6. Functional Description

6.1 Introduction

6.1.1 Access Times

A number of devices appear in the processor memory map:

- Dynamic Random Access Memory (DRAM)
- Read Only Memory (ROM)
- Input/Output Controllers
- Video Controller
- MEMC internal registers:
  - (i) MEMC Control Register
  - (ii) DMA Address Generators
  - (iii) Logical to Physical Address Translator

These devices have very different access times, ranging from 125ns for DRAMs in page-mode to over 1µs for handshake driven IO cycles. MEMC synchronises the processor with the device it is accessing by stretching the processor clocks, **PH1** and **PH2**.

The processor is the default user of the data bus and memory. However, DMA (Direct Memory Access) and refresh operations require control of the DRAM and data bus, so MEMC disables the processor temporarily during these operations by tri-stating the processor data bus (using **DBE**), and holding the processor clocks.

6.1.2 N-cycles and S-cycles

MEMC uses the *page mode access* capability of DRAMs, where, once a row address has been strobed into the DRAM, any column in that row may be accessed merely by strobing in the new column address.

This facility is used whenever a number of sequential addresses in the DRAM are to be accessed (either by the processor or during a DMA operation). The first memory access in the sequence is a Non-sequential memory cycle (where both the row and column addresses are strobed to the DRAMs). The subsequent memory accesses are Sequential memory cycles (where the previous row address is held, and only the column address is strobed to the DRAMs).

Sequential memory cycles may also be used with nibble-mode ROMs, where the access time is reduced if only the low order address bits are changed.

The terms *N-cycle* and *S-cycle* refer to Non-sequential and Sequential accesses in DRAM or nibble-mode ROM.

6.2 Processor (ARM) Interface

6.2.1 Processor cycles

There are two basic types of processor operation:

(i) Memory access cycles

Where the processor accesses a device in its address space.

(ii) Internal cycles

Where the processor is performing an internal operation, and so does not access any external devices.

6.2.2 Processor signals

- Address bus {A[0:25]}

  The processor address bus is decoded by MEMC to give the processor access to the various devices.

  Much of the processor memory map is only accessible while MEMC is in Supervisor mode (which is selected by taking the SPVMD line HIGH).

- Memory request { $\overline{\text{MREQ}}$}

  This signal determines whether the **next** processor cycle will be a memory access or internal cycle.

- Not-Read/Write { $\overline{\text{R}}$/W}

  Determines the direction of data flow during processor memory access cycles. This signal is ignored during processor internal cycles.

- Not-Byte/Word { $\overline{\text{B}}$/W}

  Selects a byte (8-bit) or word (32-bit) data transfer during processor memory access cycles. A byte access to Physically and Logically Mapped RAM only enables the appropriate 8-bit *block* of DRAMs. ROM accesses always return a word quantity, regardless of the state of $\overline{\text{B}}$/W. This signal is ignored during processor internal cycles.

- Sequential access {SEQ}

  A HIGH on this line indicates that the processor will generate a sequential address during its **next** cycle. MEMC uses SEQ to determine whether a fast S-cycle may be used during the next DRAM or nibble-mode ROM access.

- Supervisor mode {SPVMD}

  A HIGH on this line puts MEMC into Supervisor mode, allowing the processor to access restricted areas of the memory map.

6.2.3 Processor controls

- Processor clocks {**PH1, PH2**}

  MEMC provides the processor with two non-overlapping clocks, **PH1** and **PH2**. Memory access cycles are nominally 250ns long, (with **PH1** HIGH for approx 175ns, and **PH2** HIGH for approx 55ns), but the following exceptions apply:

  (i) Sequential DRAM & nibble-mode ROM accesses

  DRAM and nibble-mode ROM S-cycles only take 125ns to complete, so the processor is given an 8MHz clock while S-cycles are in progress (see Section 8.4).

  (ii) ROM accesses

  ROM access times vary from 450ns to 200ns. **PH1** is held HIGH long enough to meet the ROM access time requirement. (see Section 8.5)

  (iii) IO cycles

  IO cycles take a variable length of time to complete. During the longer IO cycles, the processor is suspended by holding **PH2** HIGH until the I/O Controller is ready to complete the cycle. Suspending the processor during its **PH2** phase allows the IO cycle to be completed quickly when the I/O Controller is ready.

  (iv) DMA and refresh operations

  DMA and refresh operations have priority over most processor memory access cycles (S-cycles are uninterruptable). A processor memory access is suspended during DMA and refresh operations by disabling the processor data bus drivers (using **DBE**), and holding the **PH1** clock HIGH until the operation has finished. The processor then continues with its delayed memory access (unless another DMA/refresh operation is pending).

  DMA and refresh operations may also occur during long IO cycles. In this case, the IO cycle is delayed until the DMA/refresh operation completes.

  Internal cycles are always 125ns long, with **PH1** and **PH2** each HIGH for approx 55nS.

- Data bus enable {**DBE**}

  Enables the processor data bus during processor write cycles. This signal may also be inverted externally, and used as a DRAM write enable signal.

- Memory access abort {**ABORT**}

  Warns the processor that the requested access is illegal (either because an attempt was made to access a protected address while MEMC was in an insufficiently privileged mode, or an access to a non-existent Logical page was attempted).

6.3 Dynamic RAM (DRAM) Interface

MEMC interfaces directly to most standard Dynamic RAMs, providing a 10-bit multiplexed RAM address bus, RA[0:9], a row address strobe, $\overline{\text{RAS}}$, and a set of four column address strobes, $\overline{\text{CAS}}$[0:3].

6.3.1 Byte and Word accesses

The DRAM is divided into four blocks of eight bits, with $\overline{\text{CAS}}$[0], $\overline{\text{CAS}}$[1], $\overline{\text{CAS}}$[2] and $\overline{\text{CAS}}$[3] each controlling a block ($\overline{\text{CAS}}$[0] is the least significant block, and $\overline{\text{CAS}}$[3] is the most significant).

The processor Byte/Word select line, $\overline{\text{B}}$/W, selects whether a whole word (32-bits) or a single byte (8-bits) is to be read from or written to DRAM. During a word access, all four blocks are activated, allowing the full 32-bits to be accessed. During a byte access, the two least significant address bits, A[0:1], select one of the 8-bit blocks to be activated, and only the appropriate eight bits are read from or written to the DRAMs.

6.3.2 DRAM Configurations

The *Page Size* setting (in the *MEMC Control Register*) controls how the DRAM address is presented on the RAM address bus, RA[0:9], as shown in Table 2.

Total amount of RAM	Page Size setting	Typical Configuration	Row & Column address presented on	Bank Select
256 KBytes	4KBytes	8 64Kx4 DRAM 32 64Kx1 DRAM	RA[0:7]	-
512 KBytes	4KBytes	16 64Kx4 DRAM	RA[0:7]	RA[8]
1 MByte	8KBytes	8 256Kx4 DRAM 32 256Kx1 DRAM	RA[0:8]	-
2 MBytes	16KBytes	16 256Kx4 DRAM	RA[0:8]	RA[9]
4 MBytes	32KBytes	8 1Mx4 DRAM 32 1Mx1 DRAM	RA[0:9]	-

Table 2: Dynamic RAM address bus configurations

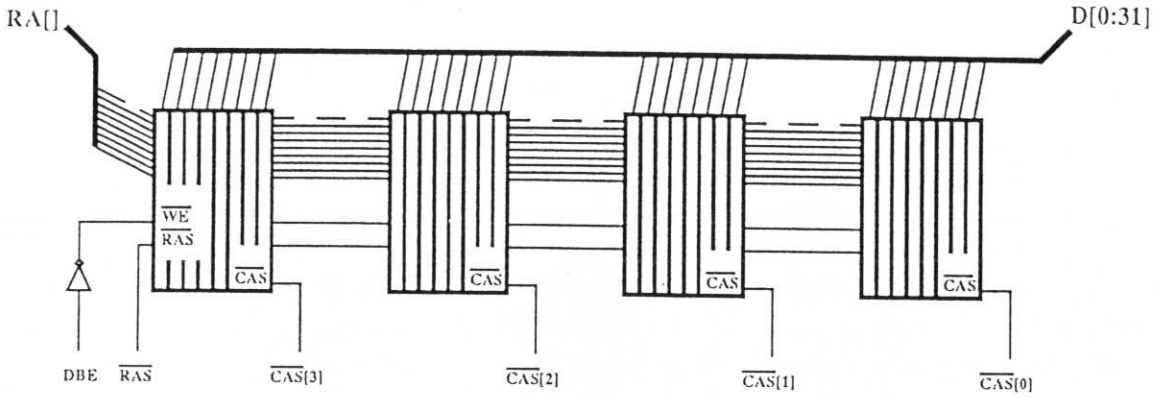
NOTES:

(i) The **RA[0:9]** lines are derived from the processor address bus, the *Logical to Physical Address Translator* and the *DMA Address Generators* by passing the lines through the *DRAM Address Multiplexer*, which inverts and re-orders the address bits. See Appendix A.

(ii) When only 256KBytes are connected to MEMC (8 off 64Kx4 or 32 off 64Kx1 DRAMs), the Bank select address bit, **RA[8]** is ignored by the DRAMs, so Physical pages 64-127 map onto Physical pages 0-63.

