostic message (system configuration, RAM size etc) on the serial line and waits.

This procedure is also automatically activated on power up.

2.7 Connecting to a host

Serial cables are host-specific. You should build your cable according to the guidelines in Appendix A and the manual for your host system.

Plug the appropriate end of the cable into the serial port on the evaluation card. Connect the other end to the serial port on your host computer. If this is a small footprint Sun workstation, you will also need to use the adaptor cable supplied with the machine.

Installing ARM software on the host

Install the ARM Software Development Toolkit on your host computer as described in the associated user manuals.

With the ARM software correctly installed on the host, and the evaluation card powered up at the other end of the link, type the following command at the host keyboard to invoke the ARM Symbolic Debugger:

```
armsd -serial
```

The `-serial` option specifies that armsd should act as a front end for the PIE card.

See the Software Development Toolkit documentation for further information about the ARM Symbolic Debugger.

If both the link and the card are functioning correctly, then the EPROM Debug Monitor software (Demon) responds with a two-line message giving the results of self-test and stating the amount of RAM available on the card. If there is no response, recheck the serial connection then exit using CTRL-BREAK (or CTRL-C).

Running example programs

The Software Development Toolkit includes a directory of example programs you can download and run on the evaluation card before your own applications become available. Some are well-known benchmarks, for example:

- Dhrystone
- Whetstone
- Sieve

These programs allow you to examine the performance of the ARM60 processor and can be modified or extended in simple programming experiments.

Your own programs, written using the ARM Software Development Toolkit, can also be run on the PIE using the symbolic debugger ARMsd which forms part of the toolkit. The toolkit documentation contains further information on this.

Figure 2.2 shows an annotated example of running the Dhrystone program.

Figure 2.2 **Running the Dhrystone program**

```
>pisd dhryston                              - invoke PISD and load the program dhryston from
                                              the current directory
A.R.M. Source-level Debugger, version 4.01 (A.R.M.) [Feb 17, 1992]
ARM60, DEMON V1.00, 0x80000 bytes RAM, ROM CRC OK, Little endian, FPE
Object program file dhryston
Low Level Debugging information found
High Level Debugging information found
pisd: break main                            - set a breakpoint in the procedure main
pisd: go                                    - begin execution
Breakpoint #1 at #dhryston:main block 0, line 98 of dhryston.c
   98         Proc0();
pisd: registers                             - after the breakpoint has been reached,
                                              display the CPU registers
  r0  = 0x00000001  r1   = 0x0000f5b0    r2  = 0x004a5552  r3  = 0x0000f5b0
  r4  = 0x0000000a  r5   = 0x00000001    r6  = 0x00000f0a  r7  = 0x00000000
  r8  = 0x00000000  r9   = 0x0000000a    r10 = 0x0007f230  r11 = 0x0007ffd0
  r12 = 0x0007ffd4  r13  = 0x0007ffc4    r14 = 0x0000b704
  pc  = 0x000080a4  psr  = %nzCvif_User32
pisd: go                                    - resume execution
Dhrystone(1.1) time for 500000 passes = 68
This machine benchmarks at 7331 dhrystones/second
Program terminated normally
pisd: quit                                  - exit PISD
Quitting
>
```

2.8 Big-endian operation

The evaluation card is shipped in little-endian configuration. However many PIE users may find that big-endian configuration is more appropriate for their requirements, for example if interfacing to mixed processor systems.

To change to big-endian operation, reposition the two-position select jumper (JP 5 on the schematic and the board) southwards away from the ARM processor, and replace the decode GAL chip shipped on the card (U9) by the alternative GAL shipped with the card.

For ease of recognition the little-endian GAL's label contains the suffix -L and the big-endian GAL the suffix -B.

If, at a later date, little-endian operation is again required, reverse the procedure, moving the jumper north and replacing the -L decode GAL.

2.9 Using ARM resources in your design

This manual contains both the circuit description (and schematics) of the ARM60 Evaluation Card and the source code for the GAL equations. The Software Development Toolkit includes the source code for Demon (the Debug Monitor). These are provided to help you design your prototype target hardware systems.

Both the hardware and software are provided as tutorial aids and so serve to demonstrate techniques rather than an optimal implementation.

Whatever target prototype system you build, it is important that you include some form of communications channel that can support the Remote Debug Protocol (RDP), so that the ARM symbolic debugger ARMsd can still operate from the host.

The RDP is designed to operate over any 8-bit communications channel and asynchronous serial RS232 is used in the evaluation card because it is the most generally available host computer interface.

For example, a target laser printer controller with 4 Mbytes of DRAM would be better designed at the target hardware level with a fast download channel (eg a bidirectional printer port) for downloading fonts, raster images etc.

2.10 Using this manual

The PIE can be connected to a logic analyser. The section *Logic analysis* on page 23 discusses the facilities for this in detail.

Many PIE users will be developing their own hardware and Chapters 3 and 5 contain essential reference information on interfacing expansion hardware to the PIE. The section *Extending the PIE* on page 39 and *Expansion interface* on page 27 are particularly relevant to this process.

PIE Hardware 3

3.1 In this chapter

This chapter covers the PIE hardware in detail, including the circuit board design and the facilities for logic analysis.

The first section describes the components of the PIE and gives details of the way they work.

The second section describes the logic analysis interface and explains how to connect your own logic analyser to the card. Details are given of the signals which can be observed through each of the six connectors.

The final sections look at the PIE's I/O and expansion circuitry, including the serial communications facility.

You should read this chapter in conjunction with the circuit diagram schematic which forms part of Appendix C. You may also find a copy of the ARM60 datasheet useful.

3.2 The ARM60 PIE system

The ARM60 PIE system contains the following principal components:

- Clock generator
- ARM60 CPU
- Programmed logic state machine
- RAM subsystem
- ROM subsystem
- Logic analysis and expansion interface

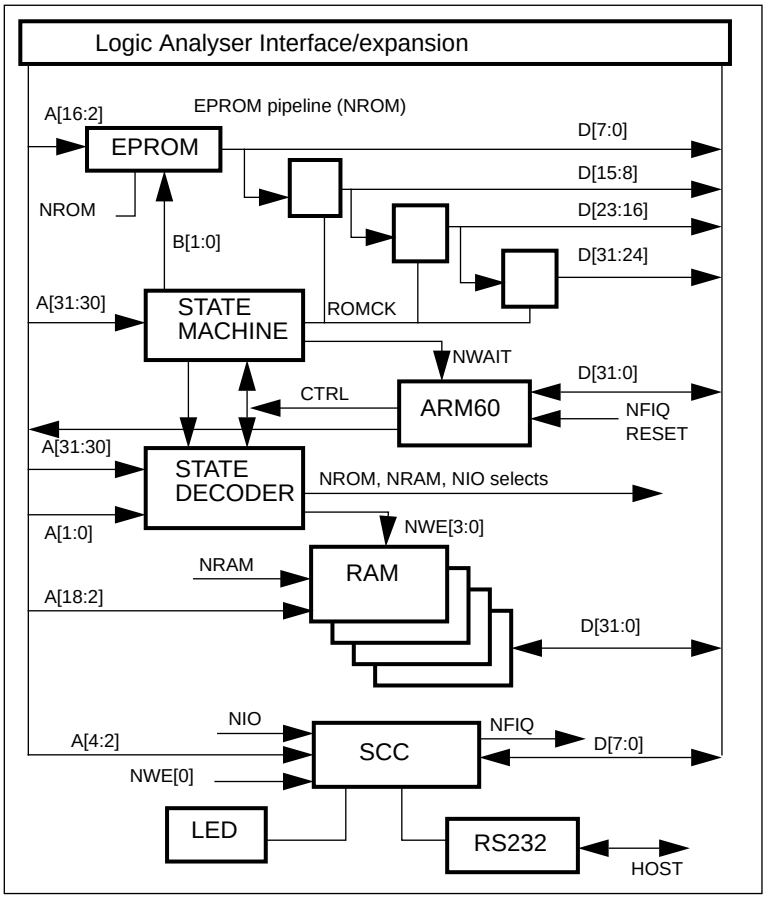
This section of the manual looks at each of these components in turn.

The basic architecure of the card is shown in figure 3.1:

The PIE board is configured for 32-bit address space, and the system memory map is divided into four 1-Gigabyte segments, with one of these reserved for external prototype extension. The memory map is illustrated in Figure 3.2.

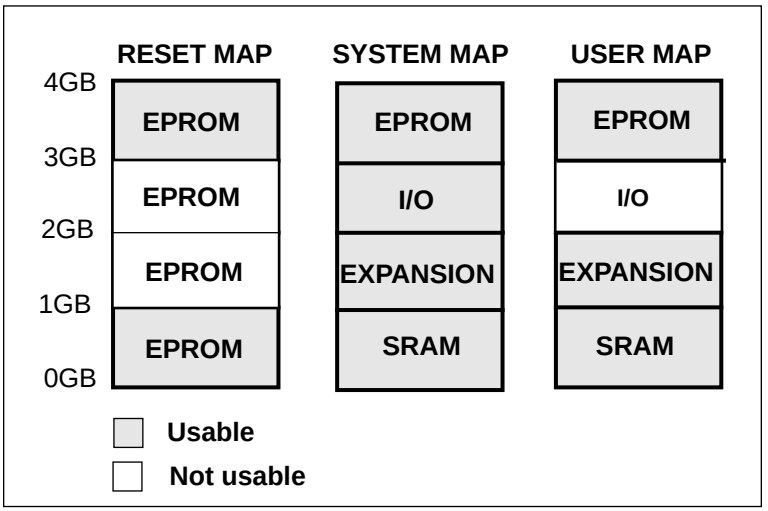
The normal memory map has RAM mapped at low memory, and the system EPROM and I/O space are both allocated to the high half of memory.

Figure 3.1 **Basic PIE architecture**



There is a special reset map, forced at hardware reset, which ensures that the image of the EPROM appears at the bottom of memory where the ARM reset vector must exist. The reset boot code switches the EPROM out of all other memory spaces after the first dummy write to the low quarter of memory space (RAM).

Figure 3.2 **System memory map**



This write does not actually write through to the RAM so that warm restart may be attempted.

3.2.1 The clock generator

The clock generator (marked U14 on the card) is an oscillator which provides the CPU and basic state machine clock.

The generator drives a fast invertor pack (U1) which produces the system clocks for the board. Resistors are used to isolate four principal clocks:

- SCLK is the state machine clock.
- ALE is the Address Latch Enable signal to ARM60.
- MCLK is the processor clock (to U7) which is gated with NWAIT.
- LCLK is a Logic Analyser clock exported (to PL6).

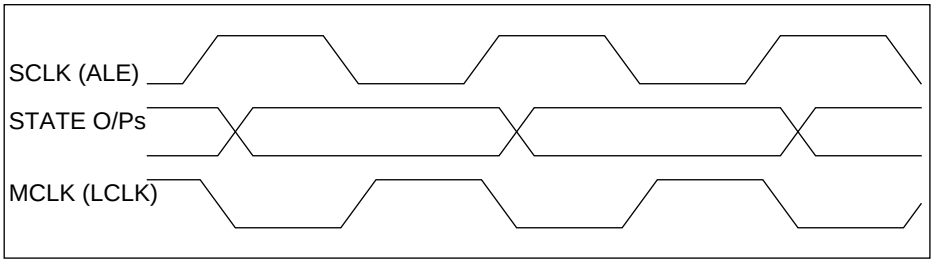
MCLK and LCLK are inverted versions of SCLK and ALE slightly delayed. The basic waveforms are shown in Figure 3.3.

ALE is used to hold addresses stable from the processor for the low phase of SCLK (the high phase of MCLK to the ARM60) to ensure that EPROM and SRAM addresses are held to the end of the bus cycle.

3.2.2 The ARM60 CPU–U7

The ARM60 CPU has separate 32-bit address and data buses which in this lightly loaded system are not buffered.

Figure 3.3 **Waveforms**



Loading should be kept to two CMOS loads plus the logic analyser probes for user prototyping purposes on the expansion connectors.

The processor clock, MCLK from the clock generator, is gated with the NWAIT output from the state machine (U6). The clock to the processor is inhibited when NWAIT is low. A simple AND gate keeps the core static and unclocked during wait cycles to save power.

Configuration

The ARM60 CPU is configured for full 32-bit data and program operation (CF3 and CF4 signals). Thus the system memory map is decoded by looking at the top two address bits (A31,A30).

The board is shipped with little-endian byte ordering by default but this may be changed with the shunt setting of the two-position jumper JP5 on the circuit board. You will also need to reconfigure the external memory system for big-endian operation by changing the decode GAL (U9)–see the previous chapter's section *Big-endian operation* on page 11.

Processor timing

The basic signals with the appropriate timings, marked using the symbols defined in the AC Parameters section of the ARM60 datasheet, are shown in Figure 3.4. The MCLK cycles to the processor are marked as inactive or active depending on whether NWAIT is low or high. Only when NWAIT is high is the clock gated through to the processor core, allowing the control signals to change.

The data cycle completes on (active) MCLK falling, around which input data must be setup and held.

The two forms of pipelined output, referring to the end of the data access cycle, are:

- NMREQ (Not Memory Request) and SEQ (Sequential), which are pipelined one full cycle ahead of the cycle to which they apply.
- Other control outputs, such as Not Read/Write (NRW) and Not Byte/Word (NBW), which are pipelined one half cycle ahead.

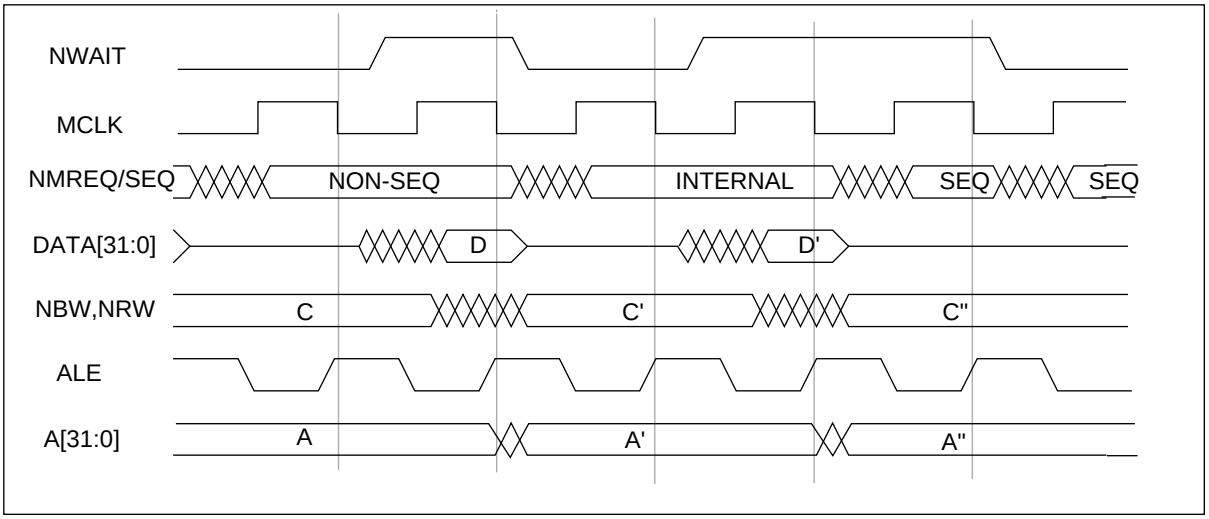
The use of the clocked ALE signal ensures that the addresses remain latched to the end of the data access cycle. There is a small guaranteed address latch fall-through time which provides address hold time to the system RAM and other writeable devices.

