IOMD
Functional Specification

Distribution:	COMPANY CONFIDENTIAL
Title:	IOMD Functional Specification
Drawing Number:	0297,030/FS
Issue:	4
Date:	23rd June 1993
Change Number:	ECO 3196
Last Issue:	3 (ECO 3150 of 23rd March 1993)

Contents

1. Introduction

This document is the specification for IOMD, a memory and I/O controller chip developed primarily for the Medusa platform. IOMD is designed to work with an ARM CPU with an on-chip memory management unit (MMU), such as ARM610. It is not intended for use with processors without on-chip MMU, such as ARM2 and ARM3.

IOMD is intended for middle to high-end desktop machines in the Acorn range. It is designed to be used with the VIDC20 video controller. Although it is not planned to use the device in a portable product, there may be a future requirement to use a derivative of the device in such a machine.

The outline specification of the Medusa platform is detailed in the document "Medusa Project Specification", Acorn drawing number 0397,000/PS and this should be referred to for a detailed specification of the machine. This document describes a machine with ARM610, VIDC20, IOMD, 0, 1 or 2MB of VRAM, and up to 64MB of DRAM on SIMM modules. A dedicated network upgrade slot will be present, in addition to 0, 2 or 4 extended versions of the Acorn podule slot.

The machine has 4 I/O DMA channels, two of which are allocated to podule slots 0 and 1. This is in addition to DMA support for sound output, cursor and video RAM control.

History

20th October 1992	AMR 3364	Issue 1	Initial release
20th January 1993	ECO 3110	Issue 2	changed sound support from serial to parallel
19th March 1993	ECO 3150	Issue 3	Pin list added
			addition of DRAM video DMA (to support video fromDRAM)
			Project name changed
23rd June 1993	ECO 3196	Issue 4	clarify detail in Section 9, Device Packaging

Features

- Direct interface to ARM610 or ARM700
- DRAM control for 2 SIMMs (4 DRAM banks)
- Control of VRAM, including generation of transfer cycles
- 16-bit byte-steered bus, for on-board peripherals such as super-IO, SCSI, and for podules
- Most of the functionality of IOC, including interrupt masking, counter timers etc
- PC keyboard interface
- Quadrature mouse interface
- Interface to CD-quality digital sound chip
- Four general purpose I/O DMA channels
- Packaged in a 208 pin SQFP package

2. General architecture

The IOMD ASIC is a physical memory, DMA and I/O controller. It has a CPU interface for an ARM processor with MMU which, with external arbitration and glue logic, can allow an additional processor to be connected. The CPU interface consists of the processor address, data and control buses.

There is a DRAM and VRAM control bus which has RAS, CAS, multiplexed address and other control lines. There are a number of DMA address generators, for sound, cursor, and general I/O DMA. There is also VRAM control logic, including logic to generate transfer cycles.

Since the whole 32-bits of the main system bus connects to IOMD it is possible for IOMD to DMA data from DRAM into itself. There is a 16 bit I/O bus on IOMD, and there is byte steering logic to allow DMA data at arbitrary byte locations to be transferred to/from the I/O system using this bus. The DMA system does not support word packing and unpacking, due to complications with arbitary start and end addresses, and early termination of data transfers. The 16-bit I/O bus forms the lower 16 bits of the 32-bit podule interface. IOMD controls the latches for the upper 16 bits of the extended podule bus, which allows 32 bit transfers.

IOMD contains a large subset of the functionality of IOC, including two general purpose counter/timers (timer 0 and timer 1) and the interrupt control registers. The IOC baud rate and keyboard serial rate timers are not implemented in IOMD nor are all of the general purpose C I/O lines. The allocation of interrupt lines is largely similar to previous machines.

IOMD provides a PC keyboard interface instead of the Archimedes KART interface provided by IOC. This consists of an 8-bit synchronous serial interface, with interrupt generation capability.

The chip contains a quadrature mouse interface. This consists of X and Y counters that are incremented and decremented by mouse movements. The counters wrap when they overflow or underflow, and are read regularly under interrupt. It is suggested that the VSYNC interrupt is used, although the centi-second timer could be used. The VSYNC interrupt allows updating every frame, as there is no point in updating the screen more often than this. The X and Y counters are each 16 bits wide.

IOMD provides support for a stereo sound CODEC chip with an 8 bit interface (eg Analog Devices AD1848). This allows 16 bit stereo input and output. IOMD also supports VIDC sound by implementing support for the VIDC20 sound interface. There are two DMA channels for sound. One is used for linear sound sample output to the CODEC chip. The other is used for input from the CODEC chip, or for the output of VIDC sound. The VIDC20 sound output is fed to the ADC on the CODEC, and this input is converted to the digital domain, filtered, and converted back to analogue by the CODEC's DACs. Since the analogue input path of the CODEC is used for VIDC sound output, it is not possible to output VIDC sound and input sound at the same time, and this is the reason for the the sound DMA channel allocation.

A block diagram of the Victoria platform (as defined by Issue one of the Victoria Functional Specification 0397,000/FS Issue 1) is shown in figure 2.1. A block diagram of the Medusa platform is shown in figure 2.2.

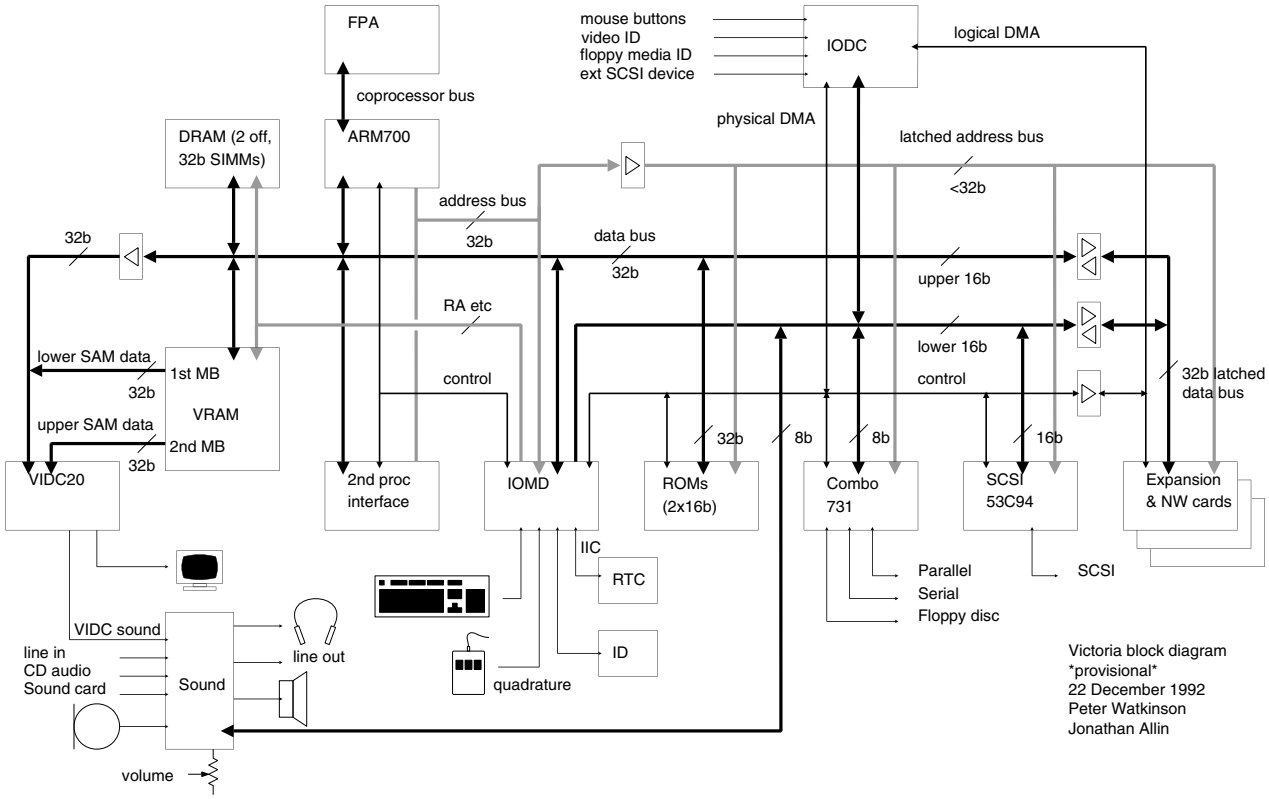

Figure 2.1 Victoria block diagram

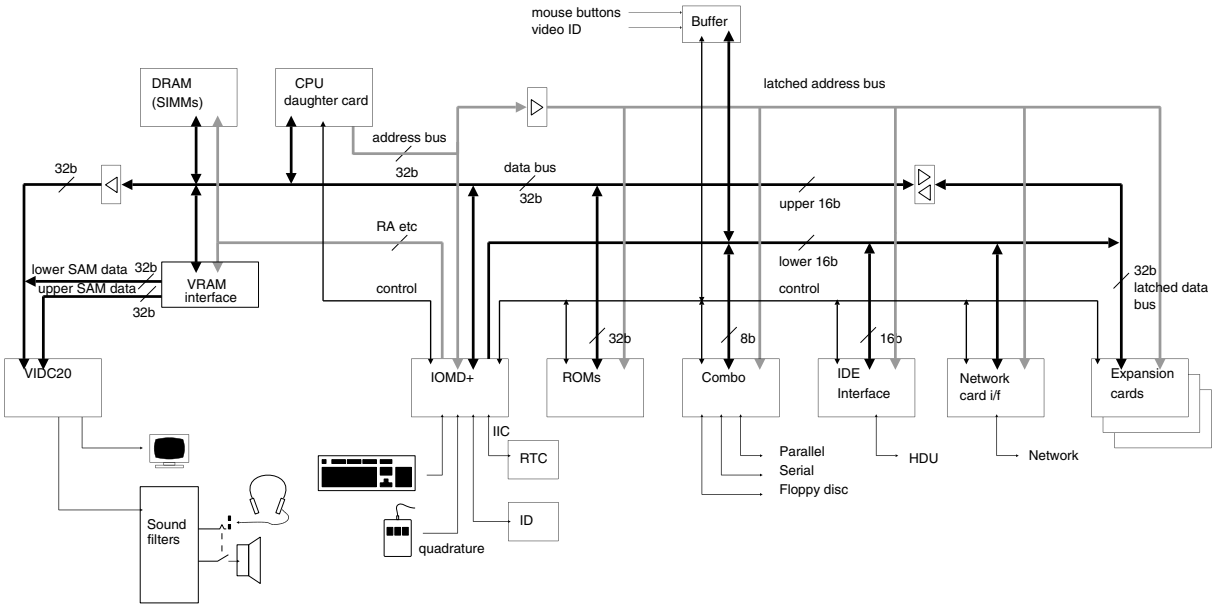

Figure 2.2 Medusa block diagram

3. Description of signals

Name	Type	Function
a[28:0]	IT	Address input from ARM microprocessor. (512MB :- 256MB DRAM, and 256MB others)
d[31:0]	ITO6HZ	Data bus (memory and CPU)
Nbw	IT	Byte/word input from ARM. If this signal is low, a byte access is being requested
dbe	O6M	Data buffer enable. Used to disable the ARM data bus during DMA memory accesses
clk64	I	64MHz clock input
Nfiq	ITO6MZ	Fiq output to ARM (Test mode input during reset)
Nirq	O6M	Irq output to ARM
mclk	O8L	Memory clock output to ARM.
Nreset	ITSO6HD	Active low reset input/output
por	ITS	Power on reset input. Schmitt level input
Nrw	IT	ARM read/write signal
Nmreq	IT	Memory request input from ARM
Npreq	IT	2nd processor request input. For 486 processor card. Higher priority than Nmreq
rclk	O6L	Reference clock output. Nominally 16MHz but can be stretched
pclk	O4	Memory bus clock output to VIDC
Ndras[3:0]	O8L	DRAM RAS line outputs
Nvras	O6M	VRAM RAS output
Ncas[3:0]	O8L	CAS line outputs
ra[11:0]	O12M	VRAM/DRAM address outputs
Nwe[0]	O12M	VRAM/DRAM write enable
Nwe[1]	O6M	VRAM write enable
Ndt[1:0]	O6M	VRAM oe/dt signals
dsf	O6M	VRAM special function output (High for split read, otherwise low)
Nromcs	O4H	ROM chip select
Nprog	O4H	Video controller write strobe
clk16	O6M	16MHz I/O clock
ref8m	O6M	8MHz I/O clock
clk2	O6M	2MHz I/O clock, for synchronous access peripherals
Niorq	O6M	I/O request for external module-type devices
Niogt	IT	I/O grant
Nwbe	O6M	Write data latch output enable
Nrbe	O4H	Read data latch output enable
Nblw	O4	Write data latch control
Nblr	O4H	Read data latch control
Nbl	ITS	Bus latch input from podule bus
Nior	O6M	I/O read for PC-I/O and podules
Niow	O6M	I/O write for PC-I/O and podules
Npboe	O4H	Podule buffer output enable. Output enables lower 16-bits of podule buffer
Nccs	O4H	Chip select for combo chip
Ncdack	O4H	Combo dack signal (really another chip select signal)
Nsccs	O4H	Chip select for SCSI chip
bd[15:0]	ITO6HZ	Peripheral data bus, for internal I/O peripherals (SCSI, super-IO), and podule interface
lrnw	O6M	Latched read/not write output
Npfiq	ITS	Podule fiq input. Active low, level triggered

