

# Application Note **29**

## Interfacing a Memory System to the ARM7TDM Without Using AMBA



Document Number: ARM DAI 0029A

Issued: December 1995

Copyright Advanced RISC Machines Ltd (ARM) 1995

All rights reserved

---

## Proprietary Notice

ARM, the ARM Powered logo, EmbeddedICE are trademarks of Advanced RISC Machines Ltd.

Neither the whole nor any part of the information contained in, or the product described in, this application note may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this application note is subject to continuous developments and improvements. All particulars of the product and its use contained in this application note are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties or merchantability, or fitness for purpose, are excluded.

This application note is intended only to assist the reader in the use of the product. ARM Ltd shall not be liable for any loss or damage arising from the use of any information in this application note, or any error or omission in such information, or any incorrect use of the product.

## Change Log

Issue	Date	By	Change
A	Dec 95	AMP/PN/AP/EH	Created

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

---

## Table of Contents

1	Introduction	2
2	ARM7TDM Bus Fundamentals	3
3	Overview of an Example Memory System	9
4	Conclusion	20



# Interfacing a Memory System to the ARM7TDM Without Using AMBA

---

## 1 Introduction

This application note describes how to build a typical SRAM memory system around the ARM7TDM, without using AMBA.

There are many possible ways of doing this. Different approaches should be used, depending on the size (width) of the memory system, the type of memory to be used, and whether this is on or off chip. When deciding on the memory system to use, cost and system performance must be balanced.

This application note is divided into two major sections.

- A description of the ARM7TDM bus interface. All major facets of bus interfacing are described from clocking strategies, through bus configuration and timing, to memory access control.
- A detailed example, showing how a typical memory system may be designed. The example assumes a little-endian system, using only SRAM and ROM.

ARM has developed a bus architecture called AMBA, the use of which will improve expandability, greatly ease system design, and help testing—particularly in applications that require multiple bus masters.

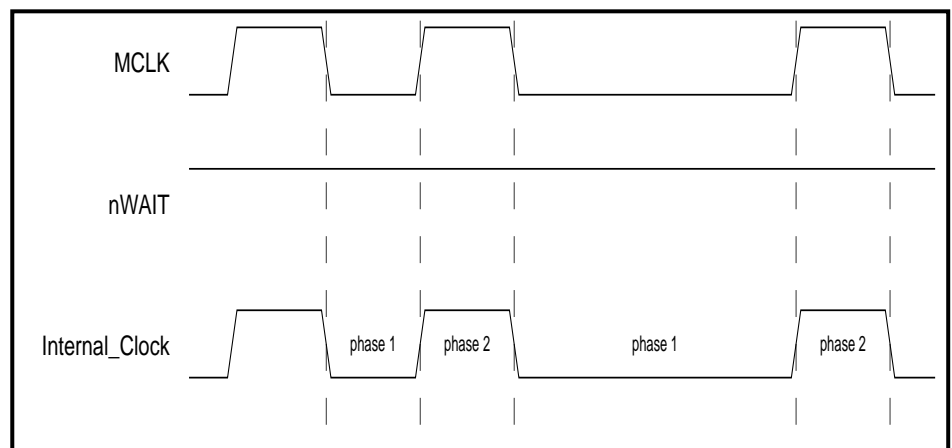
ARM recommends the use of AMBA for the design of systems based around the ARM. However, this application note explains how to interface the ARM7TDM to a memory system without using the AMBA methodology.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