3.2.3 State machine and system decode–U6 and U9

The main state machine and associated system memory decoder are provided in the form of GAL (Generic Array Logic) devices. As such the devices may be reprogrammed using a standard GAL programming system and extended for customer evaluation of specific prototype hardware interfaces.

The standard GALs may also be replaced by pin-compatible zero power devices for use with low clock rate test systems.

Figure 3.4 **Processor timing**



The two programmed logic devices are clocked by the rising edge of SCLK and are tightly coupled, sharing the following inputs:

- A[31:30] the high order CPU address bits for memory space decoding.
- NMREQ and SEQ, the ARM60 CPU next memory access flags.
- NRW, the CPU read/write line.
- ABORT, from the external interface (for expansion).

The state machine GAL also uses the following signals:

- NRESET, which is used to put the ROM vectors in low memory.
- ERDY, from the expansion interface to stretch external memory access cycles.

The state machine–U6

The main state machine (U6) is a GAL20V8 device with all outputs registered, and provides the following system outputs to other devices:

RESMAP	maps the ROM to all memory at reset (cleared by software).
NWAIT	the processor wait input for slow accesses.
B[1:0]	the EPROM byte sequencing signals.
BCK	the EPROM byte pipeline sequence clock.
NIOS	the I/O cycle strobe timing (and shared as RAM output enable).

Other outputs provide state register flag bits (Note that STATE is exported to the logic analyser interface with NWAIT for system debugging).

Any non-sequential activity, or sequential writes to RAM, adds at least one clock period wait state to re-decode the address.

There are four basic states:

Decode: entered every non-sequential memory request cycle, and allowing the high-order addresses to be decoded for the next chip select cycle (burst). NWAIT is driven low to force the processor to wait a clock cycle. The next state is Wait, unless a very fast decode bank has been selected and the Memory state is chosen (if an

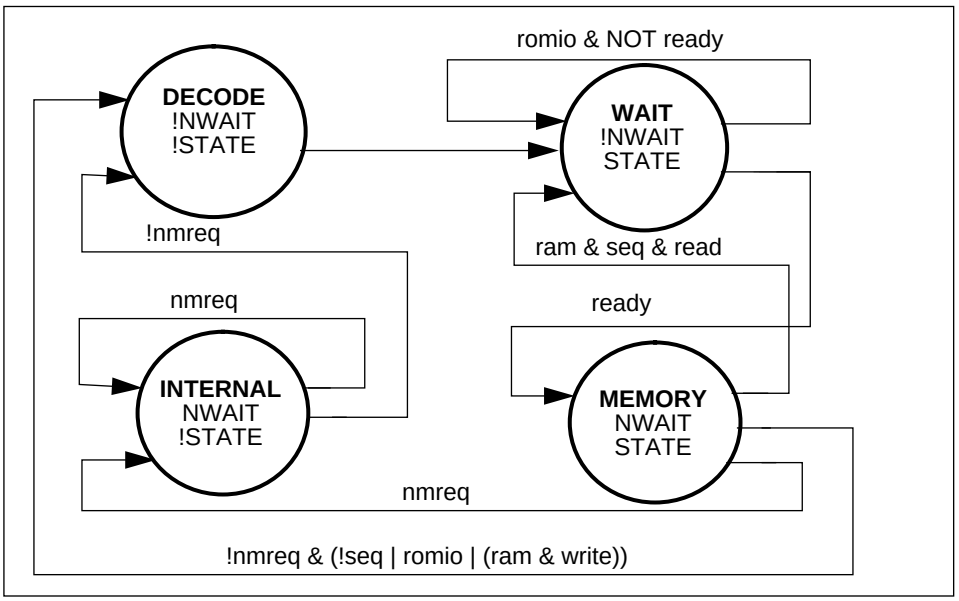
external device is ready and asserting the external ready signal ERDY).

Memory: entered during the last cycle of a memory access cycle, and de-asserts NWAIT to the processor for this cycle to allow the processor to complete the operation.

Internal: processor internal cycles run at full clock speed when memory is not requested; Internal state deasserts NWAIT to the processor to allow such high-speed operations for the duration of a multi-cycle internal operation (such as multiply).

Wait: entered for all memory request cycles that require the processor to add additional clock cycles before completing the memory cycle. NWAIT is driven low to force the processor to wait a clock cycle, and looping in this state extends the access period until changing to the Memory state when the external memory can complete the access in one further cycle.

Figure 3.5 **Basic state flow**



The basic state flow, illustrated in Figure 3.5, is:

DECODE fi WAITfi MEMORY fi DECODE/INTERNAL

For sequential bursts there is no need to redecode addresses:

WAIT fi MEMORY fi WAIT fi MEMORY etc.

The decode GAL–U9

The decode GAL (U9) uses the NWAIT, STATE and RESMAP outputs from the state machine with the addition of the following signals:

- NBW, the byte/word qualifier from ARM60
- A[1:0] the byte address bits from the CPU

These three additional signals, together with NRW, are sufficient for the byte lane decoding of write enable signals for the 32-bit system data bus. In addition the SCLK signal is also fed to a combinational logic input to the device to allow setup and hold timing for the write enables.

The decoder produces three registered chip enable outputs and four combinational byte-write signals:

- ROM, the EPROM and pipeline register enable signal
- RAM, the SRAM array common chip select
- NIO, the active low I/O space decode
- NWE[3:0] the four active-low byte write enable controls

The NWE outputs are provided as a little-endian implementation as supplied and the ordering must be reversed for big-endian system operation.

NWE0 is shared between the RAM and the I/O subsystems.

3.2.4 RAM Subsystem–U10 to U13

The PIE is supplied with four byte-wide 85ns static RAM devices providing 512KBytes memory. They share the word addresses A2 upwards — see the schematic diagram in Appendix C — with independently controlled write enable lines NWE0-3, which steer byte writes to the correct byte lane. The RAM bandwidth is 10 Mwords/second.

The static RAM devices are enabled by a common chip select signal, NRAM, and the output enables are controlled using the NIOS signal. This is in fact an I/O strobe timing signal during I/O cycle accesses and is reused as the active low RAM output

enable purely to avoid bus clashes when byte writes occur to RAM; using a common chip enable means that 32-bit write data from ARM is only written to one byte and the other bytes must not drive the bus in read mode.

A CMOS invertor buffer is used to drive NRAM to ensure the devices are in stand-by power-down mode when RAM is not selected.

Custom configuration

Four 512Kx8 static RAM devices can be soldered to the printed circuit board to provide 2MBytes RAM in place of the 512KBytes fitted as standard. A configuration link, JP2, is provided which should be moved and resoldered in the 4MB position.

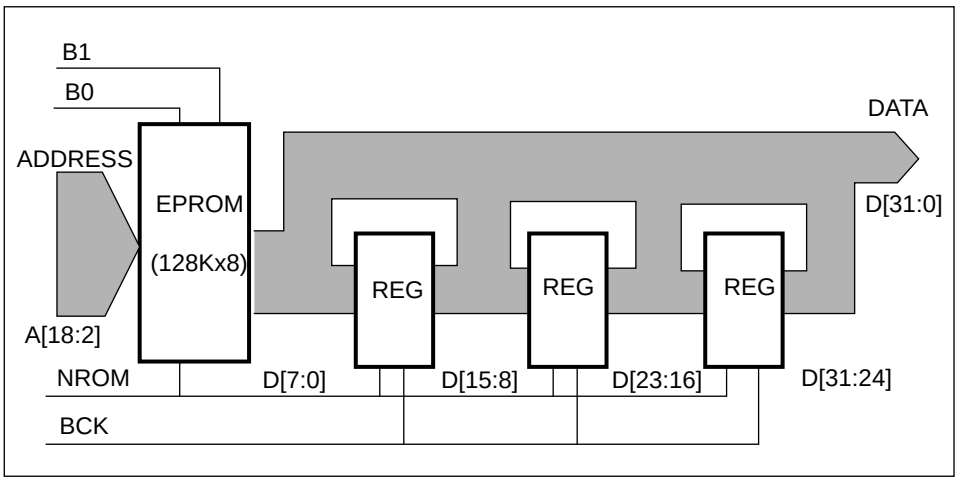
3.2.5 ROM subsystem– U8, U15, U16, U17

A socketed 128Kbyte EPROM (U8 on the circuit board) is supplied containing the bootstrap and self-test code as well as the low-level Remote Debug Protocol firmware. The ROM bandwidth is 1.5 Mwords/sec.

The ROM is an 8-bit device, and the following process, illustrated in Figure 3.6, ensures that the CPU receives the 32-bit words it requires. The processor is forced to wait whilst sequencing through four EPROM accesses building up to a full 32-bit quantity.

The bottom two address lines to the EPROM are provided from the state machine as B1, B0 byte sequence lines and the rest of the

Figure 3.6 **EPROM pipeline**



address lines are connected to the CPU word address bus (A2 upwards).

The EPROM sits directly on the bottom (D7-D0) data bits of the bus. A data pipeline comprising three eight-bit registers, U15 to U17, effectively acts as a byte-wide shift register which builds up the data bus value in reverse byte order. The sequence is byte3, byte2, byte1 and byte0.

The registers are clocked by the rising edge of BCK. This simplifies the control lines to NROM and BCK (the Byte Clock) with B1 and B0, the byte sequence bits.

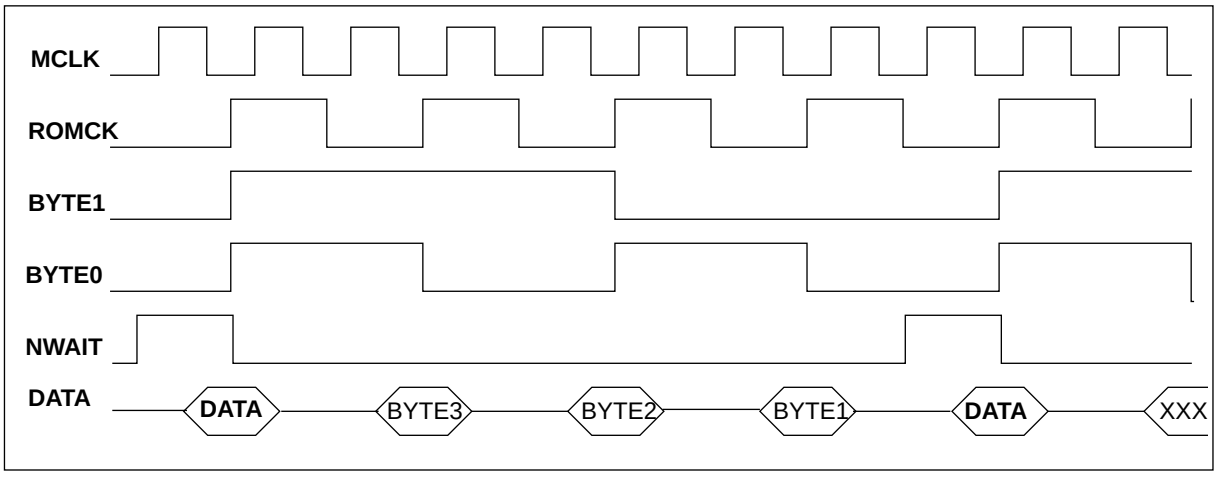
The EPROM is a slow access device, so three clock cycles per byte access are used. At 20MHz this allows a 120ns EPROM to work comfortably in the 150 ns access 'slot'. Figure 3.7 shows the timing.

A fast CMOS drive invertor is used to ensure the chip select signal to the EPROM input is driven to a true CMOS logic high level when the chip and pipeline are deselected. This ensures stand-by power operation.

Custom configuration

A pin-compatible 4Mbit EPROM may be fitted in place of the 128K x 8, 1Mbit device shipped. If a larger EPROM is fitted the configuration link (JP3) should be moved and resoldered to the 4Mbit setting.

Figure 3.7 **ROM Timing**



3.3 Logic analysis

The PIE Card is a development and evaluation platform and so provision for testing and analysis is built in the form of six connectors which allow direct connection to a logic analyser.

3.3.1 Connecting a Hewlett-Packard logic analyser

The evaluation card can be connected directly to Hewlett-Packard logic analysers supporting at least 80 channels, for example HP1650B and HP16510 series machines.

To avoid the use of "grabber" probes, use five HP Termination Adapters to interface between the analyser's 40-pin flexible cable connectors and the 20-pin header plugs (2 x 10 x 0.1 inch grid) on the evaluation card, as shown in Figure 3.8. These can be either:

- Termination Adapter HP Part No. 01650-63203
- Termination Adapter (earlier type) HP Part No. 06150-63201

The adapter connections are keyed to ensure correct orientation and allow correctly terminated high-speed signal analysis in a straightforward and reliable manner.

Other logic analyser probes with suitable header connectors will also work with the PIE.

The connector pods provide for:

- low address bus
- high address bus
- low data bus
- high data bus
- control & status port
- test & miscellaneous observation port

3.3.2 Connection procedure

First ensure both card and analyser are powered down. Then connect the analyser to the evaluation card as follows:

- Analyser POD1 to PL4 (low 16 data bits)
- Analyser POD2 to PL5 (high 16 data bits)
- Analyser POD3 to PL6 (status, clock signals)
- Analyser POD4 to PL2 (low 16 address bits)
- Analyser POD5 to PL3 (high 16 address bits)

Both systems may now be powered up and the analyser configured for use.

The analyser connectors are laid out on an overall board-wide 0.1 inch pitch grid allowing Eurocard prototype boards to be added for external user memory or I/O subsystem prototyping. 0.1 inch square grid board is recommended for this purpose. See Chapter Five for further details on expanding the PIE.

The control and status signals are grouped together on one port and allow for full processor analysis and test.

3.3.3 Logic analyser interface

The six header plugs, PL2 to PL7, are arranged along the top and bottom edges of the PIE card — see the schematic diagram in Appendix C — and signals are configured in bus order for correct bus value display.

In the following descriptions logic analyser inputs are shown in the left hand column and PIE outputs in the right.

Low Address Bus–PL2

This bus, in conjunction with the high address bus PL3, provides access to the full 32-bit address bus.