Npirq	ITS	Podule irq input. Active low, level triggered
Nsintr	IT	Serial interrupt input. Level triggered active low
Nscirq	IT	SCSI interrupt. Level triggered active low
Nfintr	IT	Floppy interrupt request. Level triggered active low
Nindex	ITS	Floppy disc index input. Falling edge triggered
flyback	IT	Flyback from VIDC, rising edge triggered
fdrq	IT	Floppy drq, connects to fiq interrupt input, not really DMA. Active high level triggered
pintr	IT	Printer interrupt request. Rising edge triggered
Niext	IT	Extended interrupt. Active low, level triggered
Nsio	O4H	General purpose I/O select output
Neasis	O6M	Extended address space interface select signal
ready	ITS	Extended address space ready signal (low to extend cycle)
Nms	O6M	Module select output
id	ITO4HD	ID chip input/output. Open drain (cf. IOC C[3])
iicd	ITSO4HD	I2C data (cf. IOC C[0])
iicc	ITSO4HD	I2C clock. (cf. IOC C[1])
kclk	ITSO4HD	PC keyboard clock. Open drain
kdata	ITSO4HD	PC keyboard data. Open drain
dreq[3:0]	IT	DMA request inputs
Ndack[3:0]	O6M	DMA acknowledge outputs
tc	O6M	Terminal count output
Nsndrq	IT	Sound request input
Nsndak	O4H	Sound acknowledge output
Nvidrq	IT	Video DMA request
Nvidak	O4H	Video DMA grant
vNc	IT	Video/not cursor
Nse	O6M	Serial port output enable for VRAM
Ncdoe	O6M	Cursor/sound/programming buffer output enable
sc	O6M	Serial port clock for VRAM
qsf	IT	Split port bank number output from VRAM
mousex[1:0]	ITS	Mouse X quadrature inputs.
mousey[1:0]	ITS	Mouse Y quadrature inputs
Nsndcs	O4H	Sound chip select
spdrq	IT	Sound playback DMA request
scdrq	IT	Sound capture DMA request
Nspdack	O4H	Sound playback DMA acknowledge
Nscdack	O4H	Sound capture DMA acknowledge
tdi	ITP	Test data in, for JTAG boundary scan
tdo	O4HZ	Test data out, for JTAG boundary scan
tms	ITP	Test mode select, for JTAG boundary scan
tck	ITP	Test clock, for JTAG boundary scan

IOMD has no equivalent pin to IOC's clk8. A correctly timed clk8 signal can be derived by inverting IOMD's ref8m signal with, eg, an 'AC04

Key :-

I - input, CMOS threshold	IT - input, TTL level threshold	ITS - Schmitt TTL input	
O4 - output (4mA drive)	O6 - output (6mA drive)	O8 - output (8ma drive)	O12 - output (12mA drive)
H - Heavy slew rate limiting	M - Medium slew rate limiting	L - Light slew rate limiting	
D - open drain output	P - internal pullup resistor	Z - tri-state output	

4. Architecture

Clock generation

The master clock source, **clk64,** is a 64MHz signal and this provides all the system timing. The DRAM timing generators use both edges of a 32MHz clock, which is generated from the 64MHz by dividing by two. The peripheral bus is clocked from the master clock divided by 4 (16MHz). The video RAM serial interface uses 64MHz divided by 3 to generate **sc** which clocks the VRAM serial ports at 21.33MHz using a 2:1 mark space ratio. The VIDC20 **pclk** signal must be the same as **sc** during video RAM data transfer, but must be the same as **mclk** during cursor DMA, VIDC20 programming (**Nprog** activity), and sound DMA (if the VIDC20 sound system is being used).

DRAM control

IOMD will directly control two standard 32-bit wide, 72-pin SIMMS. Each SIMM has one or two RAS lines, and 4 CAS lines, one for each byte in the word. Thus, IOMD has 4 RAS lines and 4 CAS lines in total. In addition, IOMD directly supports VRAM, and there is an additional RAS line to select the VRAM.

There are 12 RA address lines, and 3 control bits to control address multiplexing options, meaning there are 8 possible options, of which 4 are considered useful, as shown below. The most significant bit of the DRAM size control is only applicable for VRAM, and is therefore not present for DRAM, and is assumed by the hardware to be zero.

RA[11:0]	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Size 000 (Size 00 for DRAM)													
PA row	25	23	21	20	19	18	17	16	15	14	13	12	1M, 4M, and 16M DRAM
PA col	24	22	11	10	9	8	7	6	5	4	3	2	& 1 bank 1M VRAM
Size 001 (Size 01 for DRAM)													
PA row	25	23	21	11	19	18	17	16	15	14	13	12	256K DRAM & 1 bank 256K VRAM
PA col	24	22	11	10	9	8	7	6	5	4	3	2	
Size 010													
PA row	25	23	21	20	19	18	17	16	15	14	13	12	2 bank VRAM (256K)
PA col	24	22	12	11	10	9	8	7	6	5	4	3	
Size 110													
PA row	25	23	21	20	19	18	17	16	15	14	13	22	2 bank VRAM (1M)
PA col	24	22	12	11	10	9	8	7	6	5	4	3	

A[27:26] are decoded to select the DRAM bank, and hence the appropriate RAS line. This means that the physical memory map may be discontiguous if each bank does not contain the maximum 64MB. Normally, the VRAM used will be 256Kx32, or 256Kx64, with the two banks interleaved on a word basis on the DRAM interface side (see video interface section).

An 8-bit register (DRAMCR) is provided to control the DRAM row address options and is shown below. Four pairs of two bits control the DRAM row address mapping for each RAM bank. Another control register (VREFCR) is used to control the VRAM column address options which will vary depending on whether one or two megabytes of VRAM are fitted. The VRAM control register also contains the refresh timing information, and is shown in figure 4.3 in the section on the refresh options. The format of the DRAM control register is shown below. The DRAMCR is reset to zero when IOMD is reset.

7							0
Bank3 Sz1	**Bank3 Sz0**	**Bank2 Sz1**	**Bank2 Sz0**	**Bank1 Sz1**	**Bank1 Sz0**	**Bank0 Sz1**	**Bank0 Sz0**

Figure 4.1 DRAM Control Register

DRAM (and VRAM) control and timing is controlled by a state machine running at 32MHz. The DRAM interface supports page-mode DRAMs. S-cycles run at 16MHz, and N-cycles take 2.5 times the S-cycle time, ie they run at 6.4MHz. This means that S-cycles take 2 cycles of the 32MHz clock, and N-cycles take 5 cycles of the 32MHz clock. An example DRAM timing waveform is shown in figure 4.2 below. Note that the **CAS** signal is delayed slightly for a write, to allow more set up time for data into the RAM. It may also be necessary to delay the column address during writes, in order to ensure that there is enough column address hold time.

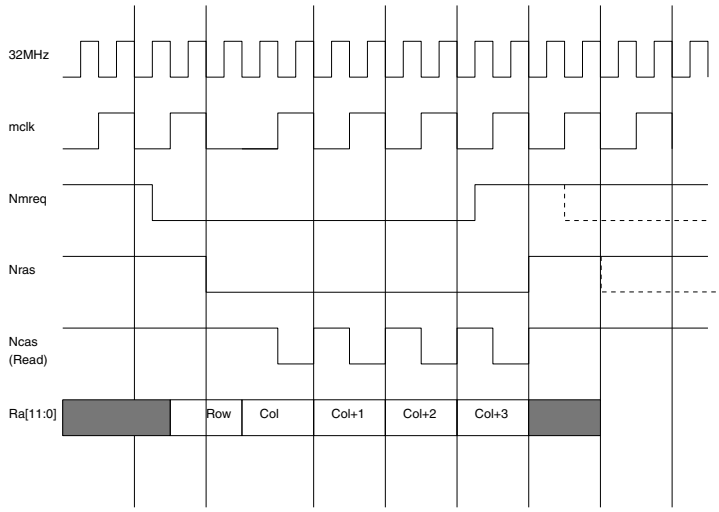

Figure 4.2a Example DRAM control signals (read)

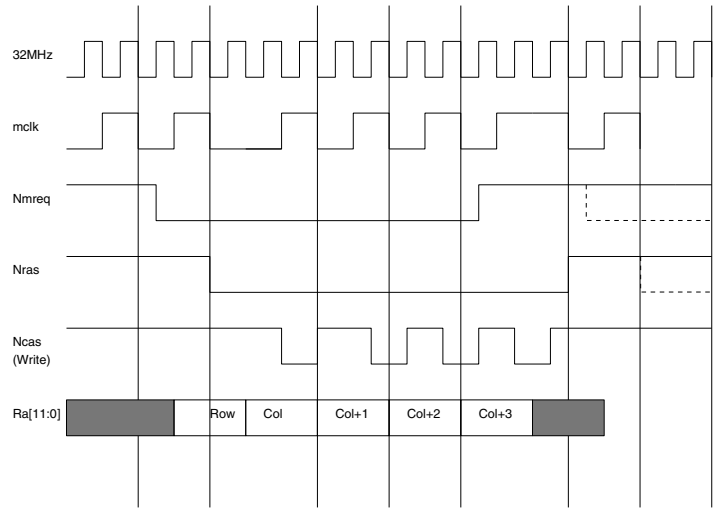

Figure 4.2b Example DRAM control signals (write)

DRAM refresh is performed using CAS-before-RAS refresh. A refresh control register (VREFCR) of which 2 bits are currently defined for refresh operation, is provided to control the refresh rate. This register also contains the VRAM control bits. The register is reset to zero when IOMD is reset. The refresh rate is derived from the reference clock (64MHz), and can be set to 16 or 128µS, or disabled. Unused bits in this register should be written to 0, but their state is undefined when read, including after reset. Thus bits 4, 2 and 1 are all undefined on a read. DRAM refreshes are staggered by IOMD, to minimise the instantaneous power consumption required.

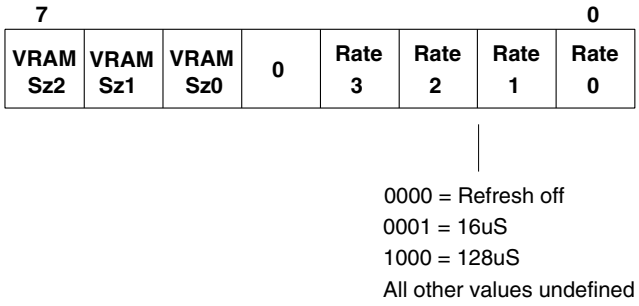

Figure 4.3 VRAM and Refresh Control Register (VREFCR)

ROM control

IOMD will support two 16MB banks of ROM with individually controllable timing parameters. The access time can be varied from around 220nS downwards in steps of 31.25nS. In addition, support is provided for burst mode ROMs which allow rapid access to sequential addresses controlled by A0 and A1. Five control bits per bank are used to control these parameters. Three bits control the basic access speed, and two bits control whether burst is used and the burst access speed. In burst mode, subsequent accesses are shorter than the initial access time. There are two ROM control registers, one for each ROM bank, with the bit allocation as shown in the diagram below. The ROM control registers (ROMCR0 and ROMCR1) are reset to zero when IOMD is reset. Unlike machines based on MEMC1A, IOMD allows writes to be made to the ROM areas. The ARM MMU should be programmed to prevent writes to the ROM space if the ROM area does not contain writeable memory such as SRAM.

During sequential accesses to ROM, Nromcs will stay low, and the ARM addresses change. Thus Nromcs is a combinatorial signal, and a falling edge on it cannot be relied upon. The timing of Nromcs is the same for reads and writes. Nromcs typically rises 0 to 5 nS before mclk falls at the end of the cycle.

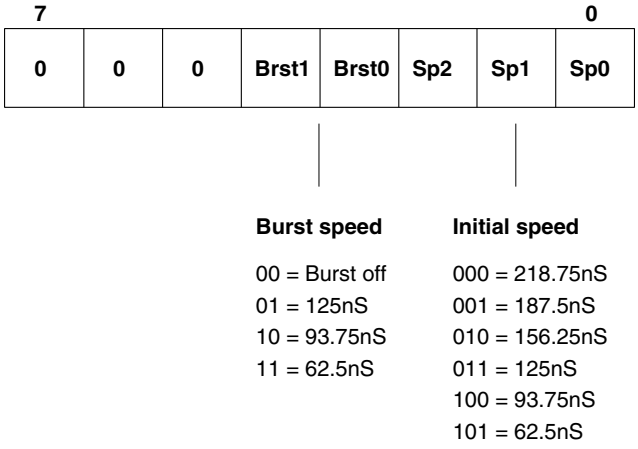

Figure 4.4 ROM Control Register, one per ROM bank (ROMCR0 & ROMCR1)

Video interface

The video interface is designed to support VIDC20 used with either DRAM or VRAM. The VIDC20 VRAM interface mode is never used. Instead, VIDC20 is used either in 32 bit DRAM interface mode for DRAM or one bank of VRAM, or in 64-bit DRAM interface mode with two banks of VRAM. Cursor, sound and programming information comes from the main data bus, as does the video data when the VRAM is not used. When used, the VRAM serial ports connect directly to the VIDC20 data input ports, The main data bus is normally isolated from the data port by a '244 type buffer. An active low output enable signal (Ncdoe) is generated by IOMD to control the buffer. The buffer should have a propagation delay of less than 10nS. The serial port of the VRAM connected to d[31:0] of VIDC must be disabled during cursor DMA (horizontal retrace time), programming and sound DMA. When both banks of VRAM are fitted, the **`Nwe`** and **`Noe/dt`** lines are used to interleave the VRAM banks on the memory bus side on a word basis, using **`A[2]`** to select the bank required.

At the start of each frame, IOMD does a full transfer to the VRAM, to initialise the video pointer in the SAM. It then does split transfers whenever needed, as indicated by the qsf pin of the VRAM. The full-transfer is done on the last but one line of the flyback period. This is to ensure that the VRAM serial port is loaded with new data if the frame buffer has been updated during flyback. To do this, IOMD must count the flyback lines, and when the last line but one is reached, do the transfer. The 8 bit FSIZE register in IOMD must be programmed with the number of flyback lines minus 1.

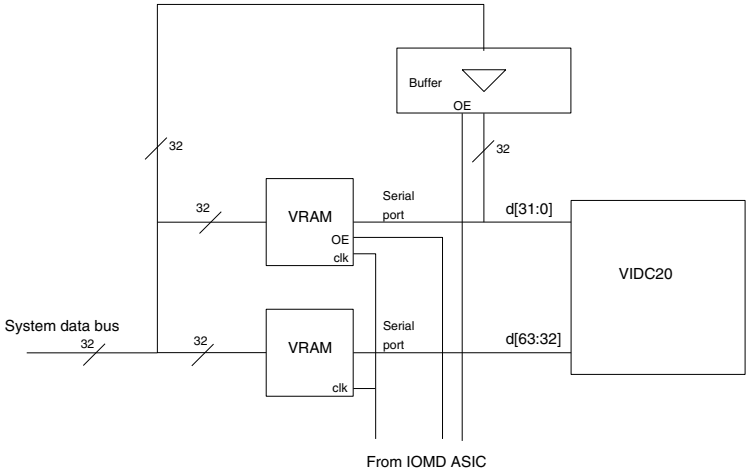

Figure 4.5 Connection of VIDC20 and VRAM to IOMD.