There are three basic Dynamic RAM configurations as follows:

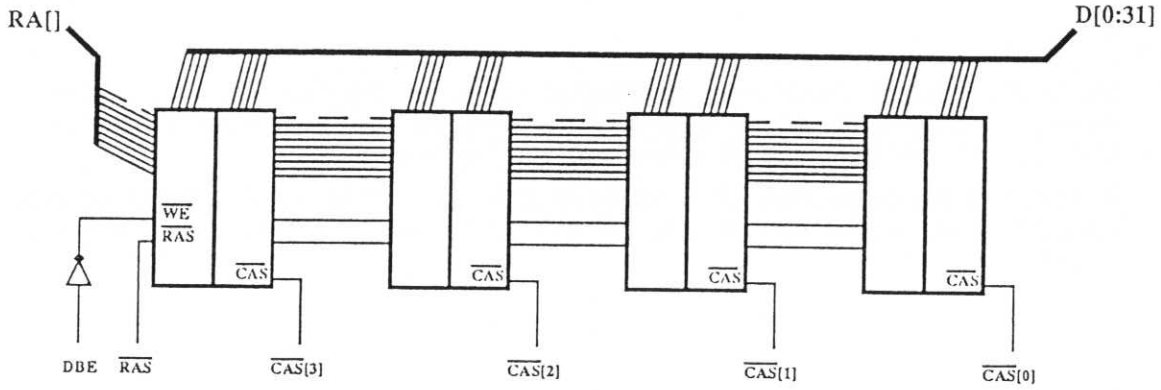
(1) Thirty-two 64Kx1, 256Kx1 or 1Mx1 DRAMs



This configuration splits the thirty-two 1-bit wide DRAMs into four blocks of eight bits. Each 8-bit block is controlled by one of the CAS[0:3] lines, allowing independent access to any of the byte-wide blocks.

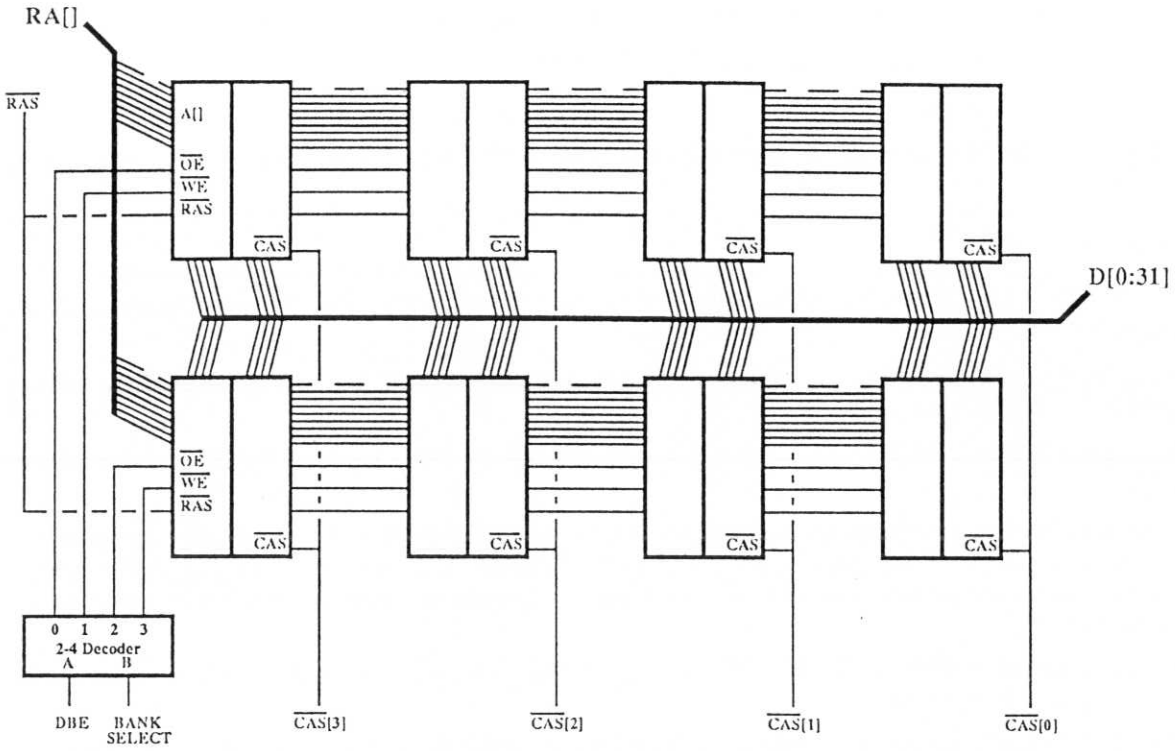
The DRAM write-enable line is derived by inverting the **DBE** signal from MEMC. The RAM address bus, **RA**[0:9], and **RAS** strobe are routed to all the DRAMs.

(2) Eight 64Kx4, 256Kx4 or 1Mx4 DRAMs



This configuration is essentially the same as that used for the 1-bit wide DRAMs. In this case, only eight chips are required for the full 32-bit data bus.

(3) Sixteen 64Kx4 or 256Kx4 DRAMs



The sixteen chips are configured as two banks of eight 4-bit wide DRAMs in parallel. One of the RAM address bits is used as a bank select line (valid at the same time as the Column addresses).

NOTE: A 2-4 Decoder (with active LOW outputs) is used to derive an output-enable and write-enable signal for each bank. This ensures that only the one bank is activated during any DRAM access.

6.3.3 DRAM cycles

There are three main types of DRAM accesses as follows:

(i) Processor access (fetching instructions and reading/writing data)

(ii) DMA operation (fetching Video, Cursor or Sound data)

(iii) Refresh operation

MEMC uses the *page mode access* capability of DRAMs, performing sequential accesses (S-cycles) where possible.

Processor accesses

MEMC monitors a processor signal, SEQ, which indicates that the next processor access will be sequential.[1]

If the SEQ signal is LOW in the processor cycle preceding a DRAM access (SEQ is a pipelined signal), an N-cycle DRAM access will be used.

Subsequent DRAM accesses will use S-cycles, provided that the SEQ signal is HIGH in the preceding cycle.

DMA operations may not interrupt the processor if it is about to perform an S-cycle memory access, so the maximum number of consecutive S-cycles is restricted to three. This limits the worst case DMA latency, whilst retaining the improvement that S-cycles bring. An N-cycle is forced under either of the following conditions:

(i) The processor SEQ signal was LOW in the preceding cycle, indicating that this access would not be to a sequential address.

(ii) The processor address lines A[2] and A[3] were both HIGH in the preceding cycle. This restricts the maximum number of consecutive S-cycles to three.

MEMC includes an optimisation for Non-sequential memory accesses that follow an internal processor cycle. During internal cycles, the ARM processor outputs an address, and sets SEQ HIGH if the address will be valid in the next cycle (see Figure 2).
In an internal cycle preceding a processor memory access, the $\overline{\text{MREQ}}$ line will be set LOW. When MEMC sees SEQ HIGH, $\overline{\text{MREQ}}$ LOW and a valid DRAM address on A[0:25] during an internal cycle, it starts a DRAM N-cycle immediately, effectively overlapping the internal cycle with the first half of an Non-sequential cycle. The processor DRAM access can then complete with a DRAM S-cycle, as the row address was strobed in during the internal cycle. This special operation does not occur if the DRAM address has both A[2] and A[3] set HIGH (this is a consequence of the multiple S-cycle limiting logic).

DMA operations

DMA operations always fetch four words (sixteen bytes) of data sequentially from the DRAMs. Thus, DMA operations are composed of an N-cycle read followed by three S-cycle reads.

Refresh operations

A refresh operation is effectively a single N-cycle DRAM read operation, with the exception that the $\overline{\text{CAS}}$[0:3] lines are not strobed LOW, so the data bus is not driven.

---

[1] NOTE: MEMC uses the SEQ to detect sequential accesses, and does not perform its own check on the address bus. If the SEQ signal is asserted incorrectly, the wrong memory page may be accessed.