D[15]	:	A[15]
D[14]	:	A[14]
D[13]	:	A[13]
D[12]	:	A[12]
D[11]	:	A[11]
D[10]	:	A[10]
D[09]	:	A[09]
D[08]	:	A[08]
D[07]	:	A[07]
D[06]	:	A[06]
D[05]	:	A[05]
D[04]	:	A[04]
D[03]	:	A[03]
D[02]	:	A[02]
D[01]	:	A[01]
D[00]	:	A[00]
CLK	:	no connect (reserved)

High Address Bus–PL3 This bus, in conjunction with the low address bus PL2, provides access to the full 32-bit address bus.

D[15]	:	A[31]
D[14]	:	A[30]
D[13]	:	A[29]
D[12]	:	A[28]
D[11]	:	A[27]
D[10]	:	A[26]
D[09]	:	A[25]
D[08]	:	A[24]
D[07]	:	A[23]
D[06]	:	A[22]
D[05]	:	A[21]
D[04]	:	A[20]
D[03]	:	A[19]
D[02]	:	A[18]
D[01]	:	A[17]
D[00]	:	A[16]
CLK	:	no connect (reserved)

Low Data Bus–PL4 This bus, in conjunction with the high data bus PL5, provides access to the full 32-bit data bus.

D[15]	:	D[15]
D[14]	:	D[14]
D[13]	:	D[13]
D[12]	:	D[12]
D[11]	:	D[11]
D[10]	:	D[10]
D[09]	:	D[09]
D[08]	:	D[08]
D[07]	:	D[07]
D[06]	:	D[06]
D[05]	:	D[05]
D[04]	:	D[04]
D[03]	:	D[03]
D[02]	:	D[02]
D[01]	:	D[01]
D[00]	:	D[00]
CLK	:	no connect (reserved)

High Data Bus–PL5 This bus, in conjunction with the low data bus PL4, provides access to the full 32-bit data bus.

D[15]	:	D[31]
D[14]	:	D[30]
D[13]	:	D[29]
D[12]	:	D[28]
D[11]	:	D[27]
D[10]	:	D[26]
D[09]	:	D[25]
D[08]	:	D[24]
D[07]	:	D[23]
D[06]	:	D[22]
D[05]	:	D[21]
D[04]	:	D[20]
D[03]	:	D[19]
D[02]	:	D[18]
D[01]	:	D[17]
D[00]	:	D[16]
CLK	:	no connect (reserved)

Control/Status Port–PL6 This bus is the principal trigger and status port.

D[15]	:	DBE
D[14]	:	ABE
D[13]	:	ALE
D[12]	:	NTRANS
D[11]	:	NOPC
D[10]	:	LOCK
D[09]	:	NBW
D[08]	:	NRW
D[07]	:	SEQ
D[06]	:	NMREQ
D[05]	:	ABORT
D[04]	:	NIRQ
D[03]	:	NFIQ
D[02]	:	NRESET
D[01]	:	NWAIT
D[00]	:	STATE
CLK	:	Processor clock: MCLK(schematic LCLK)

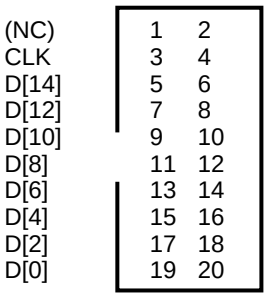
Test Port–PL7 This bus is provided for JTAG boundary scan access and testing with additional miscellaneous outputs for expansion.

D[15]	:	NWE3
D[14]	:	NWE2
D[13]	:	NWE1
D[12]	:	NWE0
D[11]	:	ERDY
D[10]	:	NIO
D[09]	:	NCPI
D[08]	:	ROMCK
D[07]	:	ROMB1
D[06]	:	ROMB0
D[05]	:	NROM
D[04]	:	TDO
D[03]	:	NTRST
D[02]	:	TDI
D[01]	:	TMS
D[00]	:	TCK
CLK	:	Processor JTAG test clock: TCK

3.4 Expansion interface

Expansion hardware may be added provided all signals are suitably buffered by installing a 0.1 inch grid prototype card over the logic analyser connectors. The connectivity (as seen from top of the PCB) is shown in Figure 3.8:

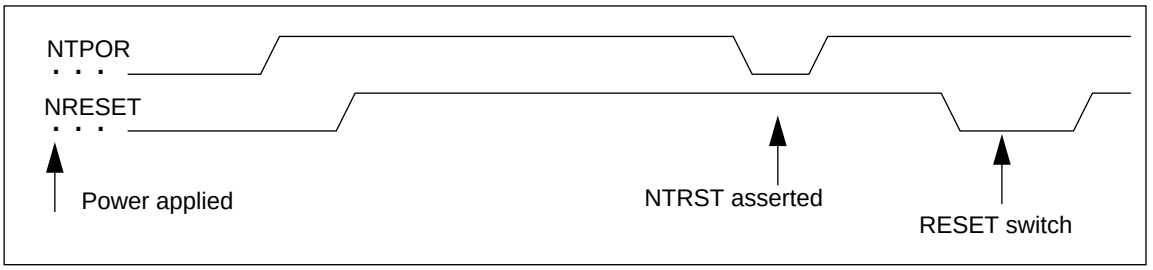
Figure 3.8 **Logic analyser port connectivity (PL2-7)**



where s[16:1] are the 16 logic analyser signals as defined above.

Pin 3, TRIG, is in fact CLK1 on later Hewlett Packard Analysers (HP16510 series) and pin 2 is hard-wired to provide power to an expansion card (a permanently high CLK2 on HP16510 analysers).

Figure 3.10 **Reset timing**



The unused trigger inputs on PL2-5 are reserved for future use, but may be used temporarily with flying leads to provide alternative triggers to the analyser (e.g. BCK, etc.).

3.5 The I/O subsystem

The I/O subsystem shown on Schematic 3 is described below in two main sections:

- Reset circuitry shared with the main system.
- Serial communications controller and host interface.

3.5.1 Reset circuitry

The reset circuitry is enhanced to allow independent resets to both the boundary scan and the processor.

The NTRST input to ARM60 must be driven low at initialisation time to disable the boundary scan circuitry and allow the ARM60 to function normally.

A separate NTPOR signal is generated which clears down the boundary scan at board power-up. This allows the boundary scan to be reset from the test connector (PL7) without requiring the CPU to be reset. This allows the possibility of stopping the processor (NWAIT held low) and then serially inspecting and testing the ARM60 in circuit.

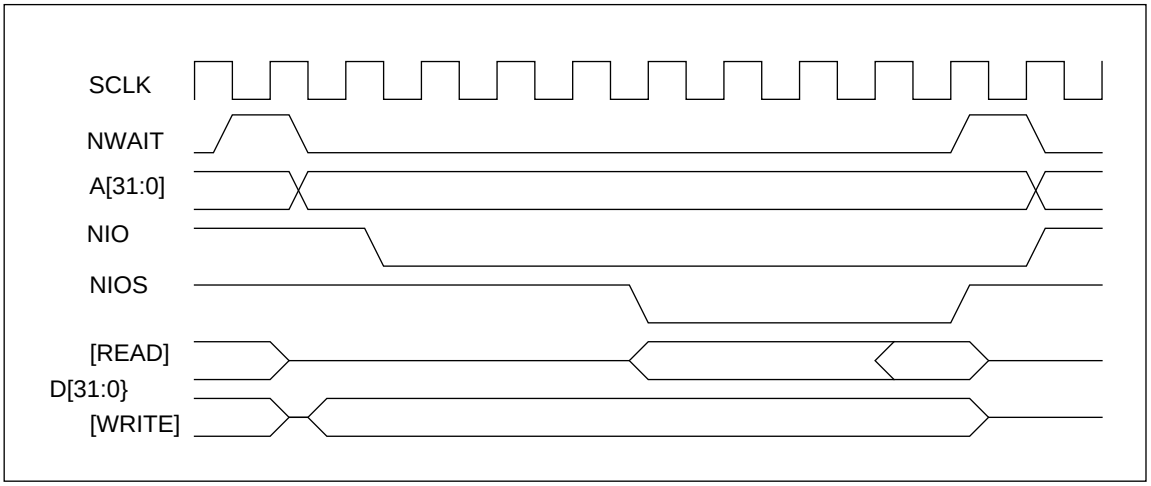
The main power-on-reset signal has a separate monostable function with a switch that allows the card to be reset. RES and NRES are required by the serial interface and the CPU system.

3.5.2 SCC2691 communication controller

A single-chip serial communications controller with internal timer counter and single-bit programmable output port is used as the host interface for asynchronous serial communications.

The chip used is a Philips/Signetics SCC2691, a single channel serial interface controller (SCC) capable of 38.4 Kbaud RS-232

Figure 3.11 **System bus interface timing**



operation. It has an active-low wired-OR Interrupt request interface which enables it to drive the inverted ARM Fast Interrupt ReQuest (NFIQ) input. Refer to the relevant datasheet from Philips.

The Counter/Timer built into the device allows basic timer services to the debugger (for timeouts etc.) with the only disadvantage being that the interrupt request line has to be shared with the serial controller (on FIQ).

Configuration

Link JP1 allows the SCC to be configured to generate IRQ as opposed to FIQ requests to the ARM60 (see the ARM60 datasheet for more information on ARM interrupts). Normally FIQ would always be used as the highest priority interrupt to allow the Remote Debugger Protocol to interrogate both user and operating system code.

System bus interface

The ARM60 is clocked at 20MHz and must use wait states to access the serial controller for I/O operations.

The state machine GAL generates timings (closely shared with the slow 8-bit EPROM interface pipeline) as shown in Figure 3.11. The interface may therefore be increased up to 40MHz (with fast static RAM) and still meet the valid I/O cycle timing constraints.

The read strobe (NRD) is removed in the cycle before the processor reads the data and so relies on dynamic bus hold time,

which is no problem on this card as all loads are CMOS with low leakage and no other devices are enabled during this cycle (and the next which is always a decode cycle in this design).

Peripherals which require address and chip select hold times after the data access should use a transparent latch to latch the data at the end of the read strobe.

3.5.3 LED indicator

The PIE includes a Light Emitting Diode, which is to be found next to the serial interface. It is driven by the Multi Purpose Output, MPO, and a high current inverting buffer. The LED defaults to off at reset.

The MPO pin is also driven by the RS232 driver to the output 9-way connector.

Software 4

4.1 About this chapter

This chapter describes the PIE card's software, and shows how it enables the card to be used with ARM software running on the host computer.

Topics include an overview and description of Demon, the Debug Monitor software in the PIE card's EPROM, and a listing of all the software interrupts (SWIs) implemented by Demon.

The PIE is designed to run software produced using the ARM Software Toolkit. The software provided on the PIE is intended to support the use of the ARM debugger ARMsd, to test and debug users' programs for ARM-based systems.

4.2 Demon and the PIE

Demon contains support code for the symbolic debugger ARMsd, and for user programs written in assembly or C. It supports host communication by handling interrupts. It supports user programs via a software interrupt mechanism. Demon also defines the default RAM memory map for the PIE.

The ARM Software Toolkit contains the source code for Demon.

The debug monitor is split into two parts:

- Level 0
- Level 1

The code in Level 1 is replaceable, allowing new targets to be debugged using the same standard interface. The Remote Debug Protocol (RDP) is implemented between the debugger and the PIE. Refer to the ARM Software Development Toolkit documentation for a complete description of the protocol.

The RDP defines the communication protocol between the debugger and debug monitor. Level 0 is responsible for recognising incoming RDP messages from the serial channel and dispatching them to Level 1 of the debug monitor. Level 0 also drives the serial channel, at the request of Level 1.

4.3 Level 0 services

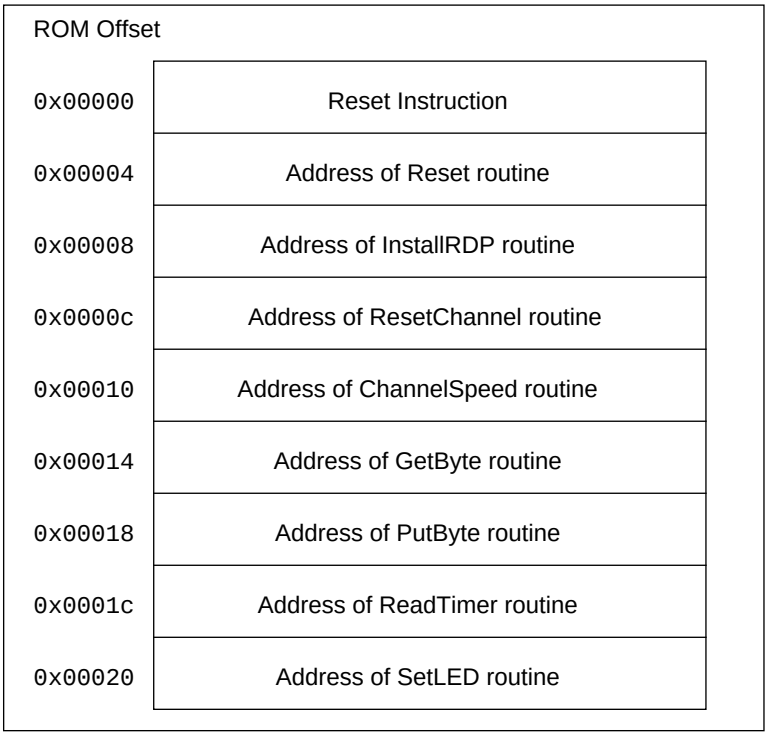
The interface between Level 0 and Level 1 is via a number of entry addresses situated at the beginning of ROM. Their layout is shown in figure 4.1 over.

The first two words are mapped into address 0 and 4 when the CPU is reset to provide the initial code entry.

The InstallRDP routine is used by the Level 1 code to register a handler of all RDP messages. The address of the handler should be passed in register 0, and the address of the previous handler will be returned in register 0. Level 1 code that does not handle all RDP messages may pass unhandled messages to the previous handler.

The RDP handler is entered in FIQ mode, with interrupts disabled, and the RDP message number in register 0. Exit from the handler by loading the program counter from the stack with an instruction like `LDMFD sp!, {pc}`. All register values have been saved before entry to the handler, and will be restored after it exits.

Figure 4.1 **Interface between Demon's Levels 0 and 1**



Once the Level 1 RDP handler has been entered it may read successive bytes from the serial channel using the GetByte routine, and send replies using the PutByte routine. The Level 1 handler may also formulate RDP messages to support application code that has called it, and these too are sent using PutByte.

The ResetChannel routine can be used to reset the serial driver, should an error be detected by the Level 1 Code.

The ChannelSpeed routine is used to change the speed of the serial channel in an implementation designed fashion. Currently the PIE board supports baud rates of 9600, 19200 and 38400 bits per second over its serial channel. These speeds can be selected by sending the values 1, 2 or 3 respectively to the ChannelSpeed routine. A value of 0 selects the default (power on reset) value, which for the PIE is 9600 bps. The meaning of all other values is undefined.

The GetByte and PutByte routines get and put bytes to and from the debug channel, as described above.

The ReadTimer routine returns in register 0 a centi-second count from an on board timer.