The VRAM serial ports are clocked at 21.33MHz using a 'bursty' clock. When VIDC20 asserts **vidrq** the response from IOMD is to assert **vidak** and to clock the VRAM serial port clock **sc** four times. The VRAM serial clock is derived from the 64MHz reference clock by dividing by 3, such that the clock is high for 2 64MHz periods, and low for one 64MHz period. Data is clocked out of the VRAM serial port on the rising edge of **sc** and is clocked into VIDC on the falling edge of **pclk**, which is the same as **sc** during video data transfers (see Figure 4.6). **pclk** is the same as **mclk** during cursor, programming and sound transfers. A change on the **qsf** output from the VRAM indicates that a transfer cycle is required. The current video pointer VCUR is then incremented by the split port SAM length, and a transfer cycle is requested to the main bus arbiter.

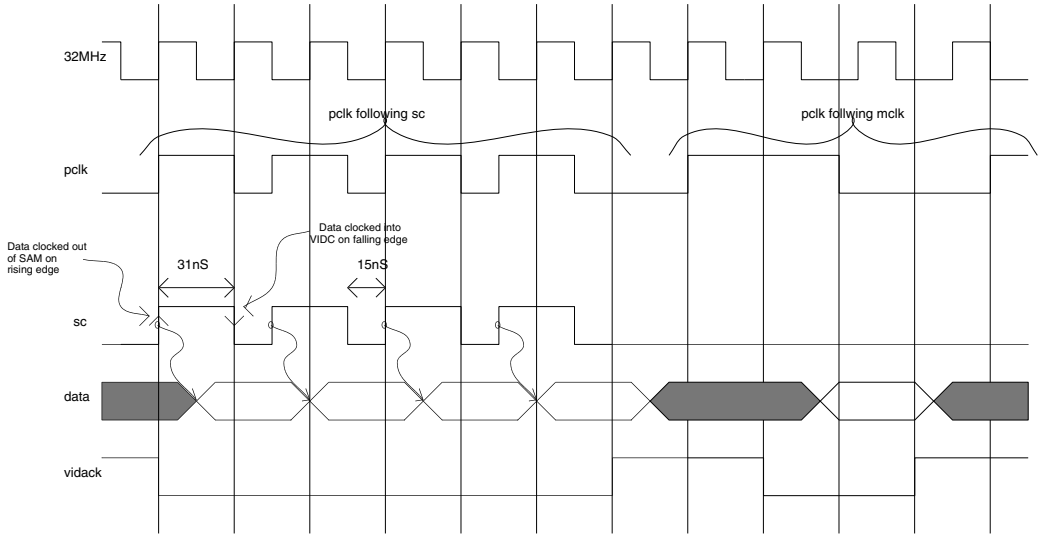

Figure 4.6 VRAM clocking signals

Using a 21.33MHz interface to VIDC gives a maximum peak bandwidth from the frame buffer to the display of 85.33MB/sec for a 1MB VRAM machine using a 32-bit interface, and 170.66MB/sec for a 2MB VRAM machine, using the 64 bit interface. As can be seen from figure 4.6, there is 31nS from clocking the SAM port to clocking the data into VIDC.

DMA channels

There are four general purpose I/O DMA channels, and two sound channels. In addition, there are two further channels for video cursor data and VRAM transfer addresses. The DMA channels have fixed priorities as shown below.

Priority	Channel
0 (highest)	Cursor
1	Sound chan. 0 (VIDC sound out and linear sound in)
2	VRAM transfer cycle
3	Sound chan. 1 (linear sound out)
4	DRAM and VRAM refresh
5	I/O DMA channels ('round robin between channels 0 to 3)
6 (lowest)	ARM

I/O and sound DMA

The I/O and sound DMA channels have two sets of pointers, so that data transfers may be 'double buffered' as is the case with the MEMC1a sound DMA channel. The DMA pointers consist of two pairs (A and B) of 29 bit current address registers, and 12 bit end pointers which also have 2 control bits in them. The current and end pointer addresses are inclusive addresses, and the end address is the address of the last transfer, which will depend upon the transfer size. The current address registers are divided into two fields. The lower 12 bits form an offset into a 4K page. The upper 17 bits form a static page number. The end register consists of a 12 bit page offset in the lower 12 bits of the register, and two control bits in the most significant bits of the register. Each channel also has an 8 bit control register, and a three bit status register. The control register is readable and writable, and the status register is read only. Five of the control register bits (Inc[4:0]) control the amount by which the current pointer is incremented after each transfer. Another bit (D) controls the direction of transfer, and one bit (E) is used to enable and disable the channel. The status bits, and the C bit of the control register are described later.

DMA transfers are constrained to a single physical page by the following mechanism. The bottom 12 bits (page offset) of the current pointer is incremented on each DMA transfer by the programmed increment, and is compared to the bottom 12 bits of the end pointer. The page number is not incremented, and is not compared. This has the effect that it is possible to program the end pointer to be less than the initial page offset, causing the DMA address to wrap around at the end of the page. This is unlikely to be useful.

In order to implement the double buffering mechanism, each pair of registers is used alternately. An interrupt is raised when one of the pair completes its transfer. If the other pair has been programmed by this time, DMA continues using this pair.

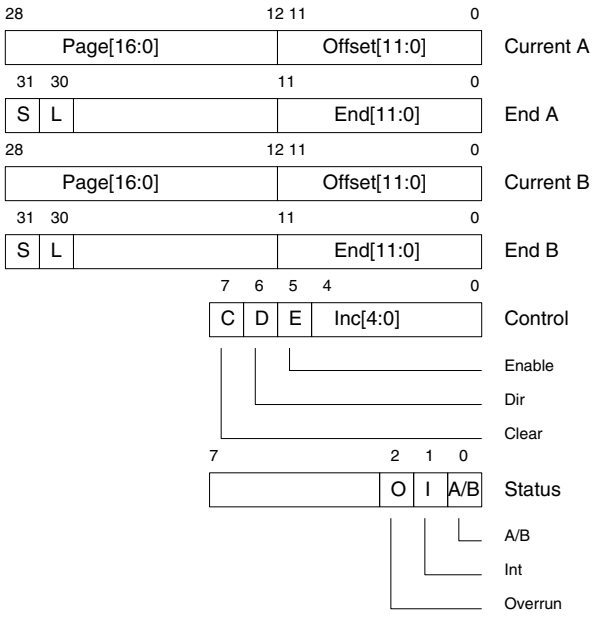

Figure 4.7 DMA Address registers (one set per channel)

The increment field in the control register determines the channel increment as follows:-

00001	Byte
00010	Half-word
00100	Word
10000	Quad-word

The I/O DMA channels support byte, half-word and word transfers only. The VIDC sound channel can support quad-word DMA, but only when DMA is to VIDC, not when it is from the CODEC. The CODEC DMA channel should always be programmed for word transfers. The increment has a different meaning for the video channel (see later).

The direction bit in the control register defines the direction of the DMA transfer. When set, the transfer is from peripheral to memory. This does not apply to sound channel 1, which is read only. Sound channel 0 uses this bit to control whether VIDC sound output mode, or CODEC sound input mode is in use.

Each DMA channel is controlled by a simple state machine. The state machine is able to run when the enable bit is set. The current state is visible in the status register and these bits are read only. The A/B bit indicates which pair of current/ end pointers is in use. The Int bit indicates when the channel is requesting an interrupt. The Overrun bit indicates when a channel has stopped because it finished a transfer, and the other pointer pair had not been programmed. Writing a 1 to the C bit of the control register resets the state machine to state 110. The C bit of the control register is self clearing and always reads zero.

The S (Stop) and L (Last) bits in each end register control the behaviour of the channel when transfer of a buffer completes. The S bit must be set if the TC pin is to be asserted at the end of the buffer. The L bit is an indication that the next transfer on that buffer will be the last. It is normally set by IOMD, but must be set by the ARM in the case where the channel is being initialised for a single transfer. For buffers that require more than one transfer, the L bit should be cleared when the end register is written.

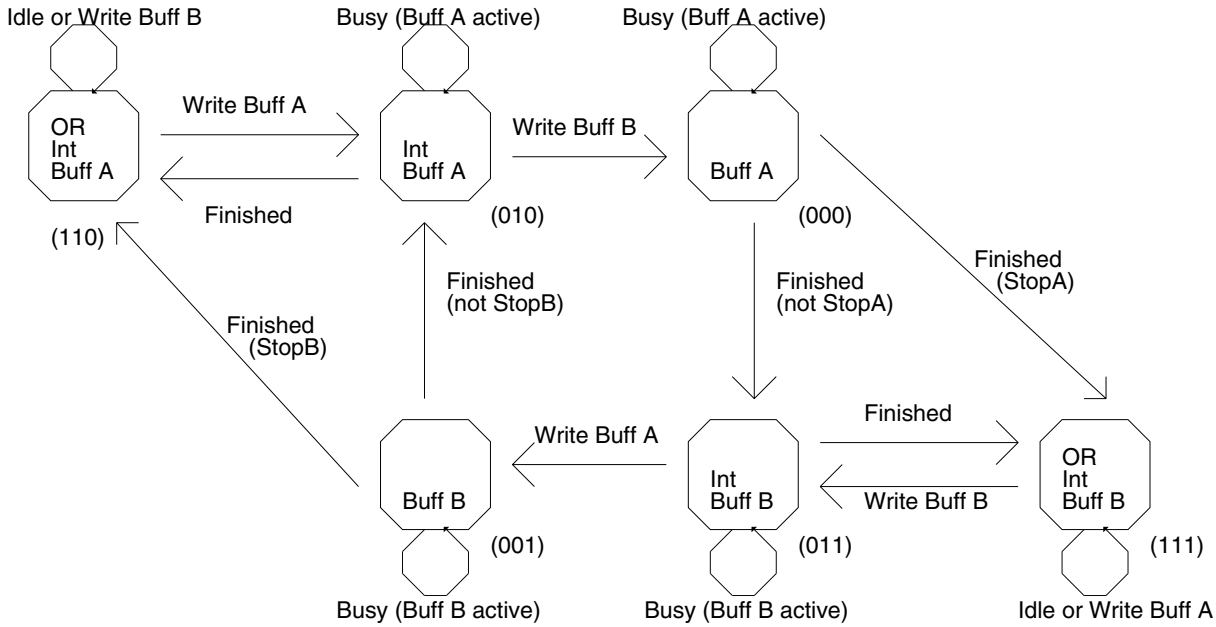

Figure 4.8a Hardware DMA state machine diagram

In the diagram above (figure 4.8a) the three bits shown beside each state correspond to the three bits of the status register, and directly reflect the state of the DMA state machine. At reset, the state machine enters state 110. This state and state 111 are the idle states in which no DMA transfers occur. The transition between states occurs either by a buffer finishing, or by the ARM programming the next pointer pair. The current and end pointers must be programmed in that order, as it is the write to the end pointer which actually causes the state transition.

In practice, a complete DMA transaction is performed by a software state machine, as shown in diagram 4.8b, where N is the buffer number being transferred and LastN is the buffer number of the last buffer to be transferred, which should have

the Stop bit set when programmed. When the last buffer has been programmed in, and the next interrupt happens, a dummy value is programmed into the next buffer pair in order to clear the interrupt. The following interrupt happens after the last buffer has been transferred, at which point DMA can be disabled and the channel reallocated. When the 'Wait for I' box is encountered, the software checks the I (Int) bit: if set it continues to the next step; if clear the software exits and waits for the next DMA interrupt. After a buffer with the S bit set has been transferred, the hardware always enters one of the idle states, it never continues to the other buffer. The scheme is designed to cope with all possible cases of overrun.

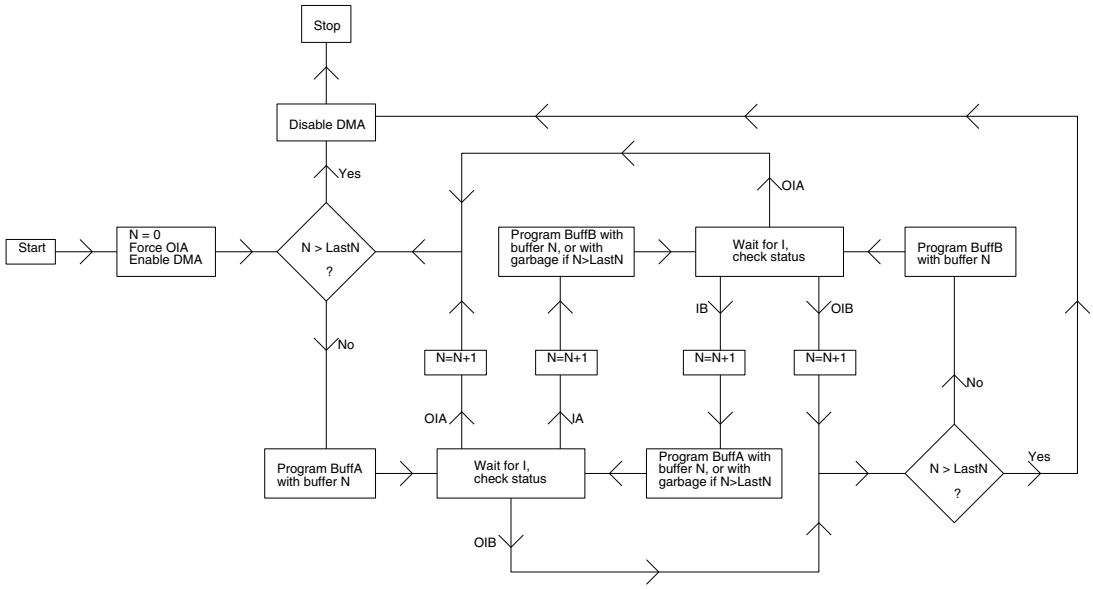

Figure 4.8b Software DMA state machine diagram

Additionally there is a set of interrupt registers for the DMA channels. For each channel there is a mask bit, a status bit, and a request bit, the status bit being a replication of the I bit in the DMA state machine. There are 6 interrupting DMA channels in total, and the interrupt registers are arranged such that there is a DMA channel per bit of each register, with 2 bits per register spare, as shown below.