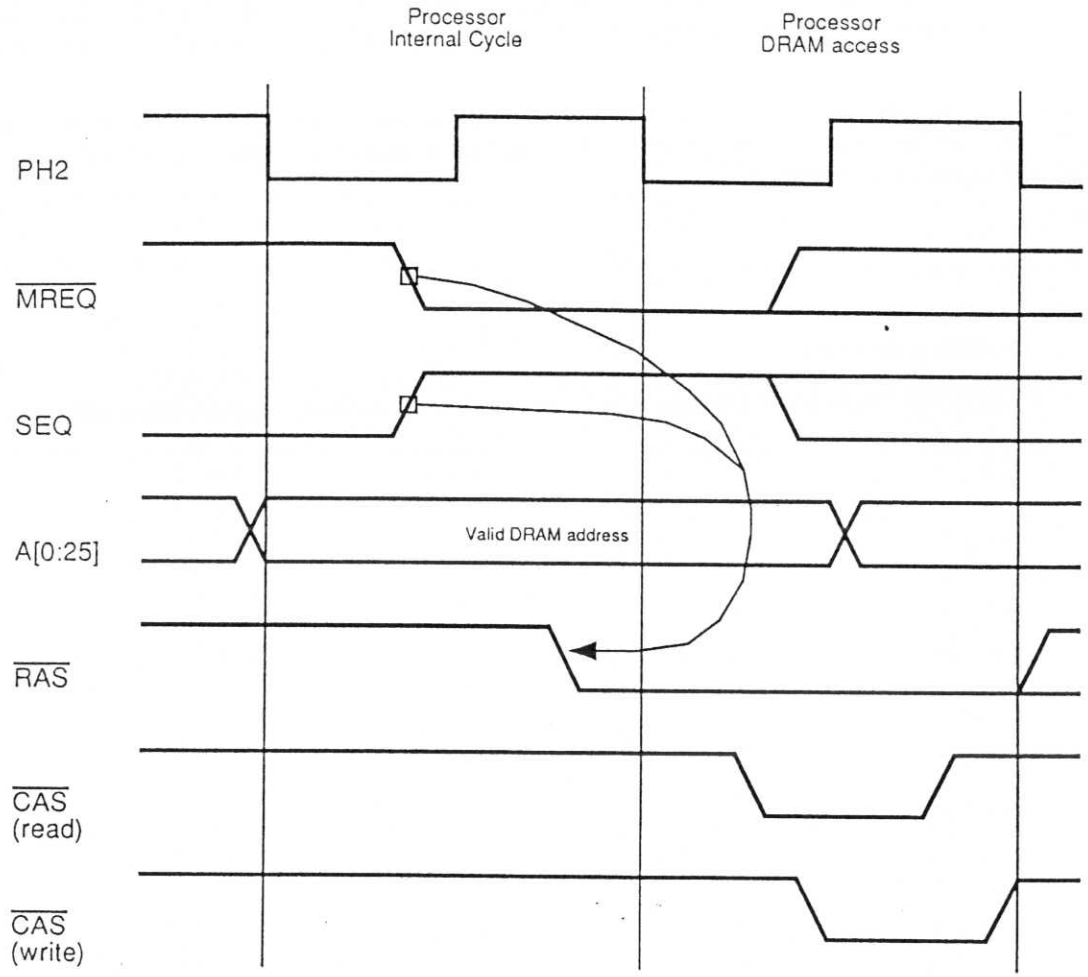


Figure 2: Optimisation for DRAM accesses following Internal cycles

6.3.4 DRAM Timing

Timing diagrams for DRAM operations are given in Section 8.4.1, Section 8.4.2 and Section 8.8.2.

The $\overline{\text{CAS}}$[0:3] strobes are generated early in read operations, and late in write operations to improve setup and hold times on the data bus. If an **ABORT** is generated during an N-cycle, the $\overline{\text{RAS}}$ strobe will be activated as usual, but the $\overline{\text{CAS}}$[0:3] signals are suppressed for this and any subsequent S-cycles, effectively disabling the DRAM.

6.4 Read Only Memory (ROM) Interface

To give the ROM as long as possible to access its data, the ROM chip select signal from MEMC, $\overline{\text{ROMCS}}$, is driven LOW at the start of every processor memory access (except S-cycles), and only disabled when the processor address lines have been decoded as addressing another part of the memory map.

The ROM area of the processor memory map is divided into two sections, High ROM and Low ROM. The ROM access time in each area may be independently programmed through the *MEMC Control Register*. Four ROM access times are programmable:

(i) 450ns

(ii) 325ns

(iii) 200ns

(iv) 200ns with 60ns nibble-mode

When a ROM cycle is performed, the processor clocks are stretched as necessary to match the ROM access time.

6.5 MEMC Control Register

The *MEMC Control Register* is a programmable register that controls some of the functions of MEMC. The parameters are encoded into the processor address lines, as shown in Figure 3, and transferred to the Control Register by performing a write operation to the appropriate address whilst MEMC is in Supervisor mode.

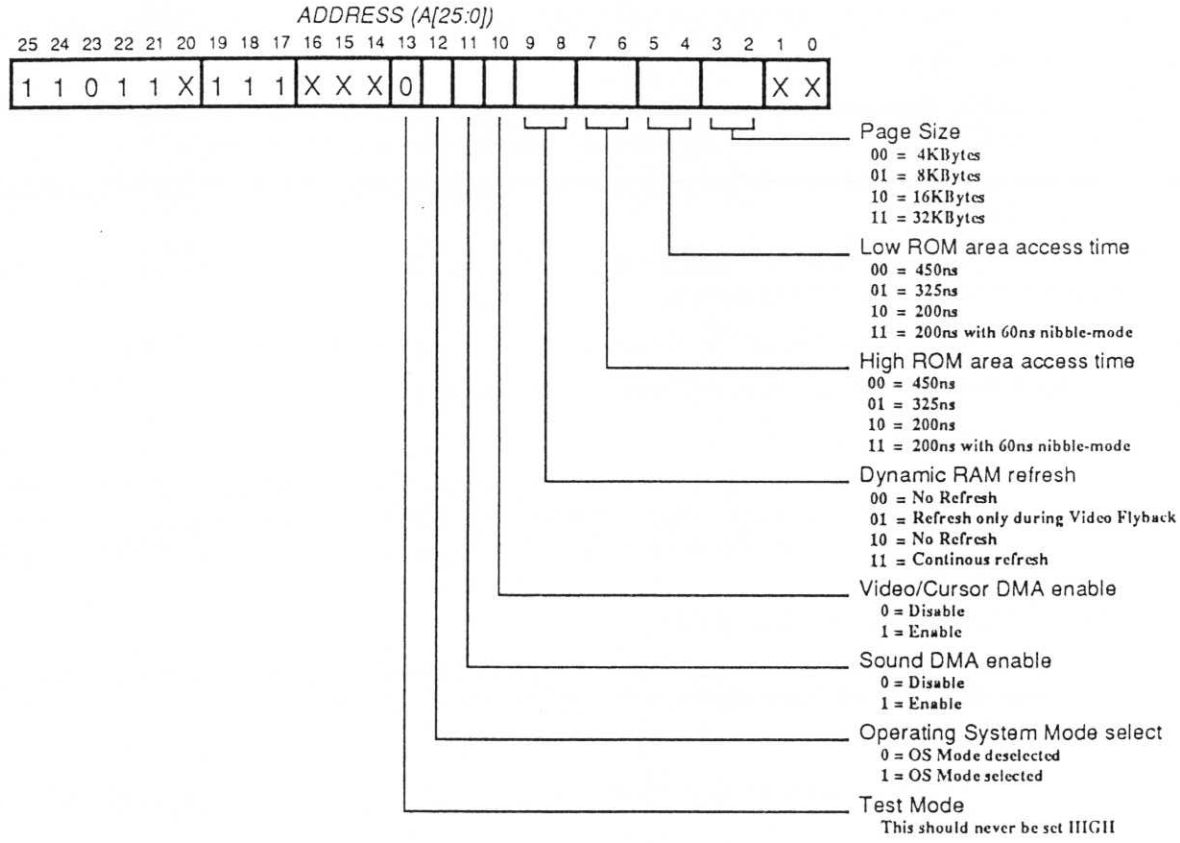


Figure 3: Programming the MEMC Control Register

The following functions of MEMC are programmable:

- *Logical and Physical Page size*

  The Logical and Physical page size must be set to correspond to the type of DRAM connected to MEMC. Page sizes of 4KBytes, 8KBytes, 16KBytes or 32KBytes may be selected.

  A default Page size of 4KBytes is selected when **RESET** is asserted.

- *ROM Access Times*

  ROM access times of 450ns, 325ns, 200ns or 200ns with 60ns nibble-mode may be selected for each of the two ROM areas (High ROM and Low ROM).

  The ROM access time in both High and Low ROM areas defaults to 450ns when **RESET** is asserted.

- *Refresh Operations*

  Video DMA operations address DRAM locations sequentially at regular intervals, effectively refreshing DRAM, but video DMA operations are normally suspended during flyback.

  For high resolution displays, the flyback time is short enough to ensure that the DRAM refresh period is not exceeded.

  Broadcast standard displays have longer flyback times, and extra DRAM refresh must be provided during flyback to retain DRAM integrity.

  When no Video DMAs are requested, all refresh operations must be generated by MEMC.

  To cover these requirements, three modes of refresh operation are available:

  (i) Continous refresh

  A refresh operation is attempted every 4µs. The refresh operation uses the *DMA Video pointer* as the refresh address source, incrementing the pointer after use. As this effectively scrambles the *DMA Video pointer*, this mode should never be selected while a Video display is being generated.

  (ii) Refresh only during Video flyback

  A refresh operation is attempted every 4µs while **FLYBK** is HIGH. This mode should be selected when a broadcast standard Video display is being generated.

  (iii) No refresh

  This mode of operation disables refresh entirely, relying on Video DMA operations to refresh the DRAM.

  Refresh operations take a single N-cycle. The processor clocks are halted during the operation (unless the processor is executing internal cycles) to ensure that the processor does not try to use the DRAM, and the refresh address is strobed into the DRAMs using the $\overline{\text{RAS}}$ line.

  The refresh address is provided using the *DMA Video pointer*, which is incremented after every refresh operation.

  Refresh operations have a lower priority than DMA operations and will be delayed if a DMA operation is in progress when the refresh is attempted.

  NOTE: There is no default setting for refresh operations, so the system software must turn on some form of refresh before using the DRAM. As neither the Video DMA enable bit nor the refresh mode is affected by **RESET** the Video Display may be maintained over a system reset.

- *Direct Memory Access (DMA) control*

  The Video/Cursor and Sound DMA operations may be enabled or disabled as required.