The SetLED routine allows the setting and clearing of the Light Emitting Diode (LED). Bit 0 of register 0 dictates the action required; 0 turns the LED off, and 1 turns it on.

4.4 Level 1 services

Level 1 handles RDP messages from the debugger, and generates RDP messages to support the actions of the application using it. The Level 1 code implements a number of SWI instructions to allow an application access to high level functions. See section 4.5, *Standard Monitor SWIs*.

4.4.1 Initial memory map

An initial memory map is defined on the following page in diagram 4.2.

The C library support uses the top of memory address (0x80000) for the base of the user mode stack. All stacks grow towards address zero.

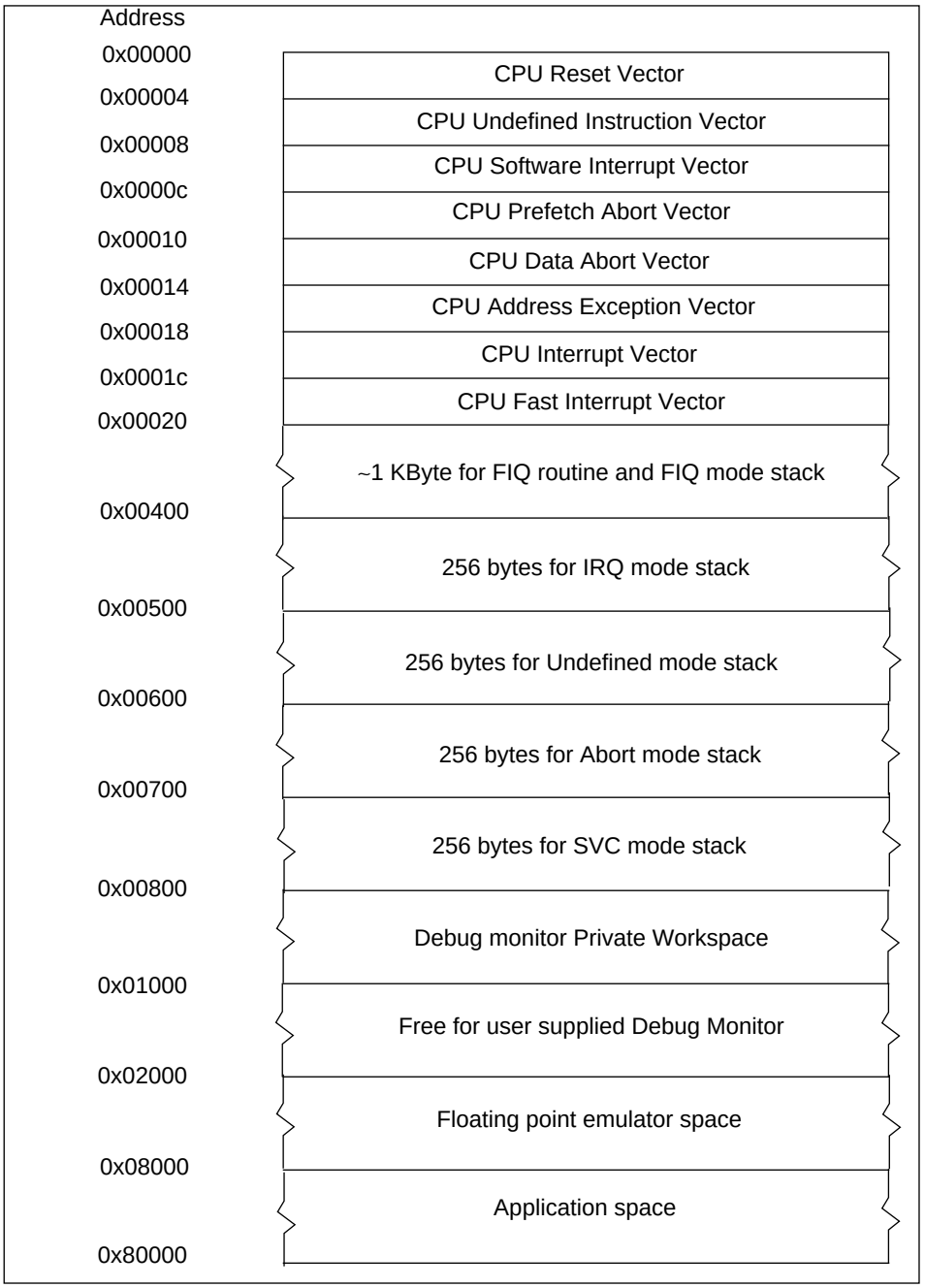


Figure 4.2 **Initial memory map**

4.5Standard monitor SWIs

Demon implements a number of useful SWIs which allow programmers to call host and operating system functions direct from programs being run and debugged on the PIE.

SWI_WriteC (SWI 0) Write a byte, passed in register 0, to the debugger. The character will appear on the display device connected to the ARM Symbolic Debugger, that is, on the screen of the host.

SWI_Write0 (SWI 2) Write the null terminated string, pointed to by register 0, to the debugger. The characters will appear on the display device connected to the ARM Symbolic Debugger, that is, on the screen of the host.

SWI_ReadC (SWI 4) Read a byte from the debugger, returning it in register 0. The read is notionally from the keyboard attached to the debugger.

SWI_CLI (SWI 5) Pass the string, pointed to by register 0, to the host's command line interpreter.

Please note that this SWI is not available in the PC/DOS release of the ARM Software Toolkit.

SWI_GetEnv (SWI 0x10) Return in register 0 the address of the command line string used to invoke the program, and in register 1 the highest available address in user memory.

SWI_Exit (SWI 0x11) Halt execution. Use this SWI to exit a program cleanly and return control to the debugger.

SWI_EnterOS (SWI 0x16) Put the processor into supervisor mode. If the processor is currently in a 26-bit mode then SVC26 is entered; otherwise SVC32 is entered.

SWI_GetErrno (SWI 0x60) Return, in register 0, the value of the C library *errno* variable associated with the host support for this debug monitor. *errno* may be set, by a number of C library support SWIs, including SWI_Remove, SWI_Rename, SWI_Open, SWI_Close, SWI_Read, SWI_Write, SWI_Seek, etc. Except where the ANSI C standard defines the behaviour, whether *errno* is set, and to what value it is set, are completely host-specific.

SWI_Clock (SWI 0x61) Return, in r0, the number of centi-seconds since the support code began execution. In general, only the difference between successive calls to SWI_Clock, can be meaningful.

SWI_Time (SWI 0x63) Return, in r0, the number of seconds since the beginning of 1970 (the Unix time origin).

SWI_Remove (SWI 0x64) Delete the file named by the NUL-terminated string addressed by r0. Return, in r0, zero if the removal succeeds, or a non-zero, host-specific error code if it fails.

SWI_Rename (SWI 0x65) Rename a file. R0 and r1 address NUL-terminated strings, the *old-name* and *new-name*, respectively. If the rename succeeds, zero is returned in r0; otherwise, a non-zero, host-specific error code is returned.

SWI_Open (SWI 0x66) Open a file. R0 addresses a NUL-terminated string containing a file or device name; r1 is a small integer specifying the file opening mode (see table below). If the open succeeds, a non-zero handle is returned in r0, which can be quoted to SWI_Close, SWI_Read, SWI_Write, SWI_Seek, SWI_Flen and SWI_IsTTY. Nothing else may be asserted about the value of the handle. If the open fails, the value 0 is returned in r0.

r1 value	ANSI C fopen() mode
0	“r”
1	“rb”
2	“r+”
3	“r+b”
4	“w”
5	“wb”
6	“w+”
7	“w+b”
8	“a”
9	“ab”
10	“a+”
11	“a+b”

SWI_Close (SWI 0x68) Close a file. On entry, r0 must be a handle for an open file, previously returned by SWI_Open. If the close succeeds, zero is returned in r0; otherwise, a non-zero value is returned.

SWI_Write (SWI 0x69) Write bytes to a file. On entry, r0 must contain a handle for a previously opened file; r1 points to a buffer in the callee; and r2 contains the number of bytes to be written from the buffer to the file. SWI_Write returns, in r0, the number of bytes not written (and so indicates success with a 0 return value).

SWI_Read (SWI 0x6a) Read bytes from a file. On entry, r0 must contain a handle for a previously opened file or device; r1 points to a buffer in the callee, and r2 contains the number of bytes to be read from the file into the buffer. SWI_Read returns, in r0, the number of bytes not read and so indicates the success of a read from a file with a zero return value. If the handle is for an interactive device (SWI_IsTTY returns non-zero for this handle), then a non-zero return from SWI_Read indicates that the line read did not fill the buffer.

SWI_Seek (SWI 0x6b) Alter file pointer position of a file. On entry, r0 must contain a handle for an open seekable file object and r1 the absolute byte position to be sought to. If the request can be honoured then SWI_Seek returns 0 in r0; otherwise a host-specific non-zero value. Note that the effect of seeking outside of the current extent of the file object is undefined.

SWI_Flen (SWI 0x6c) Return length of a file. On entry, r0 contains a handle for a previously opened, seekable file object. SWI_Flen returns, in r0, the current length of the file object, otherwise -1.

SWI_IsTTY (SWI 0x6e) On entry, r0 must contain a handle for a previously opened file or device object. On exit, r0 contains 1 if the handle identifies an interactive device, and 0 otherwise.

SWI_TmpNam (SWI 0x6f) Return a temporary file name. On entry, r0 points to a buffer and r1 contains the length of the buffer (r1 should be at least the value of L_tmpnam on the host system). On successful return, r0 points to the buffer which contains a host temporary file name.

If the request cannot be satisfied (e.g. because the buffer is too small) then 0 is returned in r0.

SWI_InstallHandler (SWI 0x70) SWI_InstallHandler installs a handler for a hardware exception.

On entry, r0 contains the exception number (see the table below); r1 contains a value to pass to the handler when it is eventually invoked; and r2 contains the address of the handler.

On return, r2 contains the address of the previous handler and r1 its argument.

On occurrence of the exception, the handler is entered in the appropriate non-user mode, with r10 holding a value dependent on the exception type, and r11 holding the handler argument (as passed to InstallHandler in r1). r10, r11, r12 and r14 (for the

processor mode in which the handler is entered) are saved on the stack of the mode in which the handler is entered; all other registers are as at the time of the exception. Any effects of the instruction causing the exception have been unwound. If the handler returns, the exception will be passed to the debugger.

NO	EXCEPTION	MODE	R10 VALUE
0	branch through zero	swi32	-
1	undefined instruction	undef32	-
2	swi	swi32	swi number
3	prefetch abort	abort32	-
4	data abort	abort32	-
5	address exception	swi32	-
6	IRQ	irq32	-
7	FIQ	fiq32	-
8	Error	swi32	error pointer

SWI_GenerateError (SWI 0x71)

On entry, r0 points to an error block (containing a 32-bit error number, followed by a zero-terminated error string) and r1 points to a 17-word block containing the values of the ARM CPU registers at the instant the error occurred (the 17th word contains the PSR).

SWI_GenerateError causes the (software) error vector to be called: this SWI also causes the installed error handler to be called (see SWI InstallHandler).

Developing your system

5

5.1 About this chapter

This chapter looks at aspects of the PIE system of interest to system designers who wish to use the PIE as the basis for their own system or ASIC designs. It should be read after users have become familiar with the standard PIE hardware and gained a good understanding of ARM processors, and after the introduction to the PIE hardware in Chapter Three.

The PIE is intended to ease the process of building systems and ASICs based on ARM technology. With the addition of your own circuits the PIE can be used to test designs for many ideas and products. Suitable uses include embedded systems such as communications interfaces, memory subsystems, network interfaces for FDDI or Ethernet, laser printer engines, and custom ICs for a variety of purposes.

The first section provides detail on timing and signals which relate to the expansion bus. System designers who intend to add expansion hardware to the PIE should pay particular attention to the information in this section.

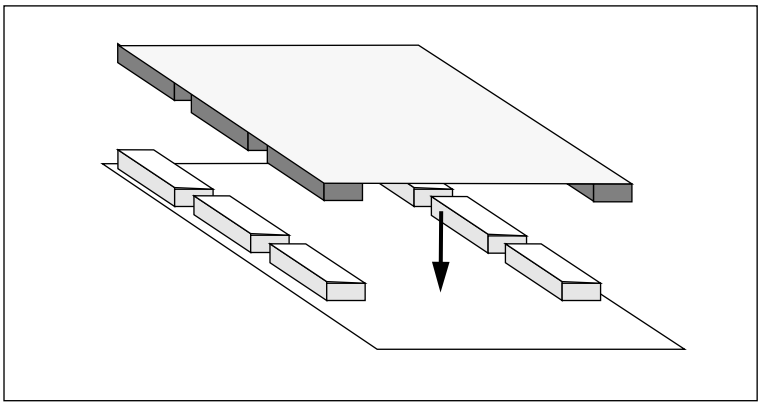
The second section covers state machine design. The PIE state machine and ways to develop it are discussed in detail.

The third section discusses serial interface programming in detail and is intended to allows PIE users to ensure that even heavily modified test systems are still able to communicate with the host.

5.2 Extending the PIE

The ARM60 evaluation card is designed to interface to a standard prototyping card, typically a 100 mm x 160 mm Eurocard which can then sit over the PIE card and be connected directly to it using standard 0.1inch PCB double row sockets, as shown in Figure 5.1 Other size cards can be used if required.

Figure 5.1 **Adding an expansion card**



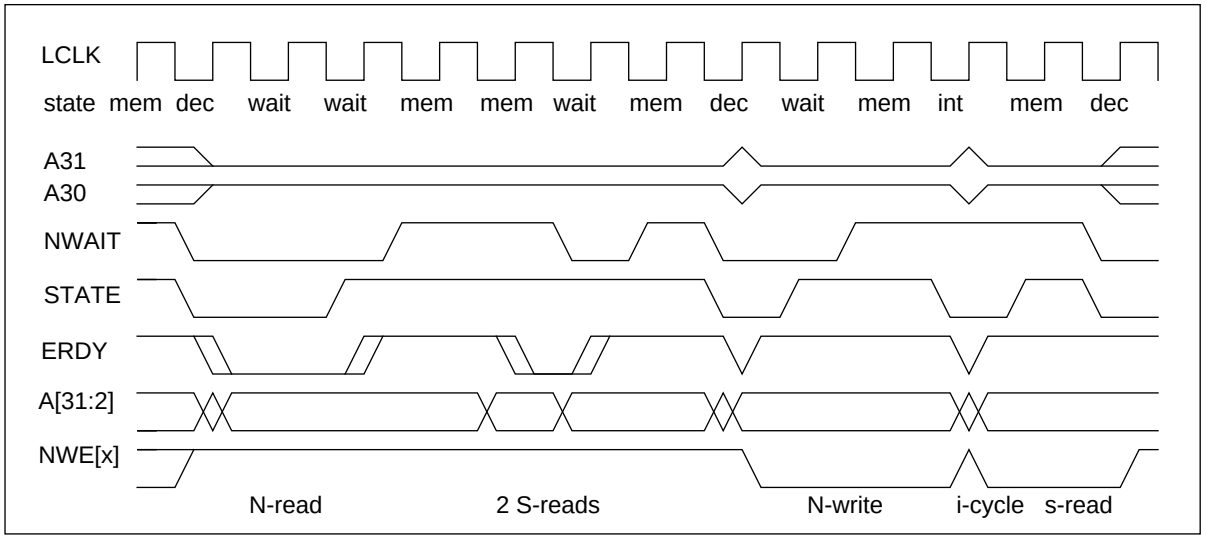
Observe the following points when building any hardware test systems to add to the PIE:

- Do not load signals excessively (buffer for more than four CMOS loads).
- Avoid ribbon cable connections over 100 mm long. Use plug/socket connections from top card if possible.
- Decode memory space to the 1-2 Gbyte address range to avoid clashing with devices (RAM 0-1 Gbyte, I/O 2-3 Gbyte, ROM 3-4 Gbyte–see Figure 3.2).
- Use the external ready (ERDY) signal with care. The system card will wait indefinitely if the signal is driven low during external I/O cycles.
- Check carefully all connections, especially for any short circuits, before attempting to use any add-on card.
- Disconnect or turn off the power supply from the card before installing or removing any external test circuitry.