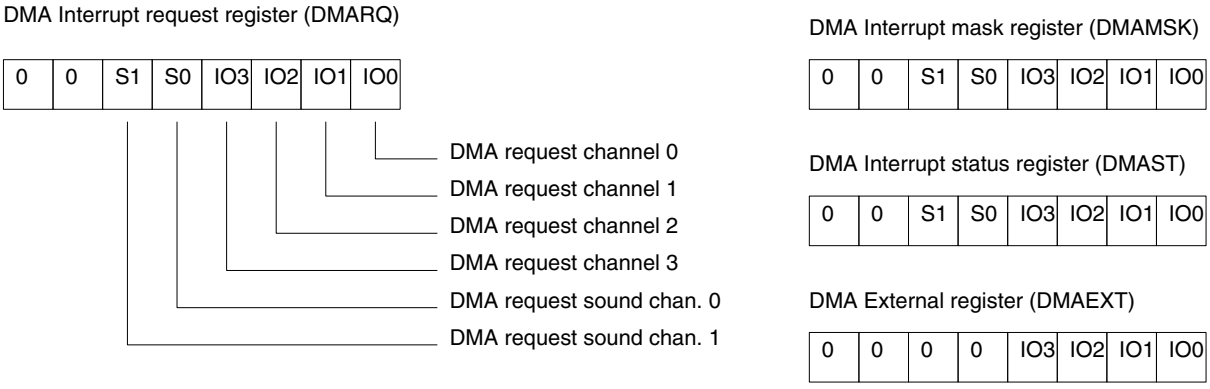

Figure 4.9a DMA Interrupt registers

In figure 4.9a, the DMA interrupt status register indicates which channels have interrupts outstanding. A channel generates an interrupt when it reaches the end of the current buffer. An interrupt will be generated if the relevant DMA interrupt mask is enabled. The DMA request register holds the logical AND of the DMA interrupt status and DMA interrupt mask registers, indicating which channels are requesting an interrupt.

The four I/O DMA channels can be used to access peripherals on either side of the podule buffer. So for DMA reads to internal peripherals such as SCSI, the podule buffer must not be output enabled. However for DMA reads to peripherals on the podule bus, the podule buffer must be output enabled. Since the DMA generators may be arbitarily allocated to physical peripherals, it is necessary to indicate which channel is accessing an external peripheral, and which is accessing

an on board peripheral. To do this, the DMAEXT register is used. Bits 3:0 of the DMAEXT register map to I/O DMA channels 3 to 0, and setting the relevant bit to a 1 indicates that the peripheral is on the podule side of the buffer. There are no DMAEXT bits for the sound DMA channel, as the sound CODEC is always an on-board peripheral.

Cursor DMA

The cursor DMA channel consists of an init register and a current register both of 29 bits. Data is transferred from the current register address in quad words under control of the **`vidrq, vidak`** and **`vnc`** signals. The init register is copied to the current register during flyback. There is no control register for this channel and hence no interrupts may be generated. It is enabled and disabled with the video transfer channel. Cursor data can be held in either VRAM or DRAM. The format of the cursor registers is shown in figure 4.9b below.

Video DMA

The video transfer channel is similar in structure to the MEMC1a video channel and has start, end, init and current registers which are all 32 bits wide.

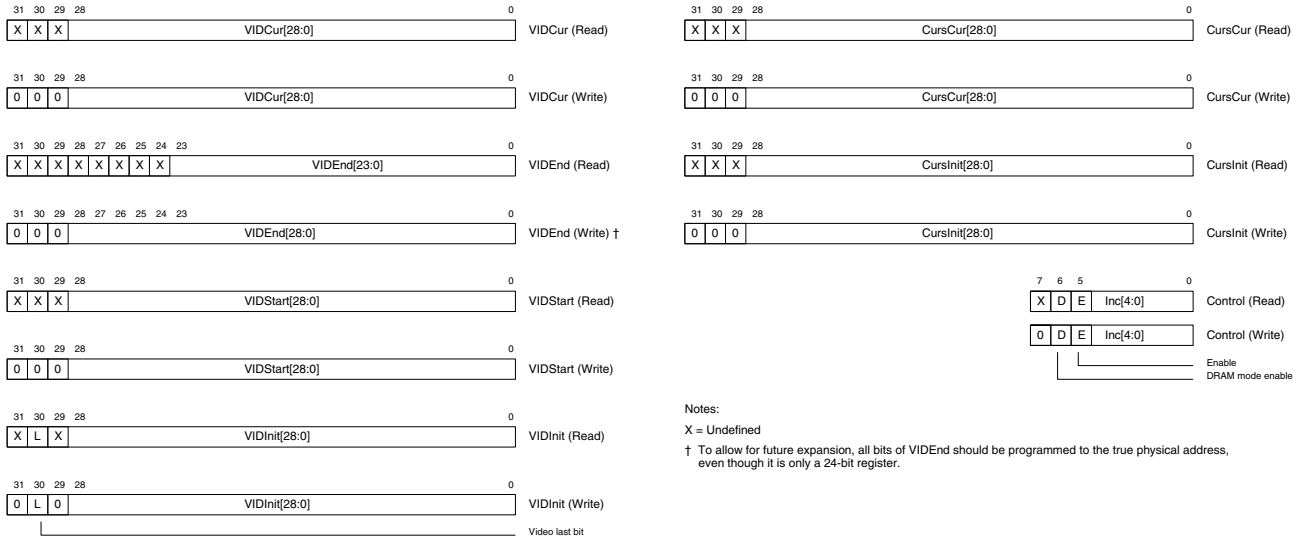

Figure 4.9b Video DMA registers

The VIDStart register is programmed with the address of the start of memory used for the screen buffer. The VIDEnd register is programmed with the address of the last transfer of the video buffer memory. Thus it equals the address of the end of the video buffer, minus the transfer size (ie quad word in DRAM mode or the half-SAM length in VRAM mode). A quad word is 16 bytes. One bank of 1Mbyte VRAM made up from 4 off 2Mbit (256K*8) VRAMs has a half-SAM length of 1*4*512/2=1024 bytes. Two banks have a half-SAM length of 2*4*512/2 = 2048 bytes.

The addresses for the start and end of the video buffer memory have certain restrictions on their values. In all cases they must lie within the bounds of a single 16MByte-aligned block in the physical address space; for VRAM mode this is not a problem since the maximum amount of VRAM is less than 16MByte. In DRAM mode they must each be on a quad word boundary, but are not further restricted. In VRAM mode, they must each be on a half-SAM boundary, with the additional restriction that the buffer size must be a multiple of 2 times the half-SAM length (i.e. N*4096 bytes for 2 banks of VRAM and N*2048 for the 1 bank case). Note that as described above the value programmed into VIDEnd in VRAM mode is not the actual screen memory end address but that value less the half-SAM length. As a result, the VIDStart and VIDEnd values written to IOMD must always differ by an odd number of half-SAM lengths.

The video init (VIDInit) register is programmed with the address of the start of the screen data to be displayed, i.e. the address of the first pixel in the frame. This address must be a multiple of N bytes where N is 16 (i.e. 4 words) in DRAM mode, 8 (2 words) on 2 bank VRAM machines, and 4 (1 word) on 1 bank VRAM machines. For generality, requiring alignment to 16 bytes will ensure compatibility with all possible video DMA configurations. This does remove a possible benefit of the VRAM modes in which horizontal scrolling can be done to a finer resolution than in DRAM modes, but is advisable in order to maximise software portability: older machines based on MEMC1/VIDC1 also have alignment restrictions of 16 bytes on VIDInit.

During flyback, the current register, VIDCur, is initialised to the value in the VIDInit register. VIDCur then increments (by a value defined via VIDCR) until either the end of the frame data, or the end of the video buffer area is reached. In the latter case VIDCur is reloaded with the value in VIDStart, and continues to increment from there, i.e. the data for the frame is wrapped around within the physical memory area defined by VIDStart and VIDEnd; this mechanism allows for efficient vertical scrolling of the video display.

For VRAM modes

The video transfer DMA channel in IOMD controls the generation of VRAM transfer addresses. IOMD uses VRAMs with split SAM port capability. On each split transfer, one half of the SAM is loaded with new data. Thus the video increment should be the half SAM length. The video start and end addresses must be on half SAM boundaries (an even number of half-SAM lengths apart). However, they will usually be on 4K page boundaries, which will always coincide with even half-SAM boundaries.

If the VIDInit register is equal to or greater than the VIDEnd register, implying the frame start address is somewhere in the last half-SAM of the video buffer, then the video last bit in the VIDInit register must be set. This is because the comparison as to whether a transfer is the last in the buffer is always done during a video transfer cycle, for the next video transfer cycle. Thus if the first half-SAM after flyback is the last half-SAM in the frame buffer, the last bit must be set manually.

In VRAM mode, bits 7:0 of the video address are never compared in checking for VIDCur reaching VIDEnd. The number of bits which are compared depends upon the value programmed into the increment register. If the increment is programmed to the value 1, meaning 256 bytes, then all of bits 23:8 are compared. If the increment is 2, meaning 512 bytes, then bits 23:9 are compared. If the increment is 4, only bits 23:10 are compared, and so on. This ensures that the current pointer when compared with the end address, will always match the end address, even if it is not aligned on a half-SAM boundary. Thus if the increment is 2^N bytes, then bits 23:N will be compared.

Thus:

I = Inc*256 = 2^N

N = log2(Inc*256) = 8+log2(Inc)

where Inc is the value in the increment register, I is the actual increment in bytes, and N is the least significant bit compared. Inc must always be a power of 2. ie there must only ever be one bit set in the increment field.

The current register is incremented after each transfer by the half-length of the VRAM serial port. A transfer cycle is requested when the state of the qsf signal changes, indicating that the VRAM has swapped to the other half of the SAM port. The transfer cycle consists of a single VRAM access, with a special combination of control signals applied to the VRAM. The increment is controlled from the control register and is the value in the increment field multiplied by 256. Since 4 bytes are transferred at a time in a 1MByte VRAM machine, and 8 bytes are transferred at a time in a 2MByte VRAM machine, the value in the increment field is the half-SAM length multiplied by 4/256 for 1MByte VRAM, or multiplied by 8/256 for the 2MByte VRAM machine. For example, a VRAM with a SAM length of 512 words has a half-SAM length of 256 words. With 1MByte of VRAM, the video increment would be 256*4/256 = 4. With 2MByte of VRAM, the video increment would be 256*8/256 = 8.

For DRAM mode

The Inc register should be programmed with the transfer size. With IOMD in DRAM mode, this is a quad word (4*4) =16 bytes. Therefore Inc should be 16.

I/O bus

The I/O bus on Medusa is 32 bits wide, of which the lower 16 bits pass through IOMD, and the upper 16 bits are latched and buffered by an external latch. Logic in IOMD is used to position bytes and half words to the appropriate position within the word, for transfer to and from memory. The lower 16 bits from IOMD connect directly to the on-board peripherals, and via a bidirectional buffer to the lower 16 bits of the podule bus. IOMD latches the lower 16 bits of the word, and provides control signals for the external latch and buffer.

The bus supports DMA, and the signals are similar to a cut-down PC-AT bus (ie Intel-style control signals, read strobe, write strobe, DACK etc). The bus is clocked at 16MHz, but a number of clock ticks are required for each transfer. In addition, the bus emulates the I/O bus of previous Acorn 32-bit machines. An 8MHz IORQ/IOGT style interface, and the 8MHz 'S-space' interface signals are provided.

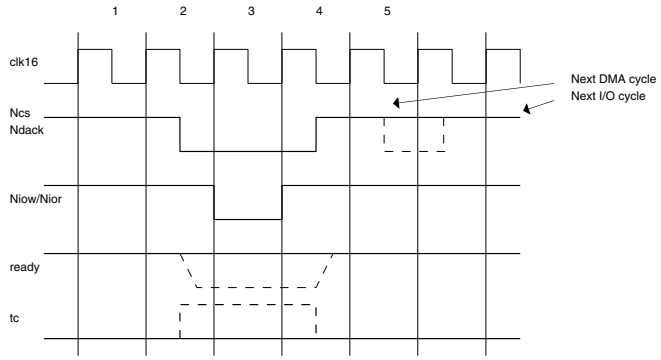

Figure 4.10a Type D I/O cycle

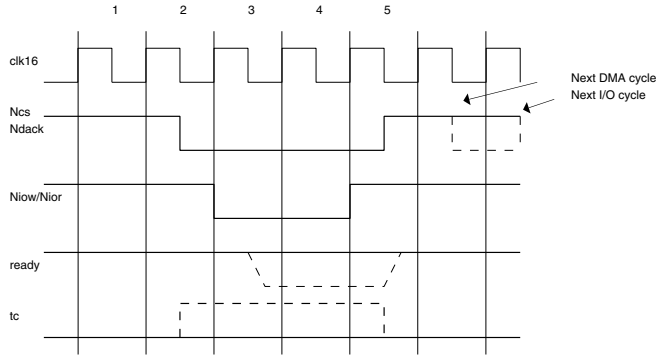

Figure 4.10b Type C I/O cycle

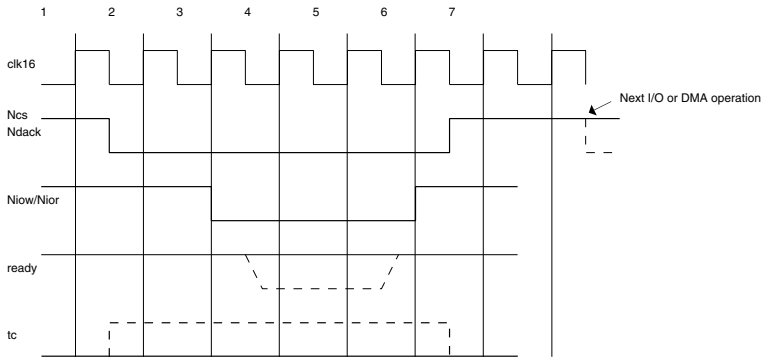

Figure 4.10c Type B I/O cycle

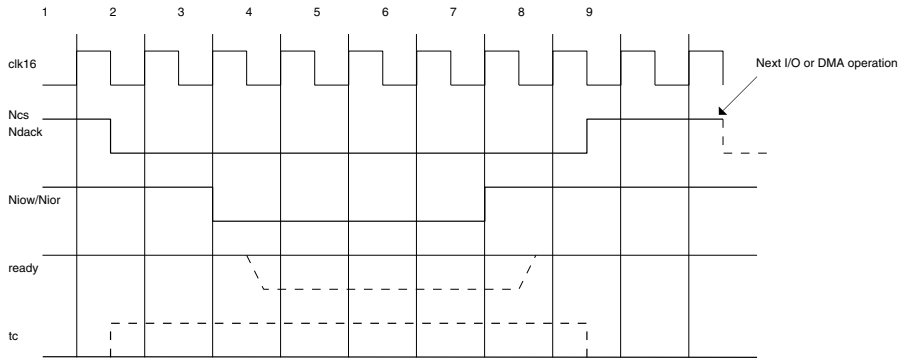

Figure 4.10d Type A I/O cycle

Cycle type	D	C	B	A
Total no ticks (I/O)	4	5	7	9
Total no ticks (DMA)	3	4	7	9
Ncs/Ndack to Niow/r	0.5	0.5	1.5	1.5
Niow/r pulse width	1.0	2.0	3.0	4.0
Niow/r to Ncs/Ndack	0.5	0.5	0.5	1.5
Ncs/Ndack pulse width	2	3	5	7
Ncs high (min)	2	2	2	2
Ndack high (min)	1	1	2	2