  Sound DMA operations are disabled when **RESET** is asserted.

  Video/Cursor operations are unaffected by **RESET**.

- *Operating System Mode*

  When Operating System mode is enabled, the processor may access certain protected Logical pages in the *Logically Mapped RAM* space. As with all *MEMC Control Register* parameters, Operating System mode may only be changed while MEMC is in Supervisor mode.

  Operating System mode is disabled when **RESET** is asserted.

- *Test Mode*

  Test mode reconfigures MEMC to a known state for functional testing of the chip out of the system. Test mode must **NEVER** be enabled during normal operation, as it removes all sources of DRAM refresh, and halts the processor.

  Test mode is disabled when **RESET** is asserted.

6.6 Logical to Physical Address Translator

The physical RAM is divided into 128 *Physical pages*, which the processor may either access directly through the *Physically Mapped RAM* area of the memory map, or indirectly through the *Logically Mapped RAM* area (which is composed of *Logical pages*). The *Logical to Physical Address Translator* controls the mapping of Logical pages to Physical pages, and allows a level of protection to be attached to each Logical page.

6.6.1 Page Protection Levels

The Logical page *Protection Levels* available are shown in Figure 4.

		Page Protection Level (PPL[1:0])			
		00	01	10	11
MEMC Protection Mode	Supervisor	Read/Write	Read/Write	Read/Write	Read/Write
	Operating System	Read/Write	Read/Write	Read	Read
	User	Read/Write	Read	—	—

Figure 4: Logical page Protection Levels

The protection level is specified by two bits, but two of the four patterns are identical, so only three protection levels are available:

(i) The lowest protection level (**PPL[1]=0,PPL[0]=0**) allows the Logical page to be freely accessed when MEMC is in any protection mode.

(ii) The medium protection level (**PPL[1]=0,PPL[0]=1**) allows the Logical page to be freely accessed when MEMC is in Supervisor or OS mode, but prevents write operations from User mode.

(iii) The highest protection level (**PPL[1]=1,PPL[0]=X**) allows the Logical page to be accessed when MEMC is in Supervisor mode, prevents write operations from OS mode, and disallows any User mode accesses.

If the protection mode of MEMC is insufficently privileged to access a protected page, or the logical page being accessed has no physical page mapping, the **ABORT** line will be taken HIGH to inform the processor that the memory operation was aborted, and the $\overline{\text{CAS}}$**[0:3]** lines will be held HIGH to ensure the DRAM is not activated.

6.6.2 Address Translator mapping

The Address Translator consists of a 128 entry lookup table. Each of the 128 entries corresponds to a Physical page number. A Logical to Physical mapping is made by storing a Logical page number in the appropriate entry. Each entry also contains the two bit Page Protection Level.

When the processor accesses *Logically Mapped RAM*, the Logical page number is applied to all 128 table entries simultaneously. If one of the entries contains the required Logical page number, and the current operating mode of MEMC is sufficiently privileged to overcome the Page Protection Level, the appropriate Physical page (the number of the entry that matched) is output to the DRAMs.

If none of the entries match the requested Logical page, or a match is found but the Page Protection Level is too high, the **ABORT** line is set HIGH, and the DRAM is not activated.

NOTE: It is possible to store the same Logical page number in more than one entry; however, when that Logical page is accessed, those entries will all match, and an invalid Physical page number will result.

6.6.3 Programming the Address Translator

The Address Translator is programmed by specifying the Physical and Logical page numbers that are to be associated and the required protection level.

As MEMC does not monitor the processor data bus, the information is encoded into the address lines, and conveyed to the Address Translator by performing a write operation to the calculated address (with MEMC in Supervisor mode). Note that the page size not only affects the number of Logical pages available, but also changes the bit order in which the Logical and Physical Page numbers are specified. Diagrams showing how the information is encoded into an address for each of the four possible page sizes are shown in Figure 5.

NOTES:

(i) The Address Translator is undefined on power up.

(ii) The Address Translator mappings are not affected by reset, but are effectively scrambled if the page size is changed.

(iii) Only one Physical page should be mapped to any given Logical page.

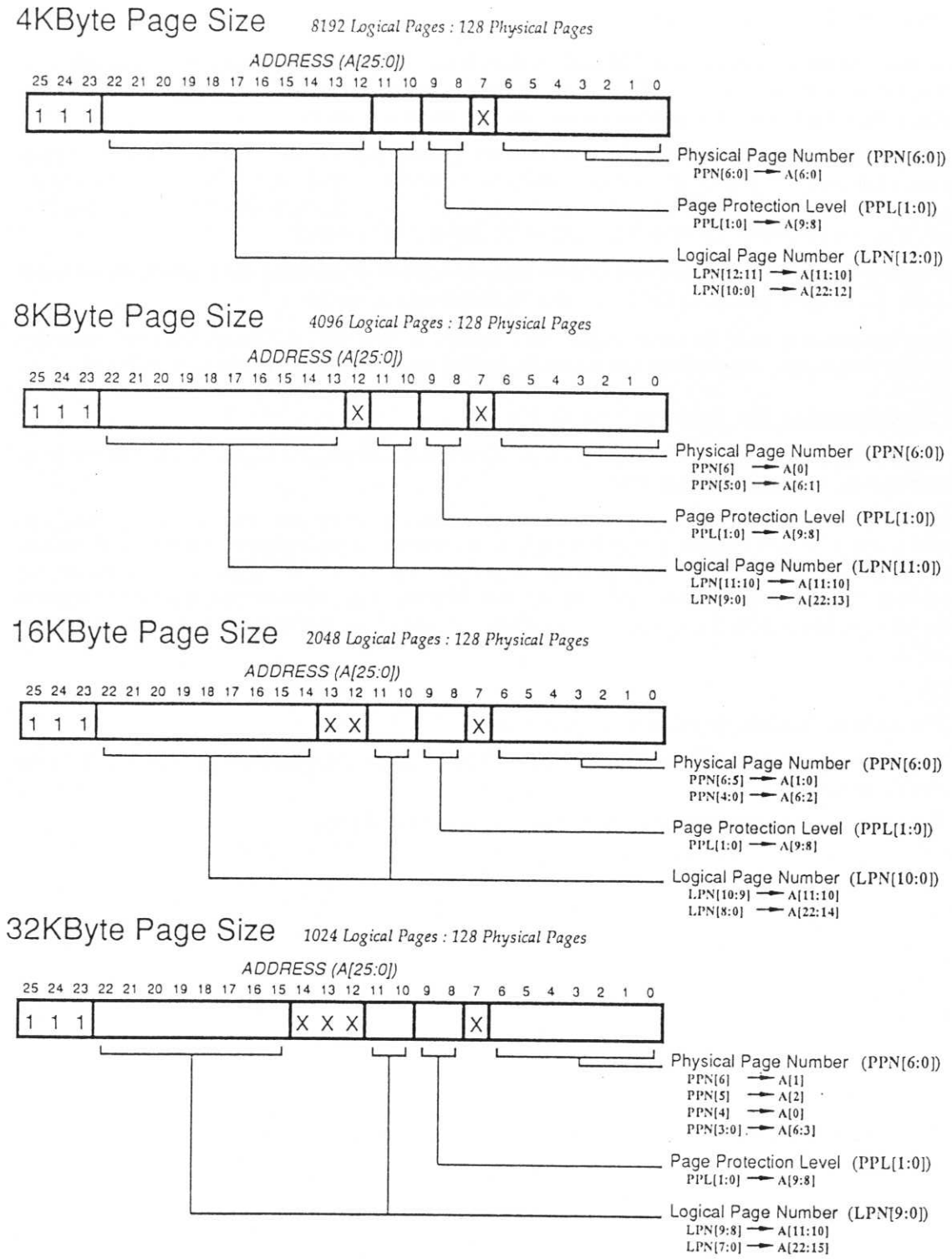


Figure 5: Programming the Logical to Physical Address Translator

6.7 DMA Address Generators

The DMA Address Generators automatically generate addresses during a DMA service. These DMA addresses are used to obtain 16 bytes of data from the DRAM (all DMA data must be aligned on 16 byte boundaries). The data is obtained using four DRAM accesses; each access supplies one word (four bytes) of data which may be latched from the data bus when the appropriate DMA acknowledge line is strobed.

The DMA Address Generators implement three sets of buffers:

(i) Video Buffer

(ii) Cursor Buffer

(iii) Sound Buffers

These buffers are defined by registers in the DMA Address Generators, which are programmed by encoding the data in the address bus and performing a write operation with MEMC in Supervisor mode. Figure 6 shows how an address is calculated to store data in the Address Generator registers.

Address of Processor write operation:

ADDRESS (A[25:0])

REGISTER	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
Vinit	1 1 0 1 1 X	0 0 0	New Register value *(see Note 1)*	X X
Vstart	1 1 0 1 1 X	0 0 1	New Register value *(see Note 1)*	X X
Vend	1 1 0 1 1 X	0 1 0	New Register value *(see Note 1)*	X X
Cinit	1 1 0 1 1 X	0 1 1	New Register value *(see Note 1)*	X X
Sstart	1 1 0 1 1 X	1 0 0	New Register value *(see Note 1)*	X X
SendN	1 1 0 1 1 X	1 0 1	New Register value *(see Note 1)*	X X
Sptr	1 1 0 1 1 X	1 1 0	X X X X X X X X X X X X X X X X	

sce Note 2

NOTES:

(1) The Register value is calculated by dividing the *Physical RAM address* by 16.