5.2.1 Expansion bus interface

The physical aspects of the expansion bus interface are discussed in Chapter Three (page 27). This section covers the signal and timing information needed to enable users to interface additional hardware to the PIE successfully.

Figure 5.2 **External timing cycles**



The external interface is decoded for the memory space from 1-2 Gbyte. There is a single External Ready (ERDY) input on connector PL7. It has a weak pull-up resistor on the card which is used to provide a variable wait-state interface (the default is ready).

ERDY must be set high to allow the cycle to complete on the next cycle. If a peripheral (memory, etc.) cannot complete in one more cycle then ERDY must be driven low to add wait states until ERDY is driven high to signify data transfer is completing in the next clock period.

ERDY must be set-up and held around falling edges of LCLK (a slightly delayed version of the SCLK signal one level back in the clock tree invertor chain).

Signal timing for a mixed variety of cycles is shown in figure 5.2.

The first non-sequential read cycle (N-read) is stretched by two wait states the next cycle (sequential) completes immediately followed by another with a single wait state.

The rest of the cycles are shown unstretched to demonstrate the inclusion of an extra wait cycle every write cycle (which allows the

time for data out to become valid from the processor - which in sequential writes to devices such as dynamic RAM is a limiting factor).

After the write cycle an internal cycle is shown which is merged immediately with the following sequential memory request as ERDY is till high. Write data is set up in such internal cycles which can complete with no wait cycles.

5.2.2 Expansion bus timing

Figure 5.3 shows the cycle times that are generated, in units of clock period (50ns for 20 MHZ clock).

5.3 State machine design

The basic state machine design described in Chapter Three is the starting point for the overall state machine design which is discussed more fully in this section.

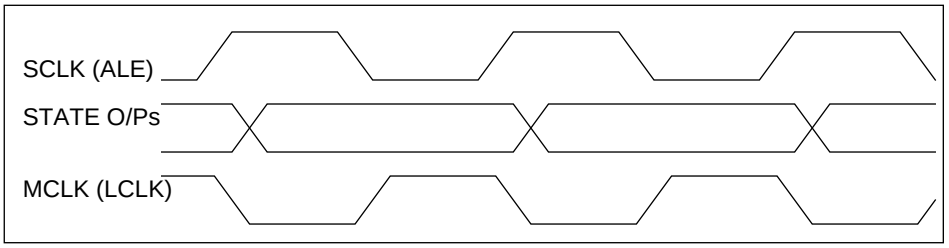
5.3.1 Processor clock interface

The ARM60 has a clock input MCLK and a gating control NWAIT which enables the clock to the processor core.The logic function is an AND-gating function of the two inputs: only when NWAIT and MCLK are both high is an active processor clock (“Phase 2” in ARM2 nomenclature) generated.

Figure 5.3 **Cycle times generated**

Cycle Type	Precharge	Access	
Non-sequential SRAM access	1	2	50+100ns-Tdsu
Sequential SRAM read	0	2	100ns-Tdsu acc.
Sequential SRAM write	1	2	
I/O access (8-bit)	1	12	650ns min cycle
(I/O read/write strobe)	(7)	(6)	300ns-Tdsu acc.
EPROM access (32-bit)	1	12	650ns cycle
(each EPROM byte access)		(3)	150ns-Tdsu acc.
Internal cycle	1	0	50ns
Non-sequential EXT access	1	1*	(1+N)x50ns
Sequential EXT read	0	1*	Nx50ns
Sequential EXT write	0	1+1*	(1+N)x50ns
* Where ERDY may hold up completion by an integer number of cycles			

Figure 5.4 **Waveforms**



The clocking scheme used in the PIE is designed to use standard (active rising-edge triggered) programmed logic components with a clock scheme as follows:

SCLK the system state machine clock

MCLK the processor clock signal (inverted form of above)

A clocked Address Latch Enable, ALE, signal is also used in this design. The ARM60 processor internal addresses are generated ("early") timed from the active MCLK rising edge (active means only when NWAIT was high). This is useful when early addresses allow memory mapping or dynamic RAM interfacing, but is not required for simple memory and I/O bus interfaces where addresses need to be held to the end of the access cycle.

When ALE is high allows the processor addresses to flow out of the CPU and when low holds them at their last value. SCLK is used as ALE, and the Address Latch open time guarantees the addresses start changing to a new state cleanly at the completion of the active MCLK cycle.

5.3.2 State interface timing

The diagram (Figure 5.5) shows the basic parameters for the state machine interface to the ARM60 processor. Note the following times:

Tq the SCLK rising to Q output change of the PLDs.

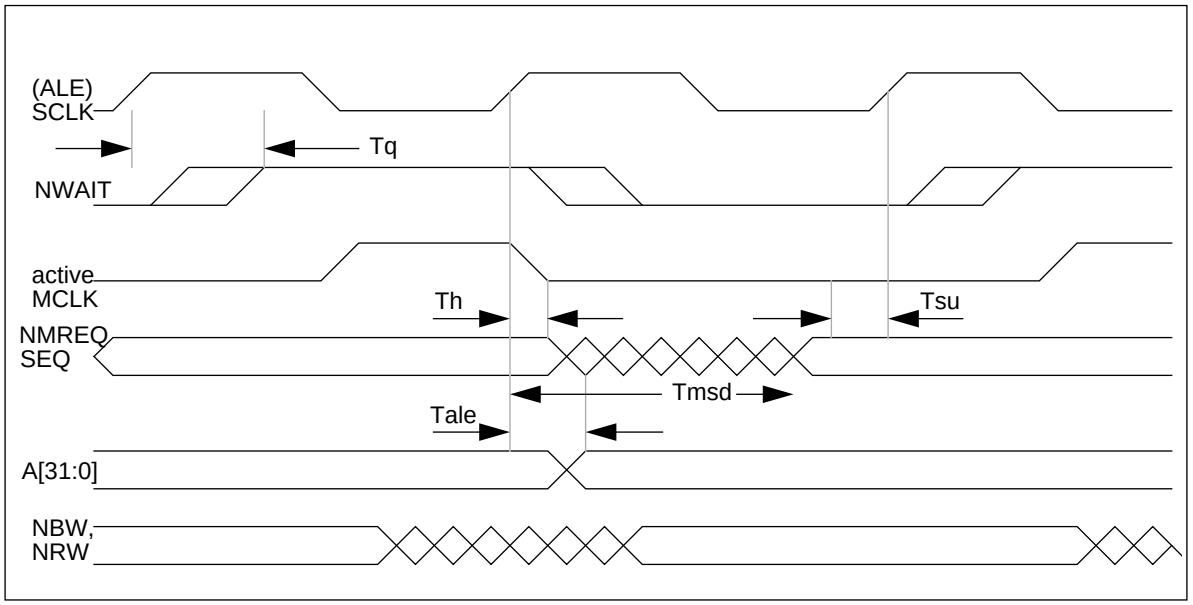
Th input hold time to SCLK rising.

Tsu input setup time to SCLK rising.

Tms the ARM processor (active) MCLK falling to new NMREQ, SEQ.

Tale the ALE latch fall-through time.

Figure 5.5 **Annotated timing diagram**



In zero wait-state operation and internal cycle operation at MCLK rate new NMREQ and SEQ status is generated every clock cycle: the state machine is constrained by valid NMREQ and SEQ time signals and the combinational logic delays inherent in the setup to the next SCLK edge. Also the SCLK output timing must meet valid NWAIT timing constraints.

5.3.3 State diagram

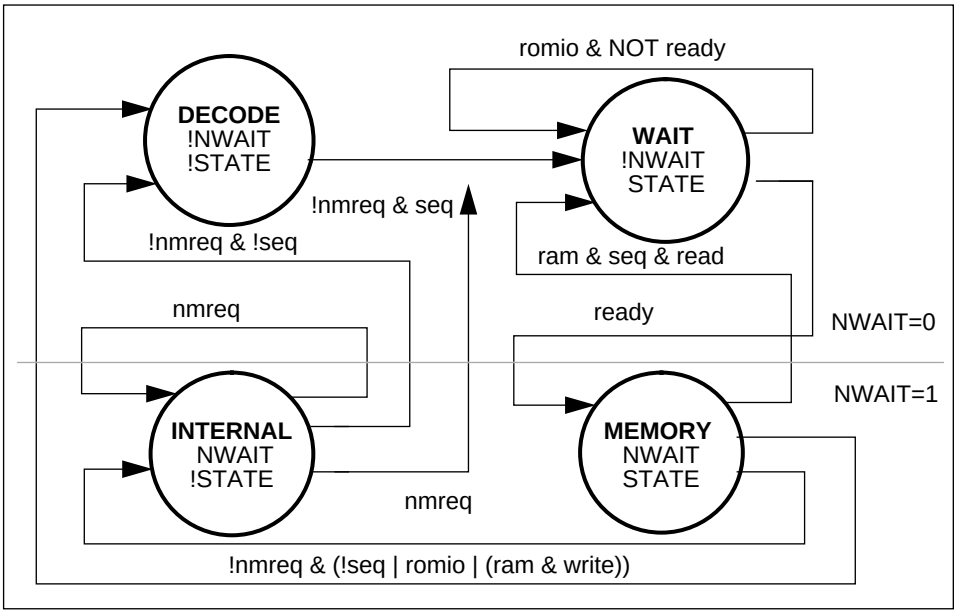
The state diagram outlined in the introduction is further expanded and explained in Figure 5.6 as follows:

- States are shown as named circles, with asserted output listed below the names.
- Input transitions are from left, output transitions to right.

Internal cycles are optimised such that a sequential memory request following an internal cycle is merged with the decode because the processor guarantees to have broadcast a valid address during the last internal operation.

The diagram splits into two halves; the upper half represents cycles where the CPU is stalled (NWAIT asserted low) and the lower half the active processor cycle operations.

Figure 5.6 **Expanded state diagram**



5.3.4 EPROM and I/O access

The principal enhancement to the simple state machine depicted in the basic flow diagram is the merging of internal cycles with following sequential cycles to avoid a wasted decode cycle, shown in Figure 5.7. The address on the bus is guaranteed to be valid during such an internal cycle and therefore the decode is implicitly performed. Note the outputs decremented in wait state.

The Byte counter BYT counts down from 3 to 0 to fetch the bytes from EPROM (preloaded when leaving internal or decode states) and only decremented each time the CNT counter cycles through zero.

The wait counter cycles through three states, preset to 2 as BYT when inactive and reloaded whilst sequencing through the bytes.

The resultant external waveforms produced are shown in Figure 5.8. Note that the ROM must be programmed little-endian byte ordered and is accessed in reverse byte order for data pipeline design simplicity.

The I/O decoder does not use the byte count (address) bits and simply uses the down counter to generate strobe timing.

Fig 5.7 **EPROM and I/O access state diagram**

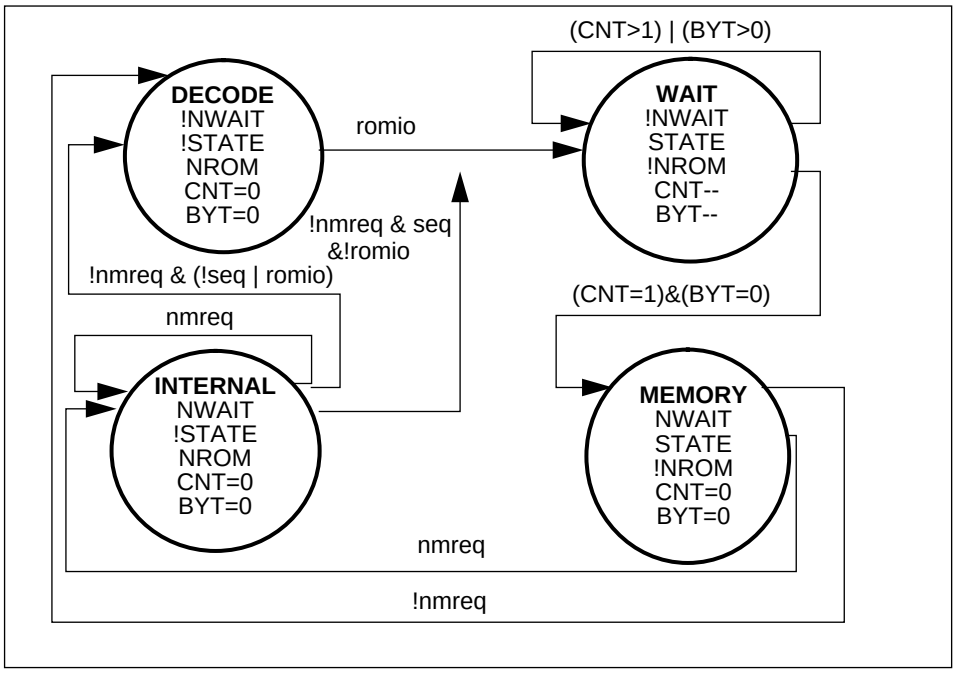
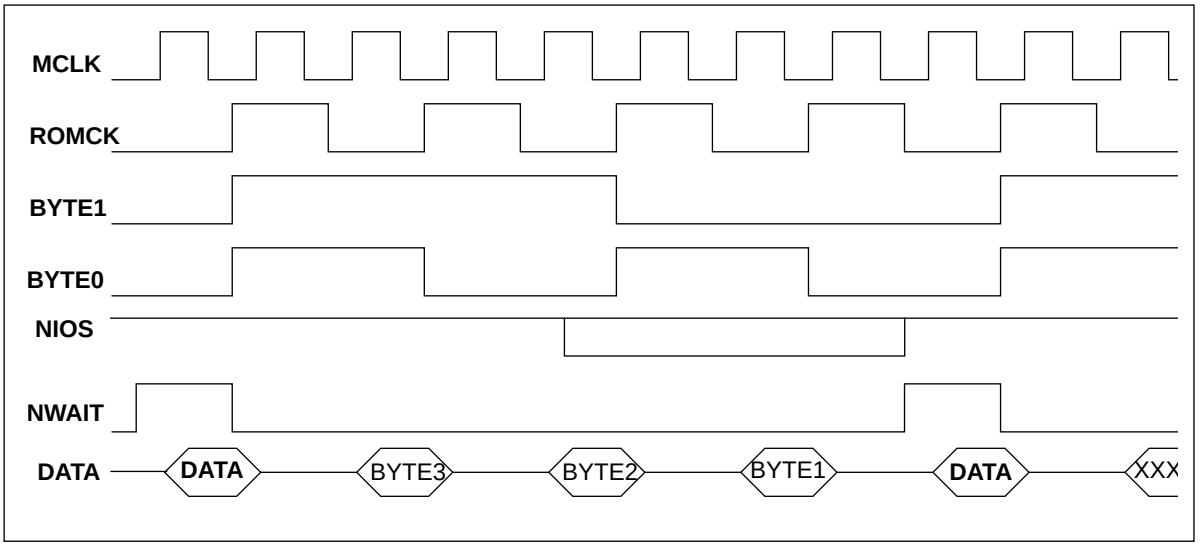