Table 4.1 I/O and DMA I/O bus timings

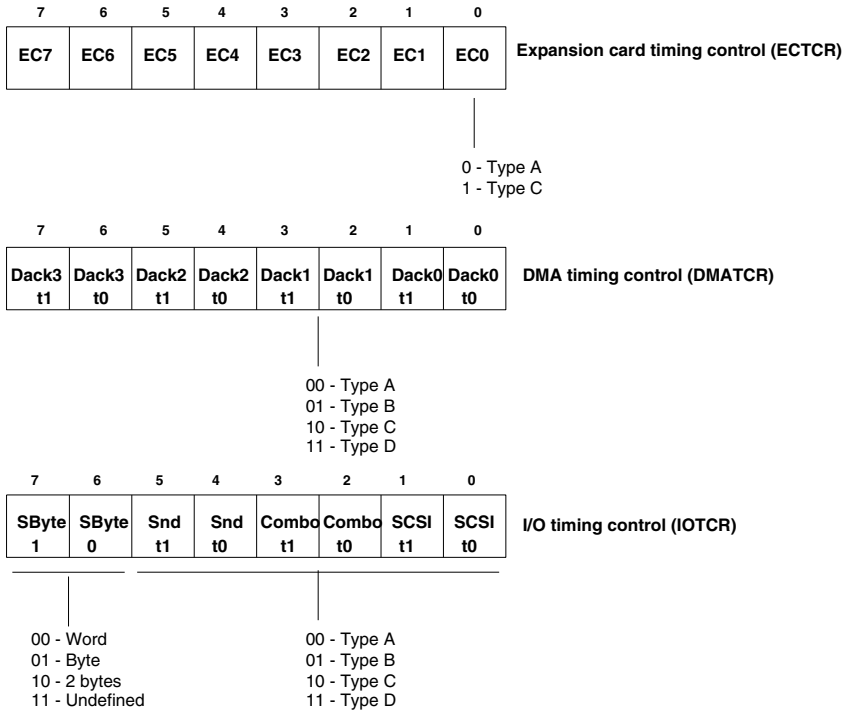

Figure 4.11 I/O, DMA and expansion card timing control registers

There are three types of access timing to the I/O bus, the first two of which are present in MEMC1a/IOC system, and the other one being new to IOMD. The first (8MHz fixed) is the IOC controlled type, in which one of four timings is selected by bits 20:19 of the address. These cycles are based on an 8MHz clock (externally derived clk8). The second type of access (8MHz variable) is also based on an 8MHz clock, and uses the Niorq and Niogt signals, which are referenced to the ref8m pin. The third type of access (16MHz) is referenced to CLK16, the 16MHz I/O clock, and is used for on-board peripherals (SCSI, combo and sound CODEC), for DACK timing to peripherals using DMA, and for the extended address space expansion cards. The basic timings used for these devices are controlled by the I/O, DMA and expansion card timing control registers (IOTCR, DMATCR, and ECTCR respectively). On-board I/O peripherals (Combo, SCSI & CODEC) and DMA have four timings available (types A to D), and expansion cards have only two timings available, these being types A and C.

Various different cycle timings may be selected for the 16MHz accesses. These are known as types A to D, where type D is the fastest and is shown in diagram 4.10a above. Type D has half a 16MHz tick of setup and hold of CS/DACK relative to IOR/IOW and a pulse-width of one 16MHz tick on IOR/IOW. Type C is similar, but has a pulse-width of two 16MHz ticks for IOR/IOW. Type B has one and a half clock ticks of setup and half a tick of hold of CS/DACK to IOR/IOW, and a 3 tick wide IOR/IOW pulse, and type A has one and a half ticks of setup and hold of CS/DACK to IOR/IOW, and a four tick wide IOR/IOW pulse. Their timings are shown in figures 4.10b to 4.10d. Expansion cards have only timings A and C available to them. The fastest two types reduce the minimum CS/DACK width to one tick for DMA, whilst keeping it 2 ticks for programmed I/O. The slower two timings have a minimum of 2 ticks high for both CS and DACK. The table

above shows the DMA and I/O timings.

The point at which DMA request must go away to ensure that another DMA does not happen actually varies, depending upon whether the DMA is a read or a write, and upon the relative phase of rclk and clk16 at the time. For safety however, DREQ should go away by the time DACK rises again. This will ensure that a second unwanted DMA acknowledge does not occur, in all possible cases.

The sound CODEC has 4 control bits allocated to it in the IOTCR register. Bits 5:4 control the timing of the sound chip select, and for normal operation timing B should be selected. Bits 7:6 control the number of bytes per sample that are transferred to the chip, and this must be programmed in addition to the DMA increment in the DAG registers.

The new I/O address space is divided into 8 areas of 16MB each, one for each of eight possible expansion cards allowed in the architecture. Each area has a single bit in the ECTCR register associated with it, to control whether a type A or type C access is used. In addition to the register controlled timing, all programmed I/O (but **not** DMA) accesses may be stretched by the ready pin. This includes the combo and SCSI chip selects, and pseudo-DMA. The ready pin should be valid a 16MHz clock tick before the rising edge of IOR/IOW. Ready is synchronised inside IOMD on the rising edge of clk16, and then the synchronised signal is sampled on the next rising edge. Note that ready must be set up to the rising clk16 internal to IOMD, which will be earlier than the external clk16, so in practice this means it is very difficult to use ready with the fastest I/O timing. However, it may be possible, if ready is driven very quickly from falling CS.

Since the memory clock mclk can be stretched for multiples of clk32 periods during RAM and ROM accesses, the free-running clk16 I/O clock and mclk may be in or out of phase with each other. In order to synchronise the I/O and memory worlds, some programmed I/O accesses must be slipped by half a 16MHz clock tick. This means that programmed I/O and I/O DMA will be slightly slower than would otherwise be the case. The clk16 is always free-running, and the processor clock is slipped to match the I/O clock, to synchronise the memory and I/O systems during programmed I/O and I/O DMA, when the two clocks are out of phase. I/O DMA cannot stretch the cycle timing with the ready pin.

The TC signal will be asserted simultaneously with DACK when using I/O DMA, to indicate the last transfer of a sequence (see the section on DMA channels). It may also be asserted in programmed I/O by writing to the combo DACK+TC address space. This is an area of address space that causes the combo DACK and TC signals to be asserted together. The combo DACK space only asserts the combo DACK signal. Note that the combo DACK is not a real DACK, and is not related to the DMA channels, it is effectively another chip select. TC has the same timing as DACK, and is asserted high to indicate the last transfer, and thus looks like an inverted DACK when asserted in a cycle.

The latched address bus is used to provide addresses to the I/O bus during programmed I/O. The IOMD I/O data path consists of a latch in each direction, as shown in figure 4.12. The multiplexers are used during DMA operations to select the required byte or half-word to be output to the device. No ‘word packing’ occurs, except for sound DMA. I/O DMA operations are byte, half-word or word only. No byte steering is required for word DMA.

Old I/O space (0300 0000 to 0300 FFFF, 0303 0000 to 0303 FFFF and 0321 0000 to 033F FFFF inclusive) simulates old machines, in that the high 16 bits of the word must be written to when writing, and the low 16 bits must be read when reading. This is performed by the byte steering logic. When writing to the new I/O space, no byte steering occurs. This is because the new I/O space allows 32 bit transfers, and it is not possible to steer to an arbitrary byte in a 32 bit word, as only the lower 16 data bits of the I/O bus go through IOMD. The IOMD registers are not byte steered. Hence the byte steering logic is enabled for addresses in the range 0300 0000 to 0300 FFFF, 0303 0000 to 0303 FFFF and 0321 0000 to 033F FFFF inclusive only, and allows data through unmodified for all other programmed I/O locations.

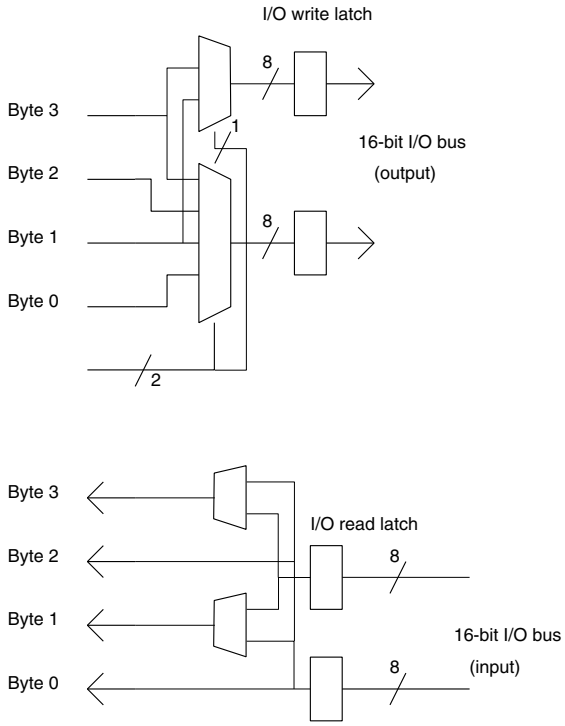

Figure 4.12 Byte steering data path logic

I/O data is latched in the I/O latches, which are internal to IOMD for the lower 16 bits, and are external latches for the upper 16 bits. As with current machines, it is possible for DMA to occur during long I/O accesses, however because of contention for the I/O bus, I/O and sound DMAs are not possible during processor I/O accesses.

Interrupt control logic and IOC timers

IOMD has a number of interrupt inputs which can be used to generate irq and fiq interrupts. In addition, there are internal interrupts generated by the DMA channels, the keyboard interface and the IOC timers 0 and 1. A set of registers similar to those in IOC allow the interrupt sources to be controlled by the processor. Some IOC register bits are unused, and some pin polarities are different to IOC. Refer to figure 4.13 for the register bit allocations, and to the table below for the interrupt active levels.

Npfiq	Podule fiq input. Level triggered, active low
Npirq	Podule irq input. Level triggered, active low
Nsintr	Serial interrupt input. Level triggered active low
Nscirq	SCSI interrupt. Level triggered active low
Nfintr	Floppy interrupt request. Level triggered, active low
Nindex	Floppy disc index input. Falling edge triggered
flyback	Flyback from VIDC, rising edge triggered
fdrq	Floppy drq interrupt. Level triggered, active high
pintr	Printer interrupt request. Rising edge triggered
Niext	Extended interrupt. Level triggered, active low

Table 4.2 Interrupt active polarities

There is a control register which provides access to a small number of I/O pins. These are used for the I2C interface, and for the ID chip interface. The ID pin is held low during reset, and then tri-stated. The I2C interface pins are tri-stated on reset. The registers are also shown in Figure 4.13. The functionality of these registers is similar to the corresponding registers in IOC, except it is not possible to generate a FIQ from the ID I/O bit, and not all the bits in the register are implemented. Refer to the IOC data sheet, and figure 4.13 for further detail.

Figure 4.13 I/O Control and Interrupt Registers

Two timers, the same as those in IOC are provided, which are clocked at the same 2MHz rate. Note that the IOC documentation is incorrect in stating that Tinterval = latch/2µS, it should say (latch+1)/2µS. Note also that if a latch command is issued immediately after a go command, the value latched may be the new value, but may be the old value of the counter. This also occurs in IOC, and should therefore not be a problem.

Mouse and keyboard interface

The mouse interface uses four pins to read quadrature information for X and Y mouse positions. Two 16-bit counters (MOUSEX and MOUSEY) are used to record the mouse movement and can be read by the processor. Clocking information is prevented from reaching the counters while the processor reads them, in such a way that no counts will be lost. Refer to 0197,256/FS 'Medusa Mouse Interface Hardware' for more details of the mouse interface.

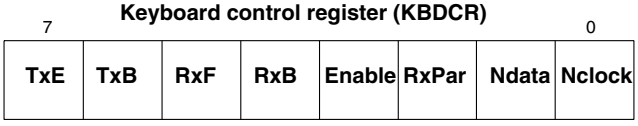

Figure 4.14 Keyboard control register

The keyboard interface uses two pins to communicate with a standard PC keyboard. The software interface is similar to the IOC keyboard interface and interrupts may be generated on character transmission and reception. Receive and transmit registers, each of 8 bits, are provided mapped to the same address. An 8-bit control register (KBDCR) provides access to the received parity bit (RxPar). Out-going parity is generated automatically. Direct access to the clock and data pins is provided in this register, via the Ndata and Nclock bits. Writing a 1 to one of these bits takes the corresponding pin low. In addition, four status bits may be read from the control register, for applications which are not interrupt driven. The TxE bit indicates that the transmit register is empty, and hence may be written. The RxF bit indicates that the receive register is full, and hence may be read. The TxB and RxB bits indicate that transmission or reception is in progress, though should not normally be needed when interrupts rather than polling is used with the keyboard interface, as will normally be the case. The Enable bit must be high for the interface to be enabled. When low, the keyboard state machine is held in its reset state. Refer to 0197,255/FS, 'Medusa Keyboard Interface Hardware' for more details of the keyboard interface.

ID and version registers

IOMD contains three byte-wide registers that allow the CPU to read the JTAG part number and version number. These registers are read only registers. Writing to them invokes various test modes that may cause the device to malfunction. See the register table for details of the addresses and data in each register.

Reset and power-on reset

IOMD has an active high power on reset input, with schmitt level input, and a bidirectional active low reset pin. The reset pin is driven low during power on reset, and may be pulled low at any time to reset the chip. The ID I/O bit is driven low during reset. The POR bit in the IRQA interrupt register is set on power on reset. The JTAG interface is reset by an internal power on reset cell in IOMD.

During reset, Nfiq becomes an input and taking it low invokes test features. Nfiq should be pulled up by an external pull-up resistor

Linear sound system

The linear sound system is based around a stereo sound CODEC (eg Analog Device's AD1848), and IOMD has a dedicated interface to this device. The interface consists of 5 pins:- Nsndcs, spdrq, scdrq, Nspdack, and Nscdack.

The sound CODEC is an 8-bit parallel interface peripheral connecting to the IOMD I/O bus. Data can be transferred to the device via programmed I/O or DMA. IOMD has two sound DMA channels for this purpose. The CODEC has two DMA channels, the capture channel and the playback channel. The sound capture channel can only be used for sound sampling. The sound playback channel can be configured for either playback or capture. The CODEC sound capture channel shares a DMA address generator with the VIDC sound playback channel, however since the CODEC can be configured to provide sound capture over its playback DMA channel, there is only limited restriction caused by this. It is thus not possible to have CODEC playback and capture at the same time as the VIDC sound system is used.