(2) The following side effects occur when the Sound buffer address generators are programmed:

- Programming the *Sstart* register sets the *Next Sound Buffer Valid flag.*
- Programming the *Sptr* register forces a sound buffer swap.

Figure 6: Programming the DMA Address Generator registers

NOTES:

(i) The Address Generator works to a resolution of 16 bytes (the number of bytes fetched during a DMA operation), so all DMA buffers must start and end on 16 byte boundaries. The data stored in the Address Generator registers is calculated by dividing the appropriate Physical RAM address by sixteen.

(ii) When the Sound pointer register, *Sptr*, is programmed, no immediate value is specified. Instead, a Sound buffer swap is forced, copying the value from *Sstart* to *Sptr*, and resetting the *Next Sound Buffer Valid flag*. (see Section 6.7.3)

(iii) The processor may write to the DMA registers at any time, but multiple, consecutive DMA register write operations (using the ARM *Store Multiple* instruction) should never be used, as this may inhibit the initialisation of the Video pointer register, *Vptr*. (see Section 6.7.5)

(iv) The DMA Address generators are limited to addressing the bottom 512KBytes of Physical memory. However, the *Logical to Physical Address Translator* can be used make this 512KByte Physical address space appear anywhere in the processor's 32MByte Logical address space.

6.7.1 Video Buffer

This is a circular buffer, as shown in Figure 7. The buffer is a section of memory delimited by *Vstart* and *Vend* that contains all the Video data for a frame.

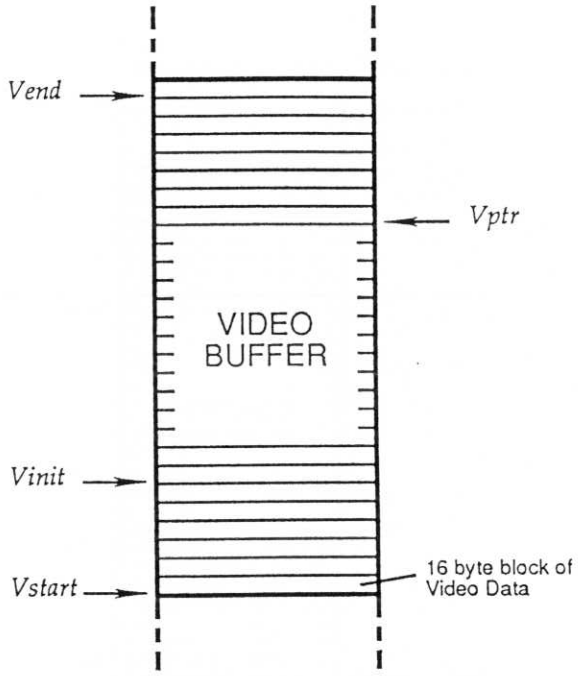


Figure 7: Circular Video Data Buffer

When a Video DMA is requested, the address held in *Vptr* (the Video Pointer) is used to obtain four consecutive 32-bit words of data (16 bytes) from the DRAM. The **VIDAK** line is strobed LOW as each word of data is read from the DRAM. The *Vptr* register is then incremented ready to point to the next four words of Video data, unless it has reached the end of the buffer (as delimited by *Vend*), in which case *Vptr* is reset to the start of the buffer (as defined by *Vstart*).

The *Vinit* register contains the address to which *Vptr* will be initialised just before the new display frame begins (denoted by a HIGH to LOW transition on **FLYBK**), and should be programmed with the address of the first byte of video data for the new frame. Hardware scrolling is effected by reprogramming *Vinit*.

The processor may program the *Vstart*, *Vinit* and *Vend* registers (provided MEMC is in Supervisor mode). The *Vptr* register cannot be altered directly by the processor, but is always reset to the value contained in *Vinit* before a new Video frame is displayed.

The Video Pointer register, *Vptr*, doubles as a refresh counter. When a refresh is performed, the *Vptr* address is output to the DRAMs, and *Vptr* is incremented to the next 16 byte boundary (no check is made that *Vptr* has reached the end of the buffer). As refresh operations alter the contents of *Vptr*, continous refresh must never be enabled while Video DMAs are operative.

6.7.2 Cursor Buffer

This is a linear buffer, and is shown in Figure 8. The cursor data is contained in a section of memory whose initial (low) address is stored in the *Cinit* register. While **FLYBK** is HIGH, *Cptr* (the Cursor Pointer) is initialised to the address held in *Cinit*. When a Cursor DMA is serviced, the address held in *Cptr* is sent to the DRAM, and used to obtain four consequtive 32-bit words of data (16 bytes). The *Cptr* is then incremented ready to point to the next four words of Cursor data.

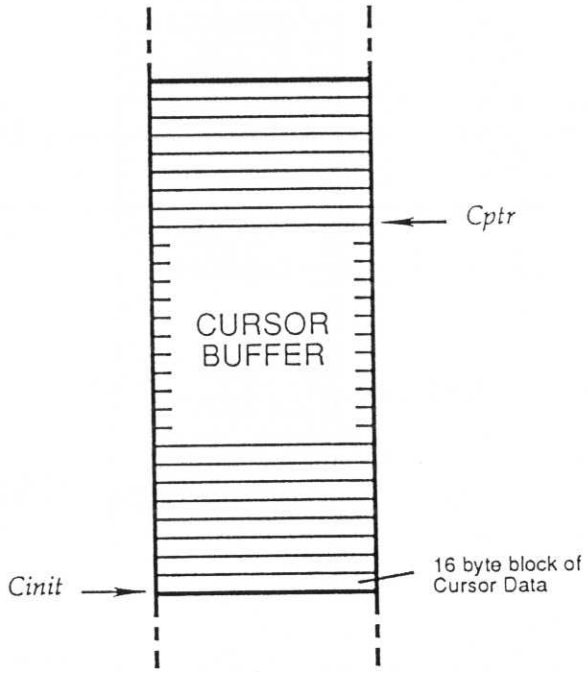


Figure 8: Linear Cursor Data Buffer

6.7.3 Sound Buffers

Sound data is divided into areas of memory called *sound buffers*. The sound system can support any number of sound buffers using the DMA Address Generators and interrupt driven software. Sound data is extracted from one buffer at a time, and when each buffer is exhausted, a new sound buffer is used.

The DMA Address Generators contain information on two sound buffers as follows (see Figure 9):

(i) *Current Sound Buffer*

This is the buffer from which sound data is extracted when a Sound DMA is requested. The address of the next 16 bytes of sound data to be supplied in response to a Sound DMA request is held in the Sound Pointer register, *Sptr*, and the end of the Current Buffer is delimited by the Current Sound End register, *SendC*.

(ii) *Next Sound Buffer*

This is the buffer of sound data which will be used once the Current Sound Buffer is exhausted. The Next Sound Buffer is defined by programming the *Sstart* and *SendN* registers. (**NOTE**: The processor can only ever program the start and end addresses of the Next Sound Buffer).

A hardware flag, *Next Buffer Valid*, is set when the Next Sound Buffer registers have been programmed.

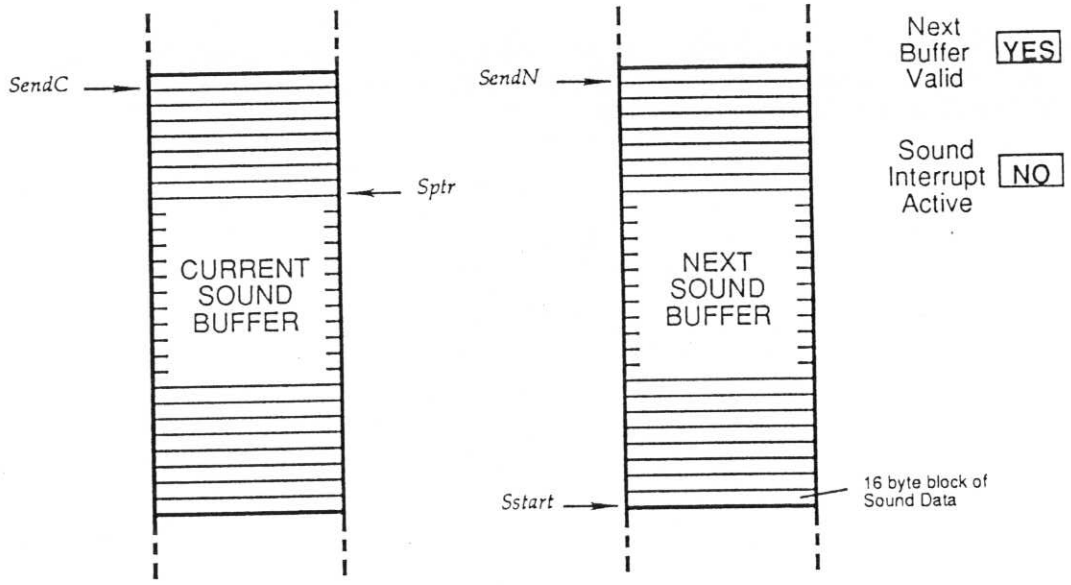
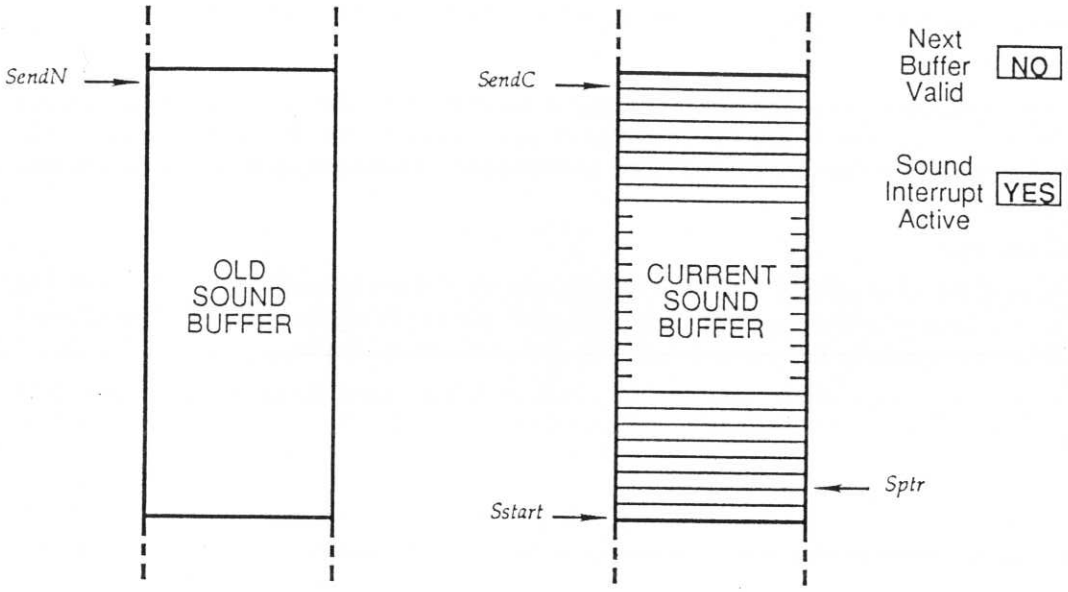