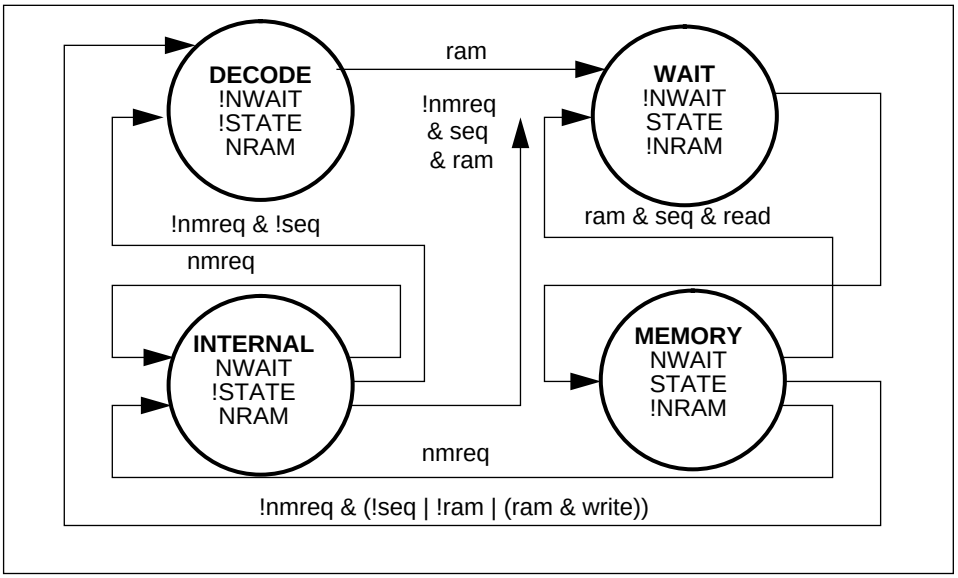
Figure 5.8 **ROM timing waveforms**



5.3.5 SRAM access

The principal enhancement for static RAM access is to merge sequential RAM read accesses to avoid re-entering the decode state and simply loop back through the wait state as the low order address increments, as shown in Figure 5.9. There is an implicit assumption here that sequential runs will not be re-decoded across the RAM/EXT memory spaces.

Figure 5. 9 **SRAM access state diagram**

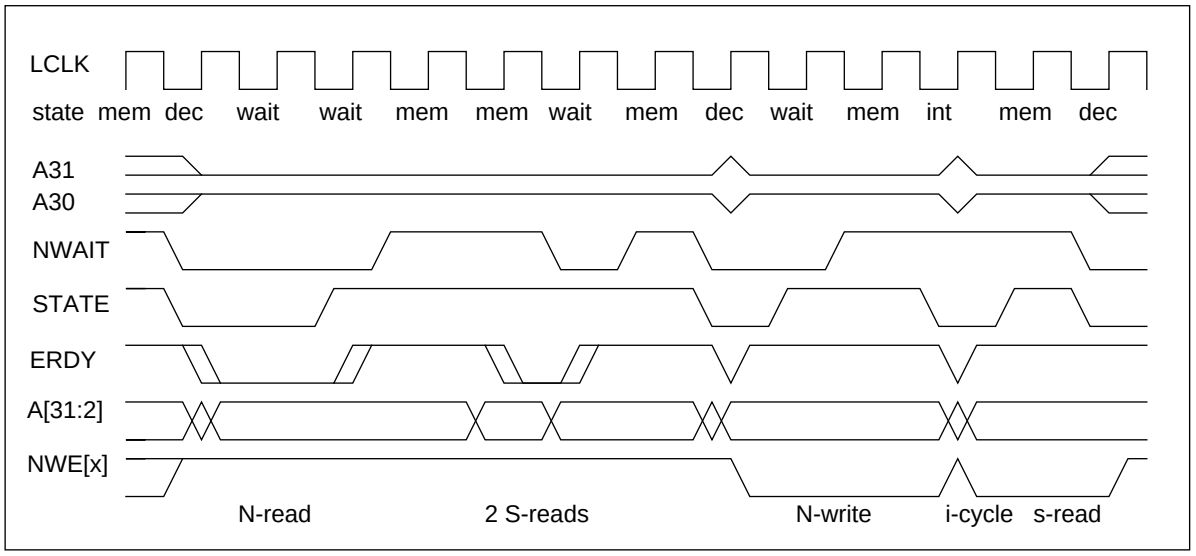


Write cycles are implemented with an extra decode cycle. This causes the NRAM chip select to strobe in write data at the rising edge. Speed could be improved by gating the write strobe and removing this extra decode tick.

Figure 5.10 shows the waveforms produced by a mix of read and write burst accesses and the merging of an internal cycle with the following sequential cycle at the end.

NRAM is asserted low for burst reads to improve static RAM access speed. This is not strictly necessary for the RAM fitted to the PIE but demonstrates a technique for using fast BiCMOS RAM with zero wait states or on-chip RAM in an ASIC design. The situation is different for writes to static RAM: NRAM must be used rather than using address lines to ensure valid writes (which in zero-wait state designs require a wait state to achieve stable data out from the ARM CPU).

Figure 5.10 **SRAM Waveforms**



In fast write cycle designs the data out time from ARM60 must be carefully observed particularly if dynamic RAM is being used (where delayed write cycles would normally be used more efficiently than early writes where data is required set up to NCAS falling edge).

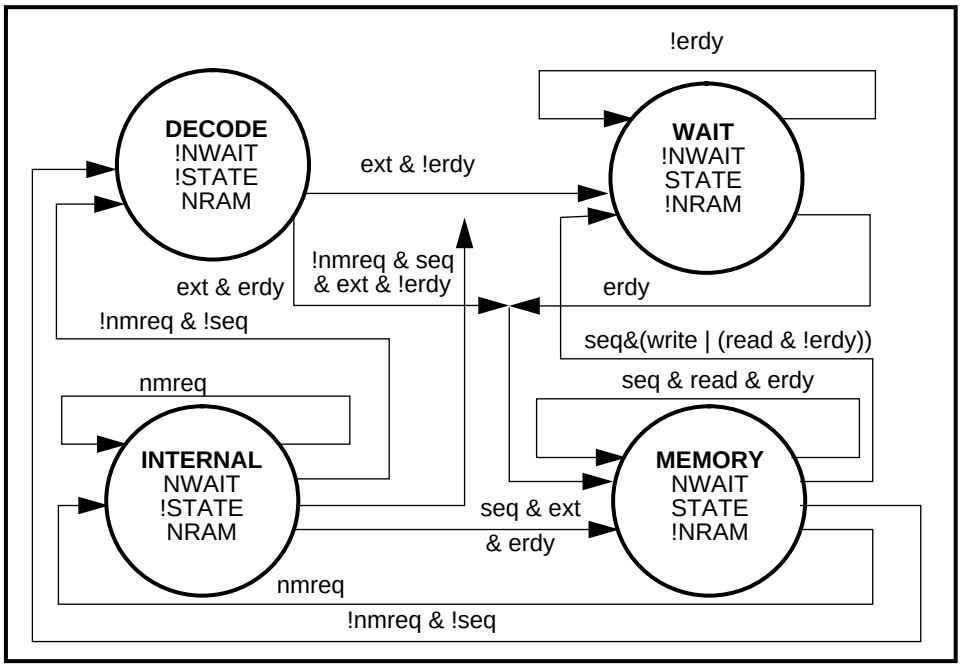
5.3.6 External I/O access

An External Ready (ERDY) signal is provided in this example of a variable wait state interface, shown in Figure 5.11. If ERDY is asserted high then the bus cycle will complete in the next clock period.

This enhancement allows zero wait state memory operation for sequential external memory read operations, with a single cycle precharge cycle added for sequential write and non-sequential accesses. Thus 40ns static RAM may be mapped into external RAM space with the ERDY signal asserted high.

Fast page-mode dynamic RAM would run at page cycle rate and only add wait states by deasserting ERDY for row access and refresh cycle operations.

Figure 5.11 **External I/O access state diagram**



5.3.7 State implementation

Two GAL20V8 electrically erasable programmable generic array logic components are used to implement a state machine to meet the specification outlined above. The states have been carefully designed to keep the number of product terms within limits and to attempt to make the design simple to alter and enhance. See Appendix B for the equation listings.

5.4 5.5 Serial interface programming

The card supports the ARM Remote Debugger Protocol (RDP) which can be run on any of Sun, PC compatible and Apple Macintosh host computers. The PIE's asynchronous serial interface allows the card to be interfaced to any of these hosts.

This avoids special hardware interfacing or system software driver support overheads.

5.5.1 Programming interface

The I/O base address is

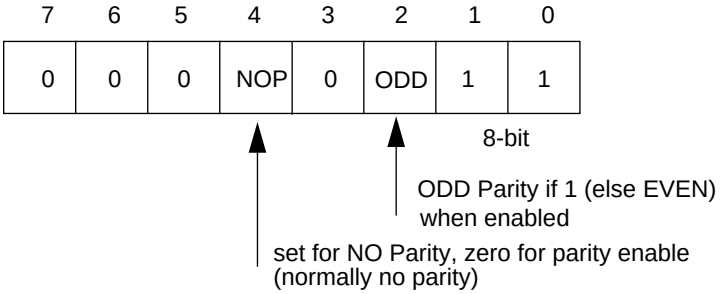
```
0x80000000
```

Only data bits D[7:0] are valid on the bus after reads from I/O space, the other 24-bits contain previously-driven bus values. Byte load and stores would normally only be used rather than word operations.

Big-endian	Little-endian	READ	WRITE
0x80000003	0x80000000	MR1, MR2	MR1, MR2
0x80000007	0x80000004	SR	CSR
0x8000000B	0x80000008	*RESERVED*	CR
0x8000000F	0x8000000C	RHR	THR
0x80000013	0x80000010	*RESERVED*	ACR
0x80000017	0x80000014	ISR	IMR
0x8000001B	0x80000018	CTU	CTUR
0x8000001F	0x8000001C	CTL	CTLR

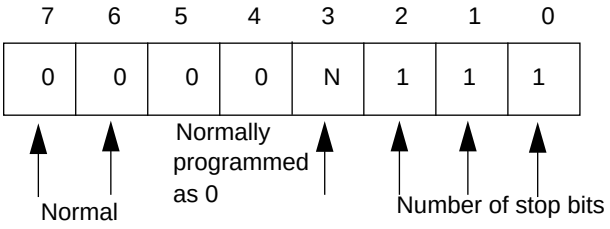
Mode register 1–MR1

The first write to this register maps MR2 in for further accesses; it is recommended that no parity is enabled.



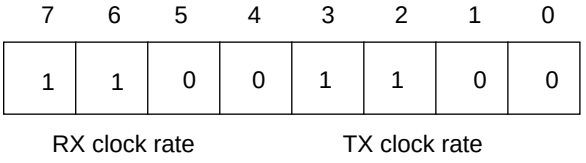
Mode Register 2–MR2

Bit 3, shown as N, is set to 1 for two stop bits and set to zero for one stop bit. It is recommended that one stop bit is used.



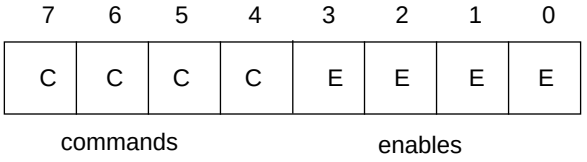
Clock Select Register–CSR

The clock select register sets the speeds for the transmission and reception of serial communications. The bottom four bits define transmission and the top four reception.



Command Register–CR

See the Philips Data Book for full information on this register.

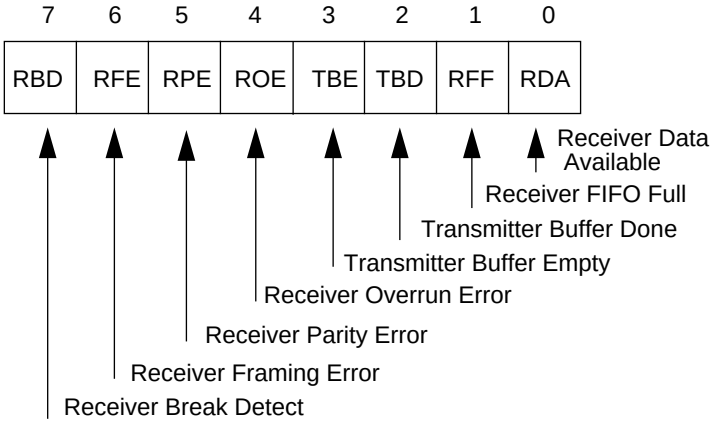


Writes to the command register require at least three rising edges (four clock cycles would guarantee this) e.g. four 3.6864MHz

cycles or 1.1 microseconds between consecutive writes. The EPROM byte-wide code has no problem with this but fast RAM code must ensure commands are not written at a faster rate.