Nsndcs is the sound chip select signal, and is a general purpose chip select with programmable timing, the same as the SCSI and combo chip selects. The sound chip select timing is controlled by bits 5:4 in the IOTCR register. It is recommended that timing B is selected for normal operation with the CODEC. spdrq is the sound playback DMA request signal, and Nspdack the corresponding DMA acknowledge signal. scdrq is the sound capture DMA request, and Nscdack the DMA acknowledge for this.

The sound CODEC can be configured in a number of modes, and these in turn affect the number of bytes per sample that must be transferred. Mono 8 bit is 1 byte/sample. Stereo 8-bit or mono 16 bit is 2 bytes/sample, and stereo 16-bit is 4 bytes/sample. It is not possible to have simultaneous playback and capture with differing numbers of bytes/sample. In addition to the increment field in the DMA address generator, the IOTCR must be programmed with the number of bytes to be transferred to the CODEC per sample. Bits 7:6 of the IOTCR control the number of bytes transferred, as shown in figure 4.11. IOMD uses a burst DMA transfer to transfer all bytes of a sample. For each playback sample, the data to be transferred is read from memory, and stored in a latch in the funnel. Then each byte of the sample is output to the CODEC, least significant byte first in a single burst transfer. For each capture sample, each byte of the sample is read from the CODEC, and stored in a latch in the funnel, and then the complete sample is transferred to memory in a single access. The timing for reads and writes to the sound CODEC is the same. An example of a 4 byte/sample read is shown below.

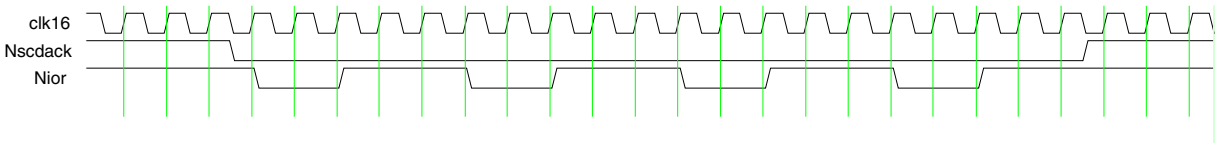

Figure 4.15 Sound DMA timing (4 bytes/sample, read)

Second processor support

IOMD has limited support for a second processor. This essentially consists of the Npreq pin. This is a signal similar to Nmreq, but with higher priority. In order to add a '486 second processor to a computer based on IOMD, it is necessary to make the '486 card emulate an ARM bus interface, by the use of some interface logic, that would normally be implemented in an interface ASIC. This ASIC must ensure that unaligned accesses are converted to ARM byte or word accesses, and make sequential accesses truly sequential (thus a '486 cache line fill would have to be modified if it does not start on a quad word boundary). The interface logic must also do all the arbitration between the second processor and the ARM, and thereby decide when to assert its Npreq signal.

The second processor interface logic must wait for a safe time, stop the ARM bus clock using the Nwait pin, and then perform an access, or burst of sequential accesses. The second processor interface bus controller arbitrates in MCLK high periods, and drives registered outputs from the falling edge of MCLK.

An additional I-cycle (ie. non-memory request cycle) is inserted between the ARM and second processor bus bursts. The second processor can only request the bus if it sees that the ARM is about to do an internal cycle. This first internal cycle is stretched to the ARM (with NWAIT) and merged with the next sequential bus cycle by the second processor with no cycle overhead, by removing ABE and driving second processor addesses onto the bus. ABE is the ARM bus enable signal, which output-enables the address and data buses, and relevant control lines. The second processor interface must also observe the ARM LOCK signal and must not request the bus in the internal cycle between the read and write cycles of a SWAP instruction.

When the second processor interface completes a transfer, or burst of sequential transfers, it removes its pipelined request, and on the following cycle removes the ABE signal. One cycle later still, it removes the NWAIT signal to the ARM. An internal cycle for bus turnaround is inserted whilst ABE is enabled, but the ARM clock is stopped by NWAIT, to ensure correct merged internal - sequential operation, should the ARM require the bus immediately.

If the second processor interface removes its request for only one MCLK period and then reasserts it, IOMD does not relinquish the bus to the ARM and thus 486-Locked cycles (for example) may be performed. If however more than one cycle of nPREQ removal occurs then the interface must relinquish the bus and wait for an ARM entering Internal cycle condition.

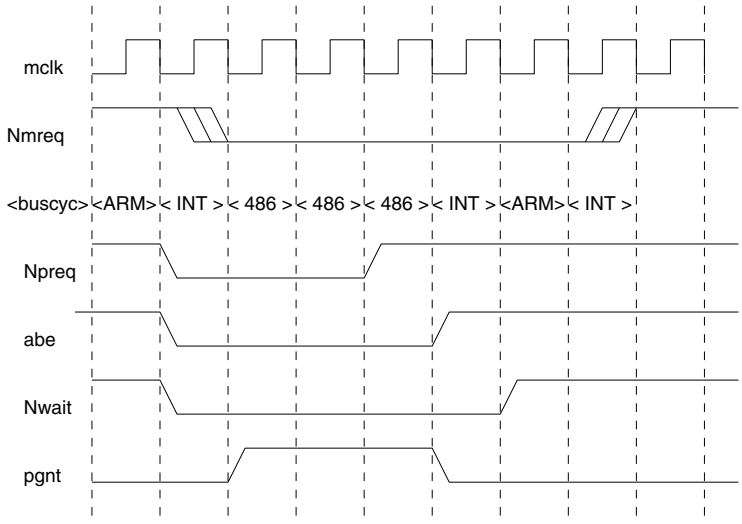

Figure 4.16 Second processor interface timing (assuming '486 second processor)

In the diagram above (figure 4.16), <buscyc> indicates the bus cycle currently being executed. The diagram shows a '486 second processor performing a two-word burst access. The first internal cycle, shown as <INT> in the diagram is the extra internal cycle added for the bus changeover. The first 486 cycle, shown by the <486> symbol would actually be the I-cycle of a merged I-S cycle, as IOMD only uses I-cycles and S-cycles. Therefore N-cycles are made from merged I-S cycles, as is done by the ARM3 for example. Hence there is an extra I-cycle inserted by IOMD during bus change-over. The pgnt signal shown in the diagram above is the mclk qualifier for the second processor, and thus is effectively active when the '486 second processor has the bus.

5. Medusa Memory Map

The Medusa physical memory map is shown below. For the purposes of DMA, the VRAM appears in the whole of the bottom half of the address space, as bit 28 of the DMA address selects between DRAM and VRAM in this implementation. The total memory map is 512MBytes, as only 29 address bits from the ARM are connected to IOMD. However, this does not prevent future machines having a larger physical memory map.

Physical address (MBytes)	**Address (Hex)**	**Contents**
0	0000 0000	Main ROM
16	0100 0000	Extension ROM
32	0200 0000	VRAM
48	0300 0000	I/O (including old podules, and IOMD registers)
64	0400 0000	Reserved
128	0800 0000	New 'extended address space' expansion cards
256	1000 0000	DRAM SIMM 0, bank 0
320	1400 0000	DRAM SIMM 0, bank 1
384	1800 0000	DRAM SIMM 1, bank 0
448	1C00 0000	DRAM SIMM 1, bank 1
512	2000 0000	End (reserved for more more DRAM in future implementations)

Since each SIMM may have one or two banks on it, and as each bank will usually be less than 64MB, the physical DRAM address map may be discontiguous.

The podule address space is the same as current machines. The extended expansion card address space allocates 16MB per card, and there may be up to 8 cards in total.

The I/O address space beginning at 0300 0000 is allocated as shown below. The area between 0300 0000 and 0320 0000 repeats 8 times. The area from 0320 0000 to 0340 0000 contains IOMD internal registers plus podules as in the current Archimedes memory map.

There are two types of peripheral access supported by IOMD. PROG address space is accessed with fixed timing and the access cycles are not divisible by DMA. True I/O address space accesses may be interrupted by DMA. The IOMD internal PROG address space is from 0320 0000 to 0324 0000 and this includes all IOMD internal registers. There is also an external PROG address space from 0340 0000 to 037F FFFF. Accesses to 0340 0000 to 035F FFFF activate the Nprog pin. Accesses to addresses from 0360 0000 to 037F FFFF do not activate the Nprog pin. This area is intended for the 486 card. Accesses to addresses in the range 0350 000 to 035F FFFF and 0370 0000 to 037F FFFF arbitrate for the memory bus on quad word boundaries, and do not arbitrate for the VIDC programming bus first. Accesses in the range 0340 0000 to 034F FFFF and 0360 0000 to 036F FFFF arbitrate for the memory bus on every accesses, and arbitrate for the VIDC data bus first. Thus VIDC should be programmed at location 0340 0000, as this activates the Nprog pin, and arbitrates for the VIDC programming bus at the start of the cycle.

True I/O address space is from 0300 0000 to 0400 0000 excluding those areas which are mapped to PROG space as detailed above. There is also another area of I/O space from 0800 0000 to 1000 0000. The I/O address space is broken down as shown below.

Accesses to SIO space in the S1 to S3 area are assumed to be internal peripherals by IOMD, and will not enable the podule buffer on a read, and accesses to SIO space in the S4 to S7 area are assumed to be external peripherals, and will thus enable the podule buffer on a read. The podule buffer is always output enabled when not reading from an on board device.

I/O Address (Hex)	Contents
0300 0000	Internal MEMC podules (asserts module select [Nms])
0301 0000	Combo chip select signal
0301 2000	Combo dack address space
0302 A000	Combo dack+TC address space
0302 B000	SCSI programmed I/O address space
0302 C000	CODEC sound chip select signal
0303 0000 to 0303 FFFF	External MEMC podules (asserts module select [Nms])
0320 0000	IOMD registers (internal prog area, does not assert Nprog pin)
0321 0000	Simple expansion cards and peripherals (asserts Nsio pin)
0340 0000	External prog space. Asserts Nprog pin. Arbitrates for video bus. Intended for VIDC register programming
0350 0000	External prog space. Asserts Nprog pin. Does not arbitrate for the video bus
0360 0000	External prog space. Does not assert Nprog pin. Arbitrates for video bus. Unlikely to be useful
0370 0000	External prog space. Does not assert Nprog pin. Does not arbitrate for video bus. Intended for '486 card.
0380 0000 to 0400 0000	Reserved (Action as for 0370 0000 - 037F FFFF)

6. IOMD registers

The following table lists the registers in IOMD. The base address of these registers is &0320 0000, the same as the address of the IOC internal registers in the current architecture. The size field indicates the width of the register, although some bits may not be used in a register.

Name	Address	Size	Read	Write	Function	Reset state
IOCR	0	(8)	Status	Control	I/O control	-1-
KBDDAT	4	(8)	Rx Data	Tx Data	Keyboard data	-x-
KBDCR	8	(8)	Status	Control	Keyboard control	-0-
IRQSTA	10	(8)	Status	---	IRQA status	
IRQRQA	14	(8)	Request	Clear	IRQA request/clear	-x-
IRQMSKA	18	(8)	Mask	Mask	IRQA mask	-0-
IRQSTB	20	(8)	Status	---	IRQB status	
IRQRQB	24	(8)	Request	---	IRQB request	
IRQMSKB	28	(8)	Mask	Mask	IRQB mask	-0-
FIQST	30	(8)	Status	---	FIQ status	
FIQRQ	34	(8)	Request	---	FIQ request	
FIQMSK	38	(8)	Mask	Mask	FIQ mask	-0-
T0LOW	40	(8)	Count Low	Latch Low	Timer 0 low bits	-x-
T0HIGH	44	(8)	Count High	Latch High	Timer 0 high bits	-x-
T0GO	48	(8)	---	Go command	Timer 0 Go command	-x-
T0LAT	4C	(8)	---	Latch command	Timer 0 Latch cmd	-x-
T1LOW	50	(8)	Count Low	Latch Low	Timer 1 low bits	-x-
T1HIGH	54	(8)	Count High	Latch High	Timer 1 high bits	-x-
T1GO	58	(8)	---	Go command	Timer 1 Go command	-x-
T1LAT	5C	(8)	---	Latch command	Timer 1 Latch cmd	-x-
ROMCR0	80	(8)	RomCr0	RomCr0	ROM control bank 0	-0-
ROMCR1	84	(8)	RomCr1	RomCr1	ROM control bank 1	-0-
DRAMCR	88	(8)	DramCr	DramCr	DRAM control	-0-
VREFCR	8C	(8)	VrefCr	VrefCr	VRAM & refresh control	-0-
FSIZE	90	(8)	Size	Size	Flyback line size	-x-
ID0	94	(8)	ID0	---	Chip ID no. low byte	
ID1	98	(8)	ID1	---	Chip ID no. high byte	
VERSION	9C	(8)	Version	---	Chip version number	
MOUSEX	A0	(16)	MouseX	MouseX	Mouse X position	-x-
MOUSEY	A4	(16)	MouseY	MouseY	Mouse Y position	-x-
DMATCR	C0	(8)	DmaTcr	DmaTcr	DACK timing control	-x-
IOTCR	C4	(8)	IoTcr	IoTcr	I/O timing control	-x-
ECTCR	C8	(8)	EcTcr	EcTcr	Expansion card timing	-x-
DMAEXT	CC	(8)	DmaExt	DmaExt	DMA external control	-x-