Figure 9: Current and Next Sound Buffers

Sound Buffer operation

(1) Sound Buffer swap

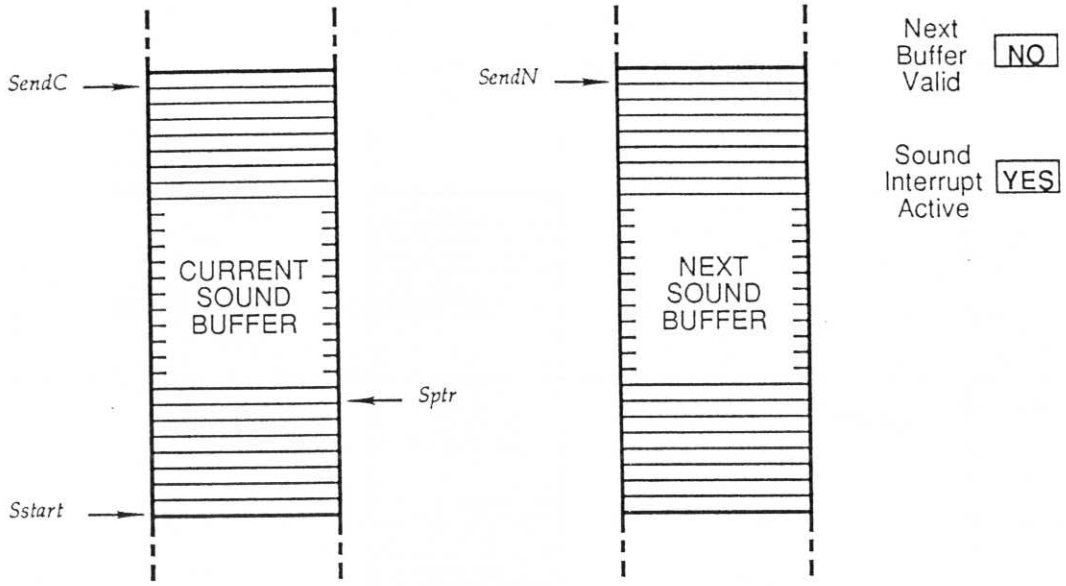
When *Sptr* reaches the end of the Current Buffer, it swaps to the start of the Next Buffer, provided the Next Buffer Valid flag is set.



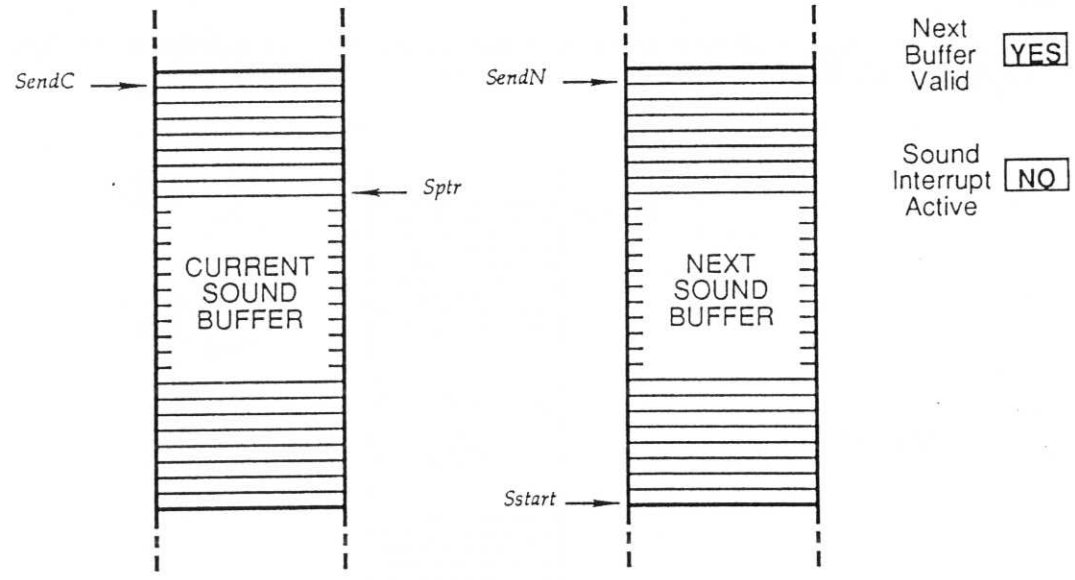
The buffer swap resets the Next Buffer Valid flag, and generates a sound interrupt by driving the $\overline{\text{SIRQ}}$ pin LOW. The *SendC* and *SendN* registers swap over, so that the value previously set up in *SendN* defines the end of the new Current Buffer.

(2) Next Sound Buffer setup

The processor should react to the sound interrupt by programming the start and end addresses for a buffer containing new sound data (the Next Sound Buffer). The first part of this process is to define the end address by reprogramming *SendN*.



The processor may now define the start address by reprogramming *Sstart*. This fully defines the Next Buffer, setting the Next Buffer Valid flag and driving $\overline{\text{SIRQ}}$ HIGH. The sequence of events now repeats from Step (1) again.



NOTE: If the processor fails to setup a new *Sstart* value before the Sound Pointer reaches the end of the Current Buffer, the Sound Pointer will jump back to the start of the Current Buffer (as defined by the old value of *Sstart*), thus repeating the last buffer of sound data.

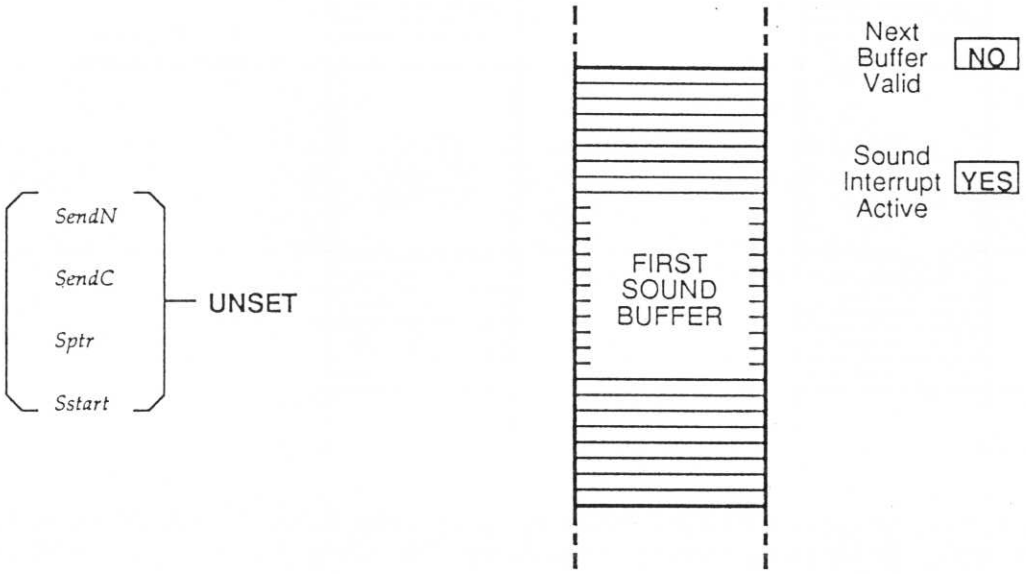
Initialising the Sound Buffers

The following procedure is recommended for initialising the Sound Buffers to a known state:

(1) Initial state

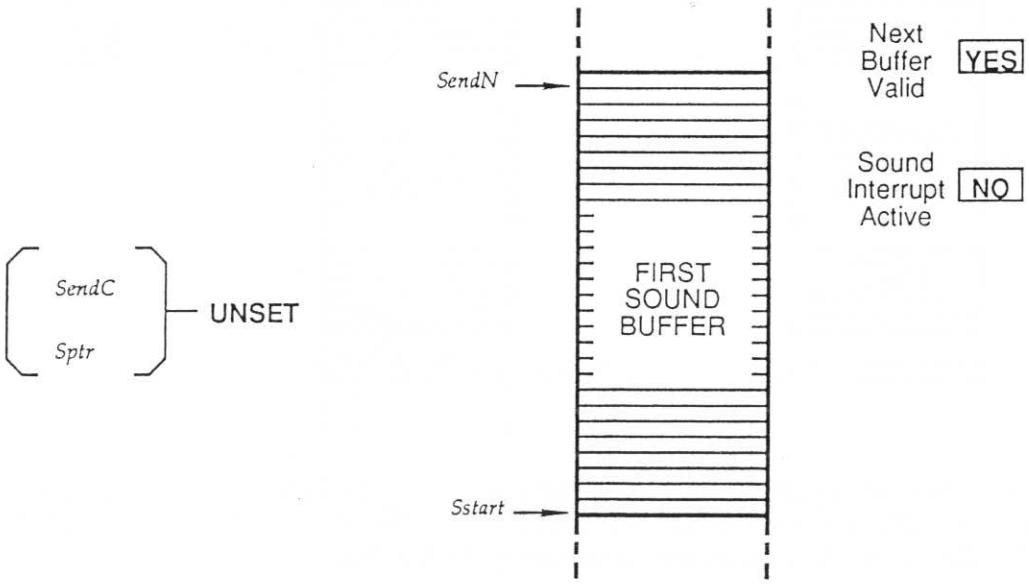
After reset Sound DMA operations are disabled. To start the sound system, the processor must first fill an area of memory with sound data; this will become the First Sound Buffer.

NOTE: The Sound Interrupt line, $\overline{\text{SIRQ}}$, is set LOW when **RESET** is asserted.



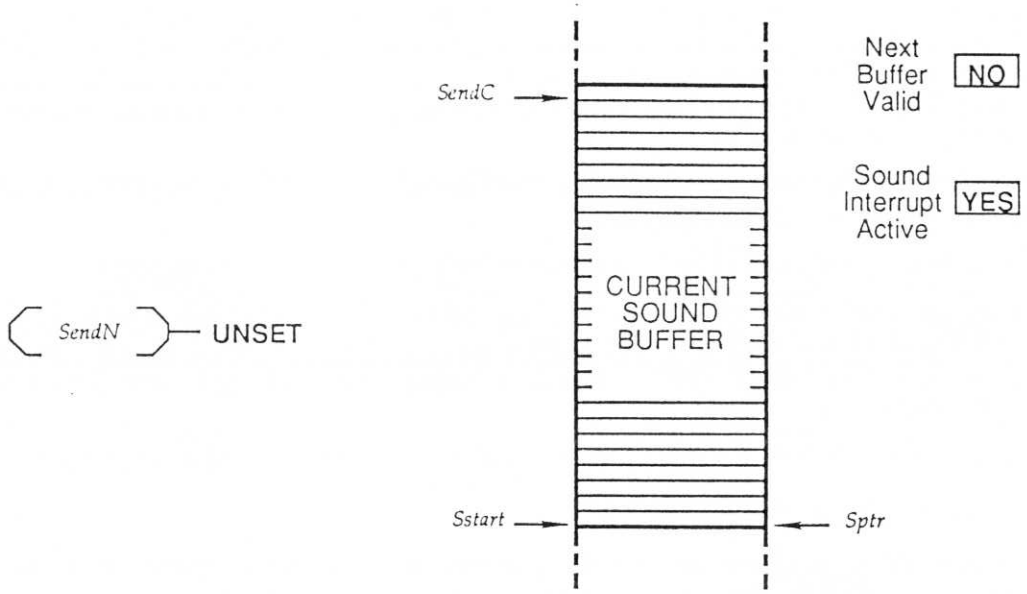
(2) Defining the First Buffer

The *SendN* and *Sstart* registers are then programmed with the end and start addresses of the First Buffer. This drives $\overline{\text{SIRQ}}$ HIGH.



(3) Initialising the Sound Pointer

The Sound Pointer is now defined by performing a write operation to the *Sptr* register. Rather than defining an immediate value to be stored in *Sptr*, this operation forces a buffer swap, copying the contents of *Sstart* to *Sptr*, swapping over *SendN* and *SendC*, and driving the $\overline{\text{SIRQ}}$ pin LOW.



The processor may now enable Sound DMA operations by reprogramming the *MEMC Control Register*, and handle the sound interrupt in the usual way to set up the Next Buffer.

6.7.4 Processor/DMA memory arbitration

DMA operations read four words from the DRAM. The memory accesses are organised as an N-cycle followed by three S-cycles. As the DMA operation uses the system data bus, the processor must be prevented from performing memory accesses until the DMA has finished.

If both a DMA operation and a processor memory access cycle are pending, the DMA operation will normally have priority, and the processor will be disabled at the end of its current cycle until the DMA operation completes. However, processor S-cycle memory accesses (either to DRAM or nibble-mode ROM) have higher priority than DMA operations, and the DMA operation will be postponed until the processor stops performing S-cycles. Excessive DMA latency is avoided by limiting the maximum number of consecutive processor S-cycles to three.

Processor internal cycles may occur concurrently with DMA operations, but the processor will be disabled if it attempts any memory access cycle during DMA.

The processor is disabled at the start of a cycle by stopping the processor clocks with **PH1** HIGH.

The DMA request may arrive while the processor is suspended awaiting the completion of an IO cycle. In this case, the $\overline{\text{IORQ}}$ signal is driven HIGH when **REF8M** next goes LOW. The DMA operation may then begin, and when it completes, the IO cycle is resumed by setting $\overline{\text{IORQ}}$ LOW again (provided no more DMA or refresh operations are pending).

The **DBE** line is always driven LOW during DMA operations to disable the processor data bus drivers.

6.7.5 DMA handshaking

Video and Cursor DMA operations are mutually exclusive, and are both requested by taking the $\overline{\text{VIDRQ}}$ line LOW. The $\overline{\text{HSYNC}}$ line determines whether a Video or Cursor operation is to be performed. If $\overline{\text{HSYNC}}$ is LOW when $\overline{\text{VIDRQ}}$ is driven LOW a Cursor DMA is performed, otherwise a Video DMA is performed.

A Sound DMA operation is requested by taking the $\overline{\text{SNDRQ}}$ line LOW.

When the DMA operation is performed, the DRAM is accessed, and an acknowledge line, (either $\overline{\text{VIDAK}}$ for Video/Cursor DMA, or $\overline{\text{SNDAK}}$ for Sound DMA) is strobed low as each word of data becomes available on the data bus. The rising edge of $\overline{\text{VIDAK}}$ or $\overline{\text{SNDAK}}$ should be used to latch the DMA data from the data bus[2] .

The appropriate DMA request line ($\overline{\text{VIDRQ}}$ or $\overline{\text{SNDRQ}}$) should be taken HIGH when the first DMA acknowledge is given unless a consecutive DMA is to be performed.

The **FLYBK** signal prompts MEMC to initialise the Video and Cursor buffer pointers. The Cursor pointer is initialised during flyback, and the Video pointer is initialised just after **FLYBK** goes LOW. The initialisation of the Video pointer is left this late to allow it to be used as refresh pointer during flyback.

The $\overline{\text{VIDRQ}}$, $\overline{\text{SNDRQ}}$, $\overline{\text{HSYNC}}$ and **FLYBK** signals may be asynchronous, so they are all passed through two synchronisation latches in MEMC to avoid synchronisation failure.

---

[2] *Some DRAMs may disable their data bus drivers before the DMA acknowledge line goes HIGH. In this case, the dynamic storage time of the data bus is normally sufficient to hold the data valid until it is latched.*

Selecting Video or Cursor DMA operations

The $\overline{\text{HSYNC}}$ signal determines whether a Video or Cursor DMA is to be performed, and is latched on the HIGH to LOW transition of $\overline{\text{VIDRQ}}$. The hold time on $\overline{\text{HSYNC}}$ must be sufficient to allow the synchronisation latches in MEMC to capture its state.

When two or more Video/Cursor DMA operations occur consecutively, $\overline{\text{HSYNC}}$ is sampled on the falling edge of the penultimate $\overline{\text{VIDAK}}$ strobe, and its state at this point determines whether the next DMA operation will fetch Video or Cursor data.

The setup and hold requirements of $\overline{\text{HSYNC}}$ are given in Section 8.8.3.

DMA latencies

Video/Cursor DMA requests have a higher priority than Sound requests, and will always be serviced first.

The maximum DMA latency from the time a $\overline{\text{VIDRQ}}$ or $\overline{\text{SNDRQ}}$ line is taken low to the first 32-bits of DMA data being read from DRAM is as follows:

(i) Video/Cursor DMA latency

$\overline{\text{VIDRQ}}$ passes through two synchronisation latches. The delay through these latches varies from approx 10ns to 125ns, depending on the relative phase of $\overline{\text{VIDRQ}}$ and the internal 8MHz synchronising clock. It then takes 187ns to process the Video request, and prepare to execute a DMA cycle. A further delay of 500ns may be incurred if the processor had just started a worst case uninterruptable DRAM access (N-cycle + three S-cycles with no internal cycles) in the preceding 8MHz internal clock cycle. Finally, it takes 250ns for the DRAM N-cycle read operation to supply the first word of video/cursor data.