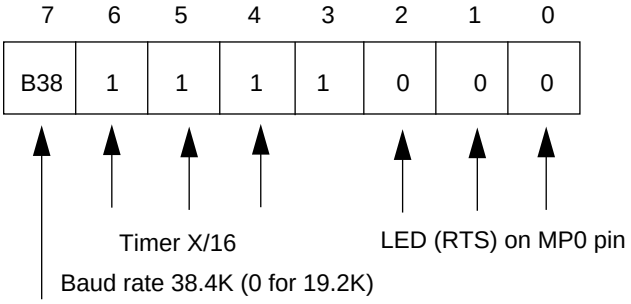
Status Register–SR

The status register is used to convey errors and other important information which requires action back to the controlling program. The errors are expressed in the eight bits of the registers as follows:



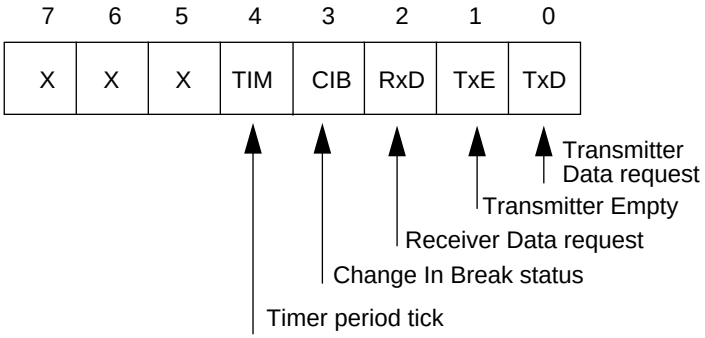
Auxiliary Control Register–ACR

The auxiliary control register contains the following information:



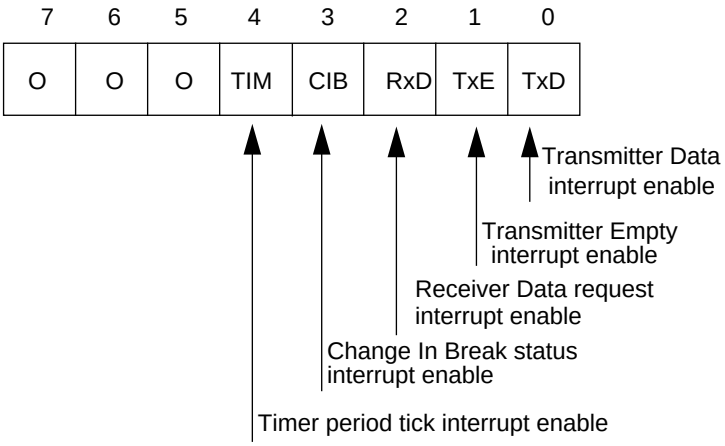
Interrupt Status Register–ISR

This register conveys information about interrupts:



Interrupt Mask Register–IMR

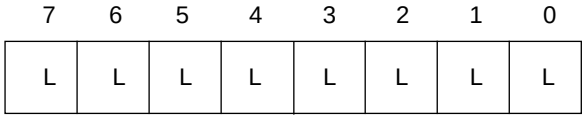
This register contains the interrupt masks:



Counter/Timer Register Low–CTL

With a 3.6864MHz crystal and the divide by 16 prescalar selected the counter/timer register deals in units of 4.34 µS (exactly 230.4 KHz count rate).

The interrupt rate is this divided by 2N where N is programmed as CTH and CTL register values. The minimum frequency is 230.4 KHz (2*65535) = 1.76 Hz.



You should be aware that reading this register and CTH may result in discontinuities as increments may occur between reads.

Counter/Timer Register High –CTH

With this register you should also be aware of possible discontinuities caused by increments between reads; see CTL on the previous page.

7	6	5	4	3	2	1	0
H	H	H	H	H	H	H	H

Appendix A

A.1 Host interface

To use the PIE you need to connect it via a cable to your host system. A nine-pin sub-miniature D-type socket is provided on the PIE for this purpose; its signal levels are RS232 compatible. You will need to build a cable, ensuring that the signals are properly carried to the host and to the correct pins on your host system's serial port. This appendix should be used in conjunction with the manual for your host system.

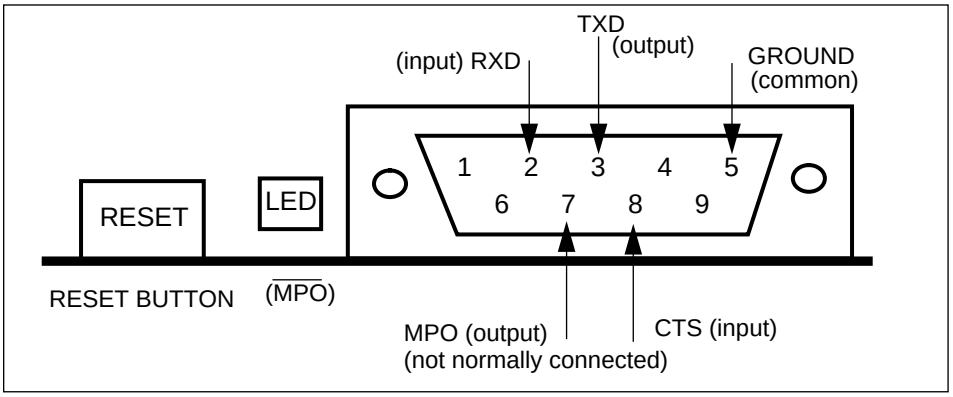
A.2 Connector

These signals appear at the connector at the edge of the card:

PIN	FUNCTION
2	Receive Data to card
3	Transmit Data from card
5	Signal Ground
(7)	(Multi Purpose Output)
8	Clear To Send to card

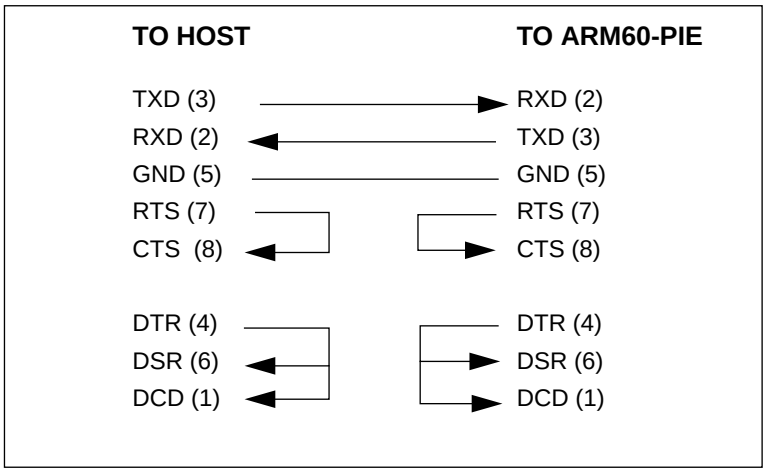
The Multi Purpose Output does have an RS232 output driver but is not normally used in serial communications handshaking unless specifically programmed to do so. It reflects the (inverse) status displayed on the LED.

Figure A.1 **Connector (View from end of card)**



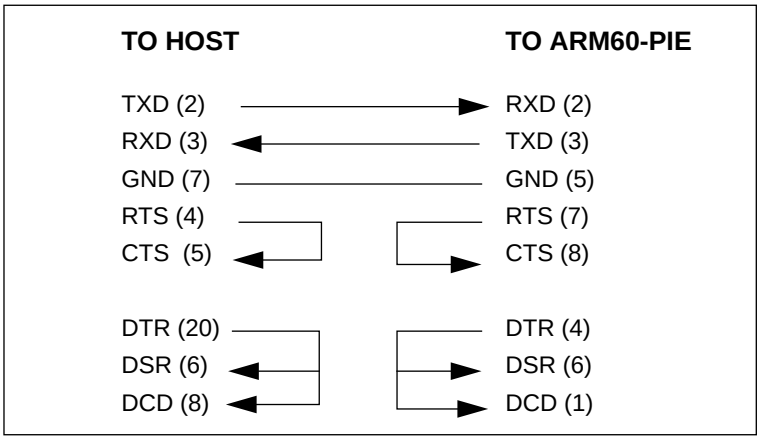
9-pin PC/AT host serial (COMx:) cable

A standard null modem cable is used to connect host and evaluation card. The cable should be a symmetric cable such as would be used to connect PC/AT machines back-to-back with 9-way sockets at either end:



25-way SUN host serial (A/B) cable

If your host is a Sun you should produce a cable wired like this:



Appendix: GAL listings

B

B.1 State Control GAL

```
Name            PIESM2 ARM60-PIE State Machine;
Partno          PIESM2;
Revision        03;
Date            17/01/92;
Designer        David Flynn;
Company         Advanced RISC Machines Ltd;
Assembly        ;
Location        U5;
Device          g20v8;

/*****************************************************/
/*   Allowable Target Device Type:        GAL20V8A                    */
/*****************************************************/

/**   Inputs   **/

pin[1]          = ck;
pin[2]          = a31;
pin[3]          = a30;
pin[4]          = abort;
pin[5]          = Nread;
pin[6]          = Nmreq;
pin[7]          = seq;
pin[10]         = Nreset;
pin[11]         = erdy;
pin[13]         = Noe;
pin[14]         = Ntrans;

/**   Outputs   **/

pin[22]         = !ios;   /* io strobe gating */
pin[21]         = q0;     /* divide/3 count */
pin[20]         = q1;     /* romck */
pin[19]         = q2;     /* romb0 */
pin[18]         = q3;     /* romb1 */
pin[17]         = state;  /* -> PIEDC1 */
pin[16]         = Nwait;  /* -> PIEDC1 */
pin[15]         = resmap; /* -> PIEDC1 */

/**   definitions   **/
$define decST   (!Nwait & !state)
$define waiST   (!Nwait &  state)
$define memST   ( Nwait &  state)
$define intST   ( Nwait & !state)

$define ramDEC  (!a31 & !a30)
$define extDEC  (!a31 &  a30 & !resmap)
$define ioDEC   ( a31 & !a30)
$define romDEC  ( a31 &  a30)
```

```
$define hiDEC   ( a31)

$define iCYC    ( Nmreq)
$define sCYC    (!Nmreq &  seq)
$define nCYC    (!Nmreq & !seq)
$define mCYC    (!Nmreq)

/**   Logic Equations   **/

/*   in fact resmap := reset + resmap & !(ramDEC & write & memST) */
resmap.D =  !Nreset
         #  resÒmap & !Nread
         #  resmap & !Nwait
         #  resmap & !state
         #  resmap &  a31
         #  resmap &  a30
         ;

/* carefully handcrafted down counter:
   low 2 bits count:    2,1,0,2,1,0...
   rom,io access count: E,D,C,A,9,8,6,5,4,2,1,0(,0)
   ram                  1,0(,1,0 for seq RAM read)

ROM/IO
      MEMDECWAIWAIWAIWAIWAIWAIWAIWAIWAIWAIMEM
            _________________
  q3  _____/                 \____________________
            _________          _________
  q2  _____/         \________/         \__________
         __       __       __       __
  q1  __/  \_____/  \_____/  \_____/  \_____/  \___
         __       __       __       __
  q0  __/  \_____/  \_____/  \_____/  \_____/  \____
      ____________________ _________________ _____
NIOS                      \_________________/
      __                                        __
NWAIT   \______________________________________/  \_

RAM  (seq read, then Nseq/writes)
      MEMDECWAIMEMWAIMEMDECWAIMEM
  q3  ____________________________
  q2  ____________________________
  q1  ____________________________
         __    __        __
  q0  __/  \__/  \_____/  \____
      __    __    __    __
NWAIT   \__/  \__/  \__/  \_
*/

q0.D     =  decST &  ramDEC & !resmap
         #  intST &  sCYC &  ramDEC & !resmap
         #  memST &  sCYC &  ramDEC & !resmap & !Nread /* ram SEQ */
         # !Nwait &  q1                 /* div by 3 ring counter */
         #  waiST & !q0 & !q1 & !q2 & !q3  /* prevent init lockup */
         ;
```

```
q1.D     =  decST &  hiDEC       /* set for ROM/IO start */
         #  intST &  sCYC &  hiDEC
         #  decST &  resmap
         #  intST &  sCYC &  resmap
         #  waiST & !q0 & !q1 &  q2
         #  waiST & !q0 & !q1 &  q3
         #  extDEC              /* force count on external space */
         ;

q2.D     =  decST &  hiDEC       /* set for ROM/IO start */
         #  intST &  sCYC &  hiDEC
         #  decST &  resmap
         #  intST &  sCYC &  resmap
         # !Nwait & !q0 & !q1 & !q2 &  q3
         # !Nwait &  q2 &  q0
         # !Nwait &  q2 &  q1
         ;

q3.D     =  decST &  hiDEC       /* set for ROM/IO start */
         #  intST &  sCYC &  hiDEC
         #  decST &  resmap
         #  intST &  sCYC &  resmap
         # !Nwait &  q3 &  q0
         # !Nwait &  q3 &  q1
         # !Nwait &  q3 &  q2
         ;

Nwait.D  =  waiST & !q3 & !q2 & !q1 & q0    /* WAI->MEM */
         #  Nwait &  iCYC                   /* MEM,INT->INT */
/*   extra terms for zero wait-state external memory space */
         # !Nwait &  extDEC &  erdy         /* DEC/WAI -> extMEM */
         #  intST &  sCYC &  extDEC &  erdy /* mrg I-cyc ->extMEM */
         #  memST &  sCYC &  extDEC &  erdy & !Nread /* SEQ reads */
         ;

state.D  = !Nwait                           /* DEC,WAIT->WAIT,MEM */
         #  intST &  sCYC                   /* INT -> WAIT,MEM */
         #  memST &  sCYC &  ramDEC & !Nread & !resmap /* ram SEQ */
/*   extra terms for zero wait-state external memory space */
         #  memST &  sCYC &  extDEC         /* ext SEQ */
         ;

/*   note this pin is also used for NRAMOE */
ios.D    =  waiST & ioDEC & Ntrans &  q3 & !q2 & !q1  /* start after 5T */
         #  waiST & ioDEC & Ntrans & !q3 &  q2
         #  waiST & ioDEC & Ntrans & !q3 &  q1       /* complete at 11T */
         # !Nwait & ramDEC & !resmap & !Nread        /* DEC/WAI->MEM */
         #  intST & ramDEC & !resmap & !Nread &  sCYC /* I-cyc->MEM */
         #  memST & ramDEC & !resmap & !Nread &  sCYC /* ram seq */
         ;
```