Name	Address	Size	Read	Write	Function	Reset state
IO0CURA	100	(32)	IO0CurA	IO0CurA	I/O DMA 0 CurA	-x-
IO0ENDA	104	(32)	IO0EndA	IO0EndA	I/O DMA 0 EndA	-x-
IO0CURB	108	(32)	IO0CurB	IO0CurB	I/O DMA 0 CurB	-x-
IO0ENDB	10C	(32)	IO0EndB	IO0EndB	I/O DMA 0 EndB	-x-
IO0CR	110	(8)	IO0Control	IO0Control	I/O DMA 0 Control	x0000000
IO0ST	114	(8)	IO0Status	---	I/O DMA 0 Status	xxxxx110
IO1CURA	120	(32)	IO1CurA	IO1CurA	I/O DMA 1 CurA	
IO1ENDA	124	(32)	IO1EndA	IO1EndA	I/O DMA 1 EndA	
IO1CURB	128	(32)	IO1CurB	IO1CurB	I/O DMA 1 CurB	as I/O 0
IO1ENDB	12C	(32)	IO1EndB	IO1EndB	I/O DMA 1 EndB	
IO1CR	130	(8)	IO1Control	IO1Control	I/O DMA 1 Control	
IO1ST	134	(8)	IO1Status	---	I/O DMA 1 Status	
IO2CURA	140	(32)	IO2CurA	IO2CurA	I/O DMA 2 CurA	
IO2ENDA	144	(32)	IO2EndA	IO2EndA	I/O DMA 2 EndA	
IO2CURB	148	(32)	IO2CurB	IO2CurB	I/O DMA 2 CurB	as I/O 0
IO2ENDB	14C	(32)	IO2EndB	IO2EndB	I/O DMA 2 EndB	
IO2CR	150	(8)	IO2Control	IO2Control	I/O DMA 2 Control	
IO2ST	154	(8)	IO2Status	---	I/O DMA 2 Status	
IO3CURA	160	(32)	IO3CurA	IO3CurA	I/O DMA 3 CurA	
IO3ENDA	164	(32)	IO3EndA	IO3EndA	I/O DMA 3 EndA	
IO3CURB	168	(32)	IO3CurB	IO3CurB	I/O DMA 3 CurB	as I/O 0
IO3ENDB	16C	(32)	IO3EndB	IO3EndB	I/O DMA 3 EndB	
IO3CR	170	(8)	IO3Control	IO3Control	I/O DMA 3 Control	
IO3ST	174	(8)	IO3Status	---	I/O DMA 3 Status	
SD0CURA	180	(32)	SD0CurA	SD0CurA	Sound DMA 0 CurA	
SD0ENDA	184	(32)	SD0EndA	SD0EndA	Sound DMA 0 EndA	
SD0CURB	188	(32)	SD0CurB	SD0CurB	Sound DMA 0 CurB	as I/O 0
SD0ENDB	18C	(32)	SD0EndB	SD0EndB	Sound DMA 0 EndB	
SD0CR	190	(8)	SD0Control	SD0Control	Sound DMA 0 Control	
SD0ST	194	(8)	SD0Status	---	Sound DMA 0 Status	
SD1CURA	1A0	(32)	SD1CurA	SD1CurA	Sound DMA 1 CurA	
SD1ENDA	1A4	(32)	SD1EndA	SD1EndA	Sound DMA 1 EndA	
SD1CURB	1A8	(32)	SD1CurB	SD1CurB	Sound DMA 1 CurB	as I/O 0
SD1ENDB	1AC	(32)	SD1EndB	SD1EndB	Sound DMA 1 EndB	
SD1CR	1B0	(8)	SD1Control	SD1Control	Sound DMA 1 Control	
SD1ST	1B4	(8)	SD1Status	---	Sound DMA 1 Status	
CURSCUR	1C0	(32)	CursCur	CursCur	Cursor DMA Current	-x-
CURSINIT	1C4	(32)	CursInit	CursInit	Cursor DMA Init	-x-
VIDCUR	1D0	(32)	VIDCur	VIDCur	Video DMA Current	-x-
VIDEND	1D4	(32)	VIDEnd	VIDEnd	Video DMA End	-x-
VIDSTART	1D8	(32)	VIDStart	VIDStart	Video DMA Start	-x-
VIDINIT	1DC	(32)	VIDInit	VIDInit	Video DMA Init	-x-
VIDCR	1E0	(8)	VIDControl	VIDControl	Video DMA Control	xx000000
DMAST	1F0	(8)	Status	---	DMA interrupt status	
DMARQ	1F4	(8)	Request	---	DMA interrupt request	
DMAMSK	1F8	(8)	Mask	Mask	DMA interrupt mask	-0-

7. DMA Latency

The DMA priority scheme is as shown in the section on DMA channels. The four I/O channels operate on a round-robin basis. All latencies are measured from the time a request is made to the time at which the transfer is complete. All numbers are in nS unless stated otherwise.

The state machine which controls bus activity has a number of points at which arbitration for the bus takes place. Arbitration always takes place while the processor performs an idle cycle and before it starts a memory access. In addition, arbitration occurs during sequential memory accesses as follows:-

DRAM/VRAM - sequential bursts of up to 8 words can occur uninterrupted by DMA. In the absence of DMA requests these bursts can extend to 256 words.

ROM - uninterrupted burst accesses of up to 4 words can occur. If non-burst ROM is used, arbitration occurs on every access.

PROG - uninterrupted bursts of up to 4 words can occur where the access is to an IOMD internal register. Where the access is to the VIDC control register, arbitration occurs on every word.

I/O - an IORQ/IOGT mechanism is used for programmed I/O and (non-I/O) DMA can occur during I/O cycles.

Cursor

Requested by Nvidrq from VIDC20 while vNc (also from VIDC20) is low indicating that cursor data is required. Cursor transfers occur at the end of line and during HSYNC. At this time, we know that no video SAM transfers can occur.

Criticality is that transfer must complete before vNc goes high again. This is a programmable quantity and can be configured for the value obtained here. Repetition is once every other line and hence unlikely to be below 30uS.

Synchronise Nvidrq (2 * rclk in slow ROM)	500
Wait for longest memory access (3 * ROM burst)	375
Time for DMA to complete (N+3S)	350
Total to finish	1225ns

VIDC Sound

Requested by Nsndrq from VIDC20. Criticality is 3uS which is the time in which a single byte of sound data is consumed by VIDC at the highest sample rate. Repetition is much slower than this as 16 bytes are fetched on each DMA. Fastest repetition is therefore 48uS.

Sound requests can occur at the same time as cursor requests and so the latency is that of the cursor DMA plus the time to do the transfer. VIDC sound can also occur at the same time as SAM video data is clocked from the VRAM. However, cursor and SAM video data cannot occur at the same time. Therefore there are two cases - when video is active, and data is being transferred from the SAM, and when video is inactive, but a cursor request could occur.

The longest SAM transfer adds 28 words to the VIDC FIFO, whilst VIDC is removing data at 160MB/sec. In addition

there is synchronisation time to move the VIDC20 clock from rclk to vclk and back to rclk. Worst case is around 2uS.

The other case, which is the worst case, is where video SAM data is not active, but cursor data is being requested.

Perform Cursor DMA	1225
Synchronise to vclk, SAM transfers, sync to rclk	687
Time for DMA to complete (N+3S)	350
	2262

Video Transfer

This DMA does not move data via the data bus, but initiates a SAM copy in the VRAM. The criticality is around 11uS which is the time to empty a half SAM at the highest VIDC clocking rate. The repetition rate is the same.

Synchronise qsf (2 * rclk in slow ROM)	500
Wait for longest memory access (3 * ROM burst)	375
Perform VIDC sound DMA (687+350)	1037
Perform VIDC cursor DMA (350)	350
Time for DMA to complete (N cycle)	156
	2418

Codec Sound

Single word DMA initiated by the external codec. Criticality and repetition rate are 21uS, determined by the highest sampling rate which is 48kHz. All of the above DMAs can stack up before this one is serviced. The CODEC must also arbitrate for the I/O bus, however the memory bus latency is worse than the I/O bus latency, and I/O and memory bus usage is decoupled.

Perform video transfer DMA & all others	2418
Time for DMA to complete (N cycle)	156
	2574ns

DRAM Refresh

Not a DMA but treated like one. Criticality and repetition rate are programmable and the worst case is 16uS for both. All of the above DMAs could stack up in front of a refresh.

Perform codec sound DMA & all others	2574
Time for refresh to complete (6 * clk32)	187
	2761ns

I/O DMA

I/O DMA has latency aspects from both the memory bus and the I/O bus, however the I/O bus latency dominates.

Perform 3 slow I/O DMAs	1688
Perform 2 sound DMAs	1875
Finish slow I/O access	2000
Perform slow I/O DMA	562.5
	6128nS

ARM Latency

The ARM latency depends upon the memory bus only, not on the I/O bus latency, unless the ARM is accessing the I/O bus. Thus, assuming the ARM is not accessing the I/O bus :

Time for all other DMA's, excluding I/O DMA	2761
Time to perform 1 I/O memory DMA	156
Time to perform 2 CODEC memory DMA's	312
	3229ns

8. Boundary scan test interface

The boundary-scan interface conforms to the IEEE Std. 1149.1- 1990, Standard Test Access Port and Boundary-Scan Architecture (please refer to this document for an explanation of the terms used in this section and for a description of the TAP controller states.)

Overview

The boundary-scan interface provides a means of testing the core of the device when it is fitted to a circuit board, and a means of driving and sampling all the external pins of the device irrespective of the core state. This latter function permits testing of both the device's electrical connections to the circuit board, and (in conjunction with other devices on the circuit board having a similar interface) testing the integrity of the circuit board connections between devices. The interface intercepts all external connections within the device, and each such "cell" is then connected together to form a serial register (the boundary scan register). The whole interface is controlled via 4 dedicated pins: **TDI**, **TMS**, **TCK**, and **TDO.**

Block diagram

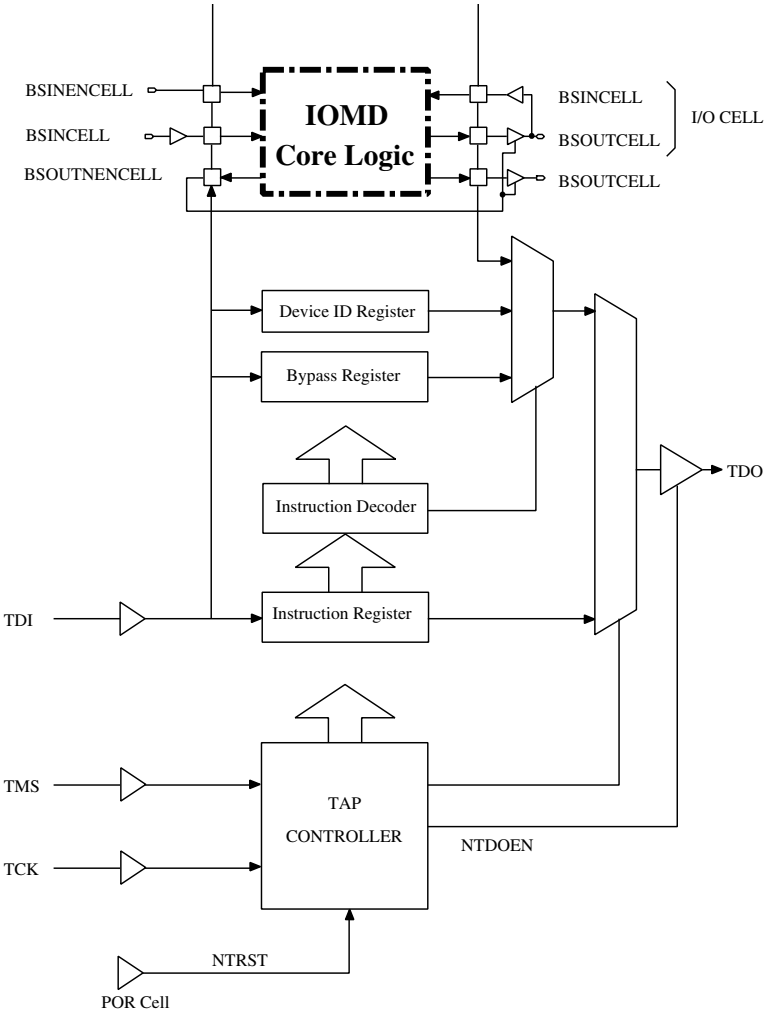

Figure 7.1 Boundary scan block diagram

Reset

The boundary-scan interface includes a state-machine controller (the TAP controller). The state machine is automatically reset on power-up by the power on reset cell.

The action of reset is as follows:

- System mode is selected (ie the boundary scan chain does NOT intercept any of the signals passing between the pads and the core).
- IDcode mode is selected. If TCK is pulsed, the contents of the ID register will be clocked out ofTDO.

Pullup resistors

The 1149.1 standard effectively requires that **TDI,** and **TMS** should have internal pullup resistors. IOMD includes these resistors.

Instruction register

The instruction register is 4 bits in length. There is no parity bit.

The fixed value loaded into the instruction register during the *CAPTURE-IR* controller state is: 0001

Public instructions

The following public instructions are supported:

INSTRUCTION	BINARY CODE
BYPASS	1111
SAMPLE/PRELOAD	0011
EXTEST	0000
INTEST	1100
IDCODE	1110
HIGHZ	0111
CLAMP	0101
CLAMPZ	1001

In the descriptions that follow, **TDI** and **TMS** are sampled on the rising edge of **TCK** and all output transitions on **TDO** occur as a result of the falling edge of **TCK**.

BYPASS (1111)

The BYPASS instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO**.

When the BYPASS instruction is loaded into the instruction register, all the boundary-scan cells are placed in their normal (system) mode of operation. This instruction has no effect on the system pins.

In the *CAPTURE-DR* state, a logic 0 is captured by the bypass register. In the *SHIFT-DR* state, test data is shifted into the

bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero. The bypass register is not affected in the *UPDATE-DR* state.

SAMPLE/PRELOAD (0011)

The BS (boundary-scan) register is placed in test mode by the SAMPLE/PRELOAD instruction.

The SAMPLE/PRELOAD instruction connects the BS register between **TDI** and **TDO**.

When the instruction register is loaded with the SAMPLE/PRELOAD instruction,
all the boundary-scan cells are placed in their normal system mode of operation.

In the *CAPTURE-DR* state, a snapshot of the signals at the boundary-scan cells is taken on the rising edge of **TCK**. Normal system operation is unaffected. In the *SHIFT-DR* state, the sampled test data is shifted out of the BS register via the **TDO** pin, whilst new data is shifted in via the **TDI** pin to preload the BS register parallel input latch. In the *UPDATE-DR* state, the preloaded data is transferred into the BS register parallel output latch. Note that this data is not applied to the system logic or system pins while the SAMPLE/PRELOAD instruction is active. This instruction should be used to preload the boundary-scan register with known data prior to selecting the INTEST or EXTEST instructions (see the table below for appropriate guard values to be used for each boundary-scan cell).

EXTEST (0000)

The BS (boundary-scan) register is placed in test mode by the EXTEST instruction.

The EXTEST instruction connects the BS register between **TDI** and **TDO**.

When the instruction register is loaded with the EXTEST instruction, all the
boundary-scan cells are placed in their test mode of operation.