Thus, the minimum and maximum delays from $\overline{\text{VIDRQ}}$ going LOW to the first word of data being available from the DRAMs are:

Minimum Video/Cursor DMA latency = 10+187+250 ≈**450ns**

Maximum Video/Cursor DMA latency = 125+187+500+250 ≈**1070ns**

(ii) Sound DMA latency

The Sound DMA latency is similar to the Video/Cursor DMA latency. However, Sound DMA operations have a lower priority than Video/Cursor DMA operations, and will be delayed by 625ns for every consecutive Video/Cursor DMA operation that is requested at the same time as, or after $\overline{\text{SNDRQ}}$ goes LOW.

Thus, the minimum and maximum delays from $\overline{\text{SNDRQ}}$ going LOW to the first word of data being available from the DRAMs are:

Minimum Sound DMA latency = 10+187+250 ≈**450ns**

Maximum Sound DMA latency = $125+187+500+250+(625*\text{DMA}_{V/C})$
$\approx 1070+(625*\text{DMA}_{V/C})$ ns

where $DMA_{V/C}$ is the maximum number of consecutive Video/Cursor DMA operations that may occur while the Sound DMA is pending.

Flyback requirements

The Video and Cursor buffer pointers must be reset between one video frame and the next.

The Cursor pointer is initialised during flyback (signalled by **FLYBK** HIGH). The initialisation takes 250ns and occurs provided that the following conditions are met:

(i) **FLYBK** is HIGH (signalling that flyback is in operation).

It may take up to 250ns for MEMC to synchronise and process the LOW to HIGH transition of the **FLYBK** signal.

(ii) The processor is performing a normal memory access (not an S-cycle), but is NOT writing to the *DMA Address Generator* or the *MEMC Control Register*.

(iii) No DMA operation is in progress.

(iv) No refresh operation is in progress.

NOTE: These conditions may be satisified many times during flyback, and the Cursor pointer will be initialised on each occasion. The **FLYBK** signal must remain HIGH long enough to initialise the Cursor pointer at least once.

The Video pointer is not reset until after **FLYBK** makes a transition from HIGH to LOW (the end of the flyback period). This allows the Video pointer to be used as a refresh address register during flyback. The initialisation takes 250ns and occurs provided that the following conditions are met:

(i) **FLYBK** has made a transition from HIGH to LOW (signalling the end of flyback), and the Video pointer has not already been initialised since **FLYBK** made the transition.

It may take up to 250ns for MEMC to synchronise and process the HIGH to LOW transition of the **FLYBK** signal.

(ii) The processor is performing a normal memory access (not an S-cycle), but is NOT writing to the *DMA Address Generator* or the *MEMC Control Register*.

(iii) No DMA operation is in progress.

(iv) No refresh operation is in progress.

The delay between **FLYBK** going LOW and the first Video DMA being processed must be long enough to allow the Video pointer to be reset.

6.8 Video Controller (VIDC) Interface

To program the VIDC Video Controller, the processor performs a write operation anywhere in the Video Controller address space while MEMC is in Supervisor mode. The Video Controller register number and data are encoded entirely in a 32-bit word which is available on the processor data bus during the write operation (see the **VIDC Datasheet** for more detail). MEMC provides a Video Controller Write signal, $\overline{\text{VIDW}}$, that may be used to latch the information off the data bus.

6.9 I/O Controller Interface

I/O Controllers use a handshaking system to synchronise I/O peripherals with the system data bus. When the processor accesses the I/O Controller address space (while MEMC is in Supervisor mode), MEMC starts an IO cycle by driving the IO cycle request line, $\overline{\text{IORQ}}$, LOW and holding the processor clocks (stretching the processor cycle when **PH2** is HIGH). The I/O Controller signals that it is ready to end the IO cycle by driving the IO cycle grant line $\overline{\text{IOGT}}$, LOW. The IO cycle terminates after both $\overline{\text{IORQ}}$ and $\overline{\text{IOGT}}$ are seen LOW on the rising edge of **REF8M**, with MEMC driving $\overline{\text{IORQ}}$ HIGH and releasing the processor clocks, and the I/O Controller driving $\overline{\text{IOGT}}$ HIGH on the next falling edge of **REF8M**.

An IO cycle is shown if Figure 10. The cycle starts with $\overline{\text{IORQ}}$ being taken low. There then follows a number of 8MHz clock ticks until the I/O Controller is in a position to complete the cycle. The $\overline{\text{IOGT}}$ line is taken low, and both MEMC and the I/O controller see $\overline{\text{IORQ}}$ and $\overline{\text{IOGT}}$ LOW on the rising edge of **REF8M**, so the IO cycle terminates on the next falling edge of **REF8M**.

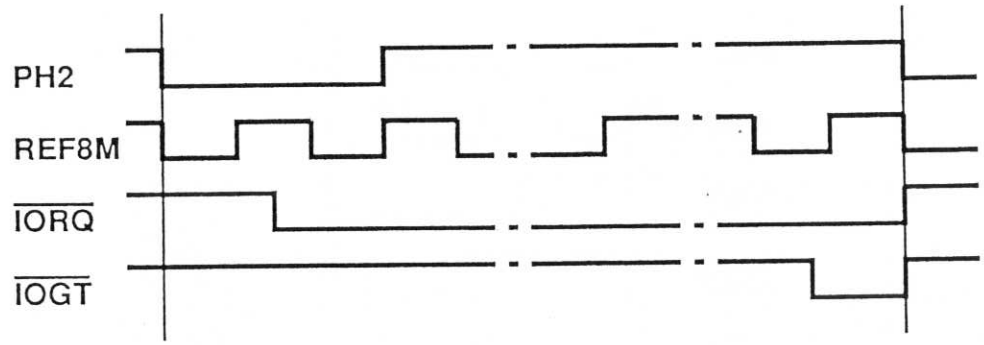


Figure 10: IO Cycle

IO cycles may be interrupted by DMA and refresh operations, as shown in Figure 11. If a DMA or refresh operation is pending, the $\overline{\text{IORQ}}$ signal is driven HIGH when **REF8M** next goes LOW. The DMA/refresh operation may then begin, and when it completes, the IO cycle is resumed by setting $\overline{\text{IORQ}}$ LOW (provided no more DMA or refresh operations are pending). The **DBE** line is always driven LOW during DMA/refresh operations to disable the processor data bus drivers.

NOTE: The I/O Controllers must not drive the data bus when $\overline{\text{IORQ}}$ is HIGH.

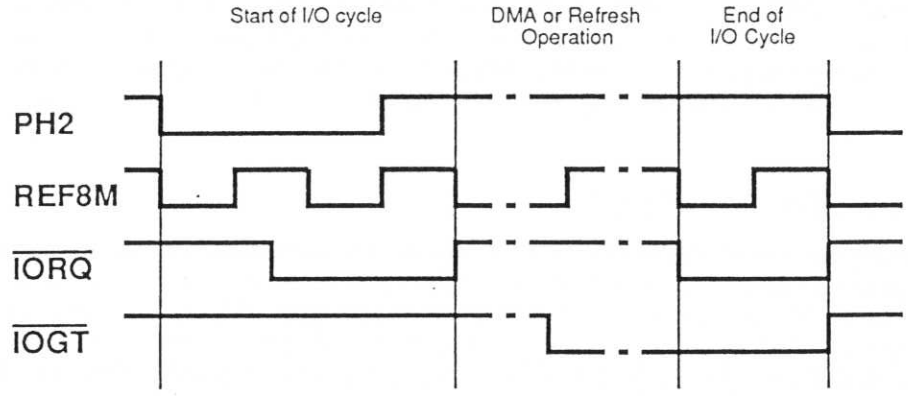


Figure 11: IO Cycle interrupted by a DMA or refresh operation

Some IO cycles may only take 250ns as shown in Figure 12. To give the Input/Output Controller adequate time to recognise such operations, MEMC produces the first $\overline{\text{IORQ}}$ early in the IO cycle.

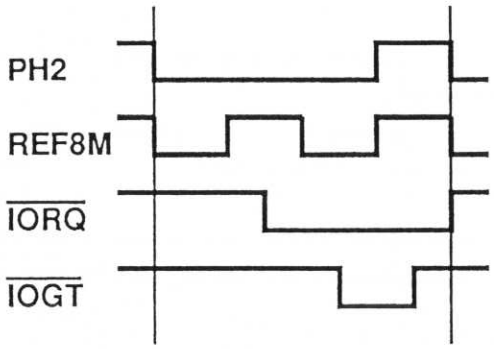


Figure 12: Fast IO cycle

The extension of $\overline{\text{IORQ}}$ only happens at the start of an IO cycle; if the $\overline{\text{IORQ}}$ signal is removed during a DMA or refresh operation, it will be reasserted when **REF8M** goes LOW.

NOTES: Care must be taken not to address a non-existent Input/Output Controller, as MEMC will hold the processor clocks indefinitely until a LOW is seen on the $\overline{\text{IOGT}}$ line, or **RESET** is set HIGH.