B.2 Decoder GAL

```
Name            PIEDC2 ARM60-PIE Decoder;
Partno          PIEDC2;
Revision        02;
Date            17/01/92;
Designer        David Flynn;
Company         Advanced RISC Machines Ltd;
Assembly        ;
Location        U9;
Device          g20v8;

/*************************************************************************/
/*   Allowable Target Device Type:        GAL20V8A                       */
/*************************************************************************/

/**   Inputs   **/

pin[1]          = ck;
pin[2]          = a31;
pin[3]          = a30;
pin[4]          = abort;
pin[5]          = Nread;
pin[6]          = Nmreq;
pin[7]          = seq;
pin[8]          = state;
pin[9]          = Nwait;
pin[10]         = resmap;
pin[11]         = Nbyte;
pin[13]         = Noe;
pin[14]         = a1;
pin[15]         = a0;

pin[23]         = clk;

/**   Outputs   **/

pin[22]         = !we3;
pin[21]         = !we2;
pin[20]         = !we1;
pin[19]         = !we0;
pin[18]         =  rom;
pin[17]         =  ram;
pin[16]         = !io;

/**   definitions   **/
$define decST   (!Nwait & !state)
$define waiST   (!Nwait &  state)
$define memST   ( Nwait &  state)
$define intST   ( Nwait & !state)

$define ramDEC  (!a31 & !a30)
$define extDEC  (!a31 &  a30)
$define ioDEC   ( a31 & !a30)
$define romDEC  ( a31 &  a30)
```

```
$define hiDEC   ( a31)

$define iCYC    ( Nmreq)
$define sCYC    (!Nmreq &  seq)
$define nCYC    (!Nmreq & !seq)
$define mCYC    (!Nmreq)

/**   Logic Equations   **/

rom.D    =  decST & !Nread &  romDEC
         #  intST & !Nread &  romDEC &  sCYC
         #  decST & !Nread &  resmap
         #  intST & !Nread &  resmap &  sCYC
         #  rom   & !Nread & !Nwait
         #  rom   & !Nread & !state
         ;

ram.D    =  decST &  ramDEC & !resmap
         #  intST &  ramDEC & !resmap &  sCYC
         #  ram   &  ramDEC & !resmap & !Nwait
         #  ram   &  ramDEC & !resmap & !state
         #  ram   &  ramDEC & !resmap &  memST &  sCYC & !Nread
         ;

io.D     =  decST &  ioDEC &  Nbyte & !resmap
         #  intST &  ioDEC &  Nbyte & !resmap &  sCYC
         #  io    &  ioDEC &  Nbyte & !resmap & !Nwait
         #  io    &  ioDEC &  Nbyte & !resmap & !state
         ;

/**  combinational Write strobes (if NO ABORT) with set-up+hold */
/**  little-endian byte decodes                                 */
we3      =  decST & !abort & !clk &  Nread &  Nbyte
         #  decST & !abort & !clk &  Nread & !Nbyte &  a1 &  a0
         #  intST & !abort &  sCYC & !clk &  Nread &  Nbyte
         #  intST & !abort &  sCYC & !clk &  Nread & !Nbyte &  a1 &  a0
         #  we3   &  memST
         #  we3   &  clk
         ;

we2      =  decST & !abort & !clk &  Nread &  Nbyte
         #  decST & !abort & !clk &  Nread & !Nbyte &  a1 & !a0
         #  intST & !abort &  sCYC & !clk &  Nread &  Nbyte
         #  intST & !abort &  sCYC & !clk &  Nread & !Nbyte &  a1 & !a0
         #  we2   &  memST
         #  we2   &  clk
         ;

we1      =  decST & !abort & !clk &  Nread &  Nbyte
         #  decST & !abort & !clk &  Nread & !Nbyte & !a1 &  a0
         #  intST & !abort &  sCYC & !clk &  Nread &  Nbyte
         #  intST & !abort &  sCYC & !clk &  Nread & !Nbyte & !a1 &  a0
         #  we1   &  memST
         #  we1   &  clk
         ;
```

```
we0      =  decST & !abort & !clk &  Nread &  Nbyte
         #  decST & !abort & !clk &  Nread & !Nbyte & !a1 & !a0
         #  intST & !abort &  sCYC & !clk &  Nread &  Nbyte
         #  intST & !abort &  sCYC & !clk &  Nread & !Nbyte & !a1 & !a0
         #  we0   &  memST
         #  we0   &  clk
         ;
```

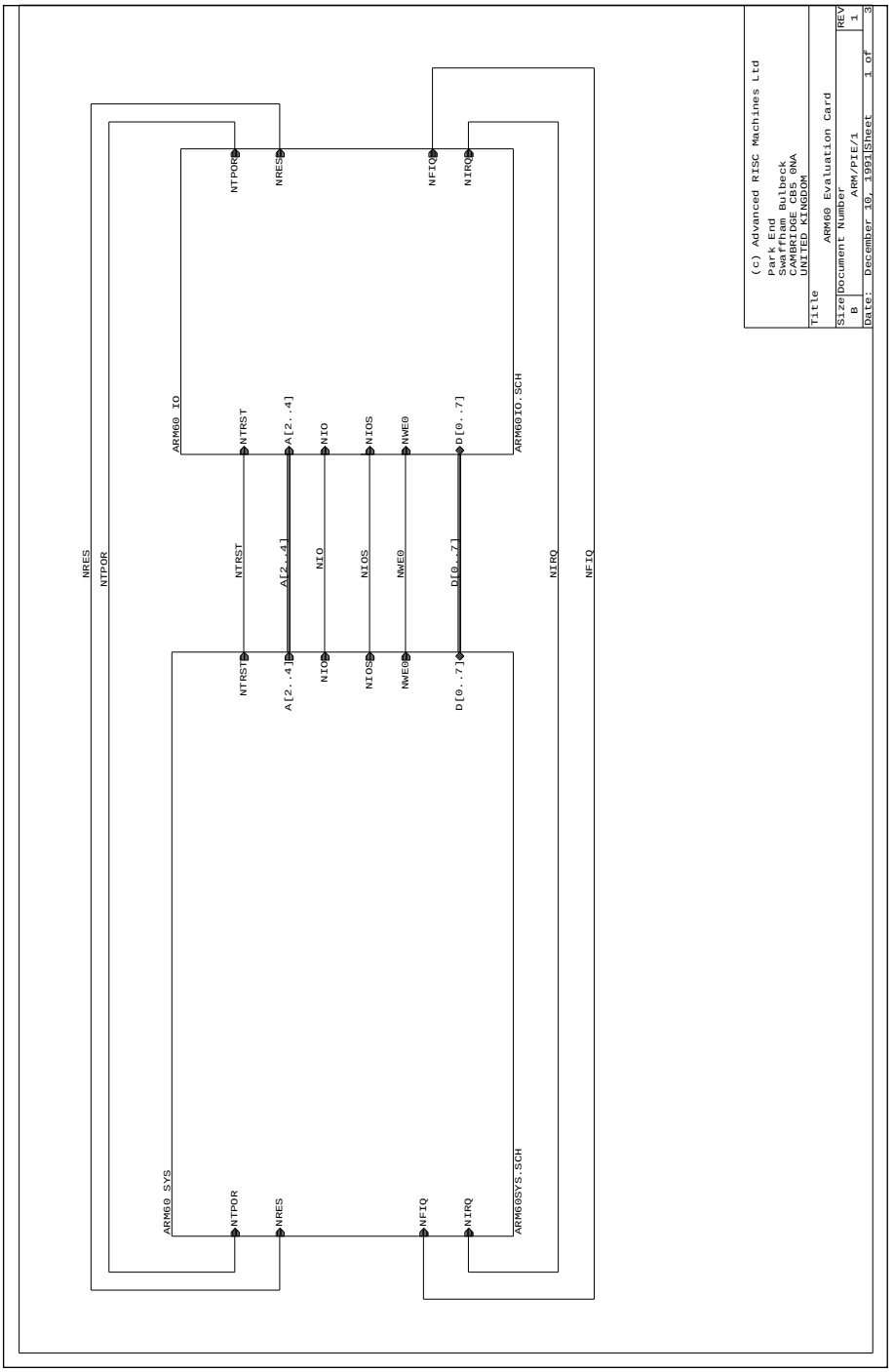
Appendix C

C.1 Circuit schematics

Three schematics are included in this appendix and can be found in the following pages:

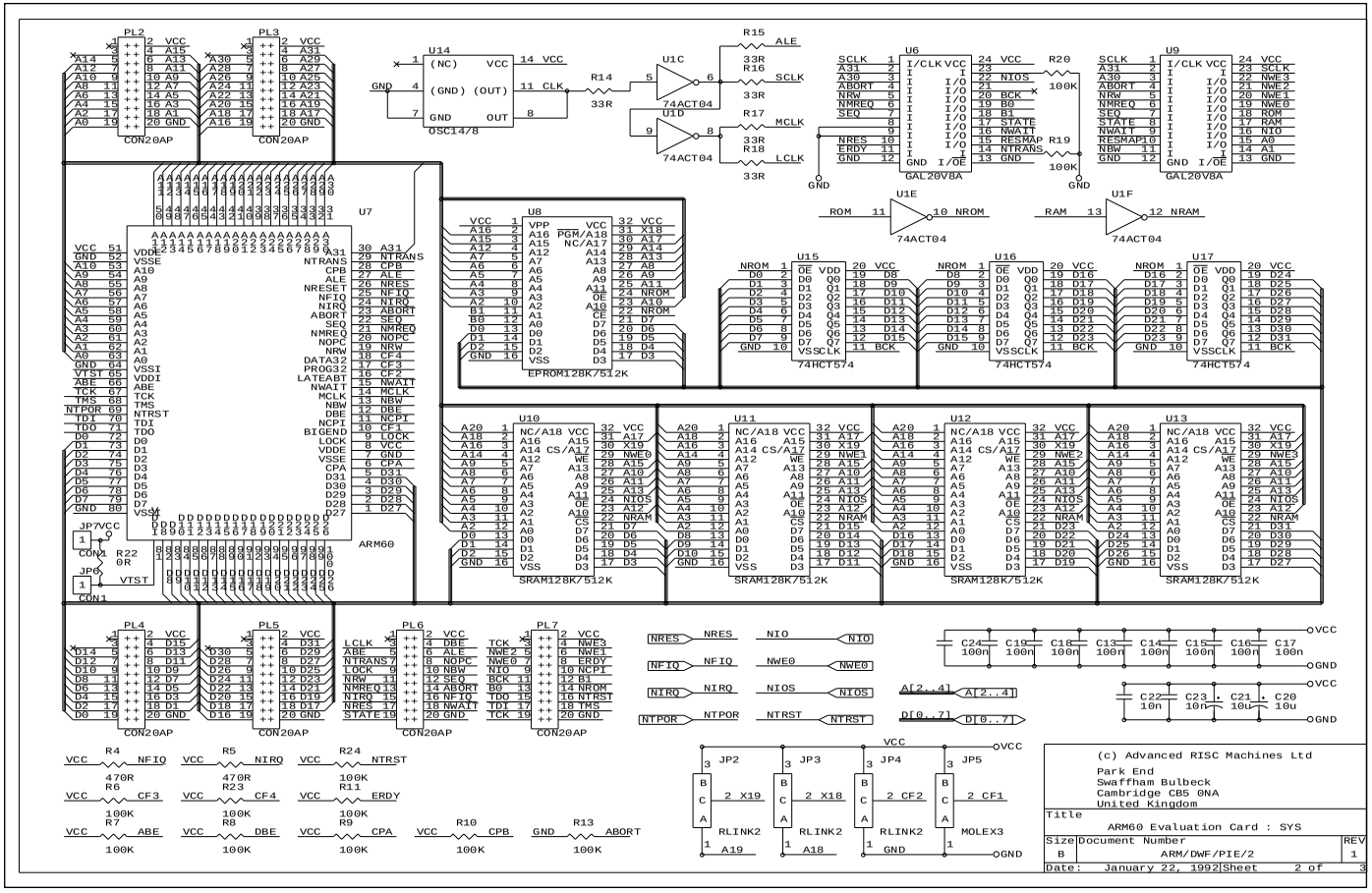
- Design Hierarchy
- System
- I/O subsystem

Schematic 1: Design Hierarchy

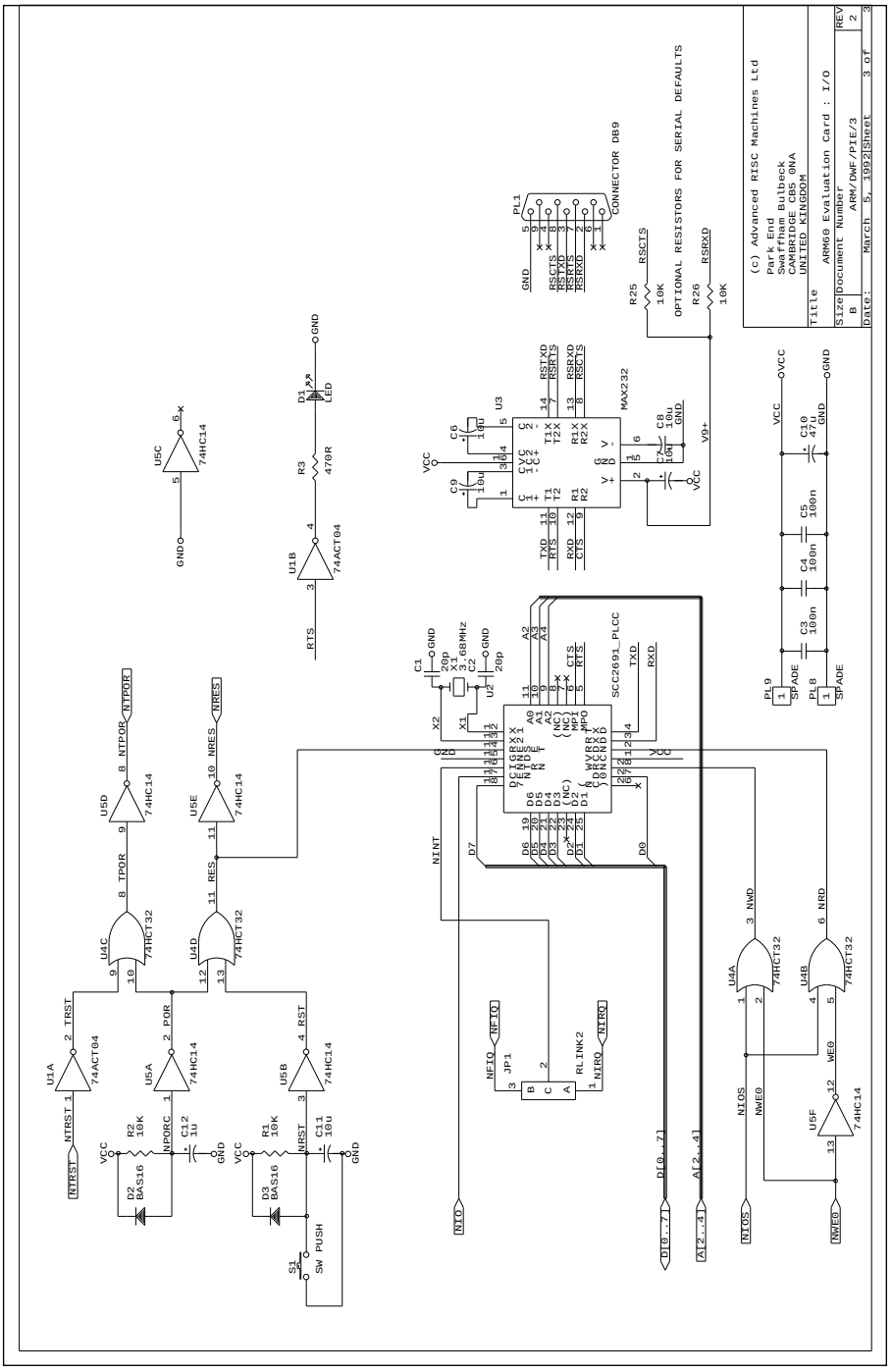


Schematic 2: System

Note: Resistors R13, R19 and R20 may be replaced by 3k3 resistors on some versions of this board.



Schematic 3: I/O Subsystem



Index