In the *CAPTURE-DR* state, inputs from the system pins and outputs from the boundary-scan output cells to the system pins are captured by the boundary-scan cells. In the *SHIFT-DR* state, the previously captured test data is shifted out of the BS register via the **TDO** pin, whilst new test data is shifted in via the **TDI** pin to the BS register parallel input latch. In the *UPDATE-DR* state, the new test data is transferred into the BS register parallel output latch. Note that this data is applied immediately to the system logic and system pins. The first EXTEST vector should be clocked into the boundary-scan register, using the SAMPLE/PRELOAD instruction, prior to selecting INTEST to ensure that known data is applied to the system logic.

INTEST (1100)

The BS (boundary-scan) register is placed in test mode by the INTEST instruction.

The INTEST instruction connects the BS register between **TDI** and **TDO.**

When the instruction register is loaded with the INTEST instruction, all the boundary-scan cells are placed in their test mode of operation.

In the *CAPTURE-DR* state, the inverse of the data supplied to the core logic from input boundary-scan cells is captured, while the true value of the data that is output from the core logic to output boundary-scan cells is captured. In the *SHIFT-DR* state, the previously captured test data is shifted out of the BS register via the **TDO** pin, whilst new test data is shifted in via the **TDI** pin to the BS register parallel input latch. In the *UPDATE-DR* state, the new test data is transferred into the BS register parallel output latch. Note that this data is applied immediately to the system logic and system pins. The first INTEST vector should be clocked into the boundary-scan register, using the SAMPLE/PRELOAD instruction, prior to

selecting INTEST to ensure that known data is applied to the system logic.

Single-step operation is possible using the INTEST instruction.

IDCODE (1110)

The IDCODE instruction connects the device identification register (or ID register) between **TDI** and **TDO**. The ID register is a 32-bit register that allows the manufacturer, part number and version of a component to be determined through the TAP.

When the instruction register is loaded with the IDCODE instruction, all the boundary-scan cells are placed in their normal (system) mode of operation.

In the *CAPTURE-DR* state, the device identification code (specified at the end of this section) is captured by the ID register. In the *SHIFT-DR* state, the previously captured device identification code is shifted out of the ID register via the **TDO** pin,whilst data is shifted in via the **TDI** pin into the ID register. In the *UPDATE-DR* state, the ID register is unaffected.

HIGHZ (0111)

The HI-Z instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO.**

When the HI-Z instruction is loaded into the instruction register, all outputs are placed in an inactive drive state.

In the *CAPTURE-DR* state, a logic 0 is captured by the bypass register. In the *SHIFT-DR* state, test data is shifted into the bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero. The bypass register is not affected in the *UPDATE-DR* state.

CLAMP (0101)

The CLAMP instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO.**

When the CLAMP instruction is loaded into the instruction register, the state of all output signals is defined by the values previously loaded into the boundary-scan register. A guarding pattern (specified for this device at the end of this section) should be pre-loaded into the boundary-scan register using the SAMPLE/PRELOAD instruction prior to selecting the CLAMP instruction.

In the *CAPTURE-DR* state, a logic 0 is captured by the bypass register. In the *SHIFT-DR* state, test data is shifted into the bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero. The bypass register is not affected in the *UPDATE-DR* state.

CLAMPZ (1001)

The CLAMPZ instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO**.

When the CLAMPZ instruction is loaded into the instruction register, all outputs are placed in an inactive drive state, but the data supplied to the disabled output drivers is derived from the boundary-scan cells. The purpose of this instruction is to ensure, during production testing, that each output driver can be disabled when its data input is either a 0 or a 1.

In the *CAPTURE-DR* state, a logic 0 is captured by the bypass register. In the *SHIFT-DR* state, test data is shifted into the bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero.

The bypass register is not affected in the *UPDATE-DR* state.

Test data registers

Bypass Register

Purpose: This is a single bit register which can be selected as the path between **TDI** and **TDO** to allow the device to be bypassed during boundary-scan testing.

Length: 1 bit

Operating Mode: When the BYPASS instruction is the current instruction in the instruction register, serial data is transferred from **TDI** to **TDO** in the *SHIFT-DR* state with a delay of one **TCK** cycle.

There is no parallel output from the bypass register.

A logic '0' is loaded from the parallel input of the bypass register in the *CAPTURE-DR* state.

IOMD Device identification (ID) code register

Purpose: This register is used to read the 32-bit device identification code. No programmable supplementary identification code is provided.

Length: 32 bits

The format of the ID register is as follows:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
version				part number																manufacturer identity											
																															1

In hexadecimal : `IOMD fabricated by GPS : 1D4E706F`

The ID is also reflected in the IOMD status registers, in locations 094 to 09C inclusive (see section 6). Location 094 contains the low byte of the part number. Location 098 contains the high byte of the part number. Location 09C contains the version number. The manufacturer number is not reflected in the IOMD status registers. These registers must not be written to, as writing to these registers invokes various test modes and features, that may cause the device to behave in an unpredictable manner.

Operating Mode: When the IDCODE instruction is current, the ID register is selected as the serial path between **TDI** and **TDO**.

There is no parallel output from the ID register.

The 32-bit device identification code is loaded into the ID register from its parallel inputs during the *CAPTURE-DR* state.

IOMD Boundary scan (BS) register

Purpose: The BS register consists of a serially connected set of cells around the periphery of the device, at the interface between the core logic and the system input/output pads. This register can be used to isolate the core logic from the pins and then apply tests to the core logic, or conversely to isolate the pins from the core logic and then drive or monitor the system pins.

Operating modes: The BS register is selected as the register to be connected between **TDI** and **TDO** only during the SAMPLE/PRELOAD, EXTEST and INTEST instructions. Values in the BS register are used, but are not changed, during the CLAMP and CLAMPZ instructions.

In the normal (system) mode of operation, straight-through connections between the core logic and pins are maintained and normal system operation is unaffected.

In TEST mode (i.e. when either EXTEST or INTEST is the currently selected instruction), values can be applied to the core logic or output pins independently of the actual values on the input pins and core logic outputs respectively. Additional boundary-scan cells are interposed in the scan chain in order to control the enabling of tristateable buses.

The correspondence between boundary-scan cells and system pins, system direction controls and system output enables is **(will be)** as shown on the following page. The cells are listed in the order in which they are connected in the boundary-scan register, starting with the cell closest to **nTDI**. All boundary-scan register cells at input pins can apply tests to the on-chip core logic. The EXTEST guard values specified in the table below should be clocked into the boundary-scan register (using the SAMPLE/PRELOAD instruction) before the EXTEST instruction is selected, to ensure that known data is applied to the core logic during the test. The INTEST guard values shown in the table below should be clocked into the boundary-scan register (using the SAMPLE/PRELOAD instruction) before the INTEST instruction is selected to ensure that all outputs are disabled. These guard values should also be used when new EXTEST or INTEST vectors are clocked into the boundary-scan register.

Output enable boundary-scan cells

To Be Determined

Single-step operation

IOMD is a static design and there is no minimum clock speed. It can therefore be single-stepped while the INTEST instruction is selected. This can be achieved by serialising a parallel stimulus and clocking the resulting serial vectors into the boundary-scan register. When the boundary-scan register is updated, new test stimuli are applied to the core logic inputs; the effect of these stimuli can then be observed on the core logic outputs by capturing them in the boundary-scan register.

9. Package and pinout

IOMD is packaged in a 208 pin QFP package, with a 0.5mm pin pitch. Pin 1 is at the top left corner, and pin numbering proceeds anti-clockwise, as shown in the diagram 9.1 below.

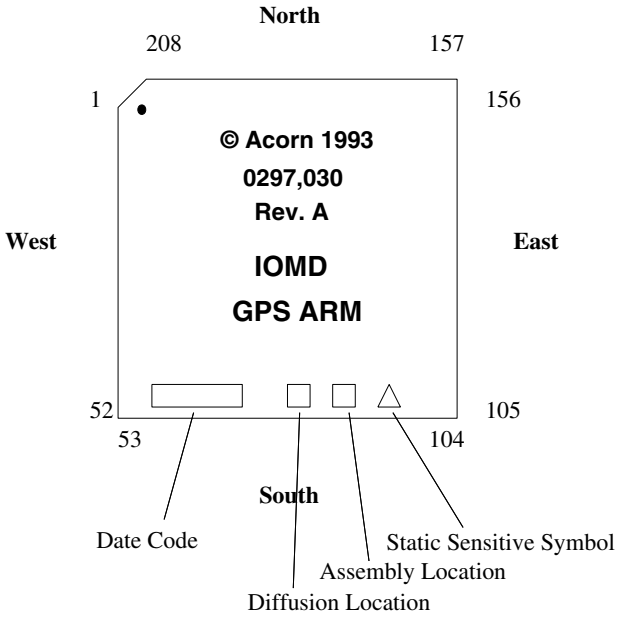

Figure 9.1 Pin location diagram for 208 SQFP package

The Packaging must have the following markings -

1. Part Number in the following format - 0297,030
2. A Revision Level - located under part number
3. Name of device - IOMD
4. Manufacturers name or Logo
5. Acorn name or logo with copyright symbol and year
6. Pin 1 identification such as a dot
7. Date Code

The Revision Level is the control on the device and reflects the build standard.

If any changes are made to the build standard of the device , then the Revision Level must be updated and Acorn must be informed of, and agree to, any alterations.

The Revision level of the device is 'A'.

The Manufacturer can add any further information such as Assembly Location etc, as long as the markings remain of a suitable size and are legible.

West			South			East			North		
1	:	XNfiq	53	:	XNpboe	105	:	XNvras	157	:	XNsccs
2	:	XNirq	54	:	Xlrnw	106	:	Xsc	158	:	Xtc
3	:	Xrclk	55	:	Vss	107	:	Xpclk	159	:	XNcdack
4	:	Vss	56	:	Xbd[0]	108	:	Vss	160	:	Vss
5	:	XNmreq	57	:	Xbd[1]	109	:	Xqsf	161	:	Xkdata
6	:	XNpreq	58	:	Xbd[2]	110	:	Xflybk	162	:	Xkclk
7	:	Xa[0]	59	:	Xbd[3]	111	:	Xvnc	163	:	XNscirq
8	:	Xa[1]	60	:	Xbd[4]	112	:	XNsndrq	164	:	XNsintr
9	:	Xa[2]	61	:	Xbd[5]	113	:	XNsndak	165	:	XNfintr
10	:	Xa[3]	62	:	Xbd[6]	114	:	XNvidrq	166	:	Xpintr
11	:	Xa[4]	63	:	Xbd[7]	115	:	XNvidak	167	:	Xfdrq
12	:	Xa[5]	64	:	Xbd[8]	116	:	XNcdoe	168	:	XNiext
13	:	Xa[6]	65	:	Xbd[9]	117	:	XNprog	169	:	Xd[0]
14	:	Xa[7]	66	:	Xbd[10]	118	:	Xra[0]	170	:	Xd[1]
15	:	Xa[8]	67	:	Xbd[11]	119	:	Xra[1]	171	:	Xd[2]
16	:	Xa[9]	68	:	Xclk16	120	:	Xra[2]	172	:	Xd[3]
17	:	Vss	69	:	Vss	121	:	Vss	173	:	Vss
18	:	Vdd	70	:	Vdd	122	:	Vdd	174	:	Vdd
19	:	Xa[10]	71	:	Xref8m	123	:	Xra[3]	175	:	Xd[4]
20	:	Xa[11]	72	:	Xbd[12]	124	:	Xra[4]	176	:	Xd[5]
21	:	Xa[12]	73	:	Xbd[13]	125	:	Xra[5]	177	:	Xd[6]
22	:	Xa[13]	74	:	Xbd[14]	126	:	Xra[6]	178	:	Xd[7]
23	:	Xa[14]	75	:	Xbd[15]	127	:	Xra[7]	179	:	Xd[8]
24	:	Xa[15]	76	:	XNeasis	128	:	Xra[8]	180	:	Xd[9]
25	:	Xa[16]	77	:	XNiorq	129	:	Xra[9]	181	:	Xd[10]
26	:	Xa[17]	78	:	XNiogt	130	:	Vss	182	:	Xd[11]
27	:	Xa[18]	79	:	XNbl	131	:	Xra[10]	183	:	Xd[12]
28	:	Xa[19]	80	:	XNindex	132	:	Xra[11]	184	:	Xd[13]
29	:	Xa[20]	81	:	Xdreq[0]	133	:	XNras[3]	185	:	Xd[14]
30	:	Xa[21]	82	:	Xdreq[1]	134	:	XNras[2]	186	:	Xd[15]
31	:	Xa[22]	83	:	Xdreq[2]	135	:	XNwe[1]	187	:	Xd[16]
32	:	Xa[23]	84	:	Xdreq[3]	136	:	XNras[1]	188	:	Xd[17]
33	:	Xa[24]	85	;	XNior	137	:	XNras[0]	189	:	Xd[18]
34	:	Xa[25]	86	:	XNiow	138	:	XNwe[0]	190	:	Xd[19]
35	:	Vdd	87	:	Vdd	139	:	Vdd	191	:	Vdd
36	:	Vss	88	:	Vss	140	:	Vss	192	:	Vss
37	:	Xa[26]	89	:	Xclk64	141	:	XNcas[0]	193	:	Xd[20]
38	:	Xa[27]	90	:	Xpor	142	:	XNcas[1]	194	:	Xd[21]
39	:	Xa[28]	91	:	XNreset	143	:	XNcas[2]	195	:	Xd[22]
40	:	XNrw	92	:	XNms	144	:	XNcas[3]	196	:	Xd[23]
41	:	XNbw	93	:	Xready	145	:	Xspdrq	197	:	Xd[24]
42	:	Xiicd	94	:	Xid	146	:	Xscdrq	198	:	Xd[25]
43	:	Xiicc	95	:	XNdack[0]	147	:	Xmsy[1]	199	:	Xd[26]
44	:	Xtdi	96	:	XNdack[1]	148	:	Xmsy[0]	200	:	Xd[27]
45	:	Xtms	97	:	XNdack[2]	149	:	Xmsx[1]	201	:	Xd[28]
46	:	Xtck	98	:	XNdack[3]	150	:	Xmsx[0]	202	:	Xd[29]
47	:	Xtdo	99	:	Vss	151	:	XNpirq	203	:	Xd[30]
48	:	XNwbe	100	:	XNsio	152	:	XNpfiq	204	:	Xd[31]
49	:	XNrbe	101	:	XNdt[0]	153	:	XNspdack	205	:	XNromcs
50	:	XNblw	102	:	XNdt[1]	154	:	XNscdack	206	:	Vss
51	:	XNblr	103	:	XNse	155	:	XNsndcs	207	:	Xmclk
52	:	Xclk2	104	:	Xdsf	156	:	XNccs	208	:	Xdbe