## 2 ARM7TDM Bus Fundamentals

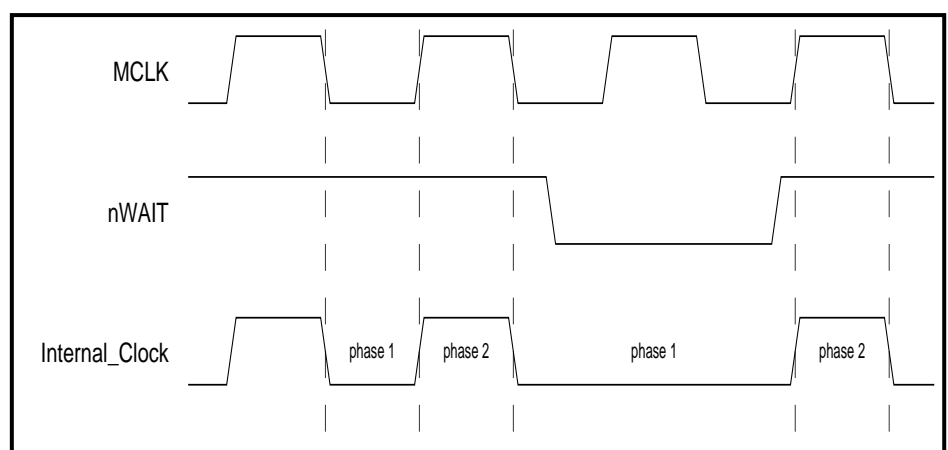
### 2.1 Bus clocking

There are two relevant signals, **MCLK** and **nWAIT**. **MCLK** times all processor activity in normal modes of operation. The static nature of the ARM allows external circuitry to stretch either phase of the clock for slow peripheral accesses. A typical scheme would be to employ an external PAL fed with a 40 MHz clock which generates a 20 MHz clock, stretched as required. This technique may prove difficult to use in some designs. This phase stretching is shown in **Figure 1: Phase stretching**.



**Figure 1: Phase stretching**

An easier and more common alternative to having a high frequency clock within the system is to use **nWAIT** in conjunction with **MCLK**. **nWAIT** is ANDed internally with **MCLK**, so to avoid truncated phase2 (high) cycles, **nWAIT** can only be changed safely during the clock low period (phase1). As a result, the **nWAIT** technique can only be used to stretch the clock low period. This technique is shown in **Figure 2: nWAIT technique**.



**Figure 2: nWAIT technique**

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

## 2.2 Data and address buses

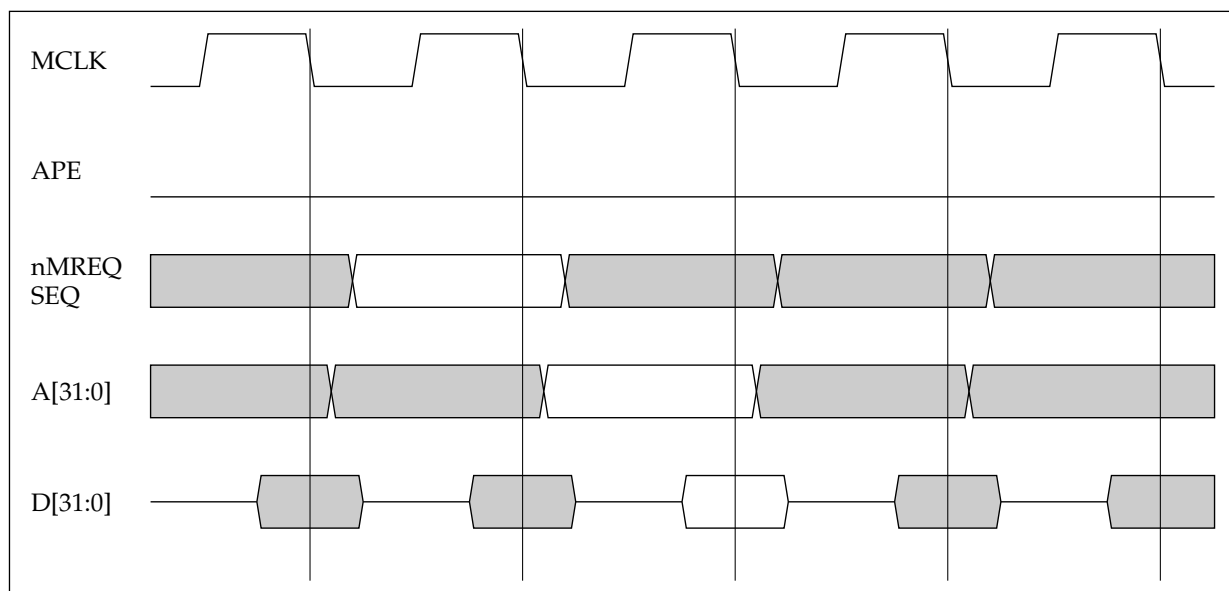
### Bus timing

The ARM address bus can operate in a pipelined or depipelined configuration. This is controlled via the **APE** signal.

**APE** = 0                      depipelined addresses

**APE** = 1                      pipelined addresses

For interface to SRAM or ROM systems, **APE** should be tied permanently LOW ensuring that the address is stable throughout the memory cycle. Memory systems using DRAM, however, can benefit from the earlier valid address timing available when **APE** is HIGH. Address timing using **APE** is shown in **Figure 3: De-pipelined addresses**.



**Figure 3: De-pipelined addresses**

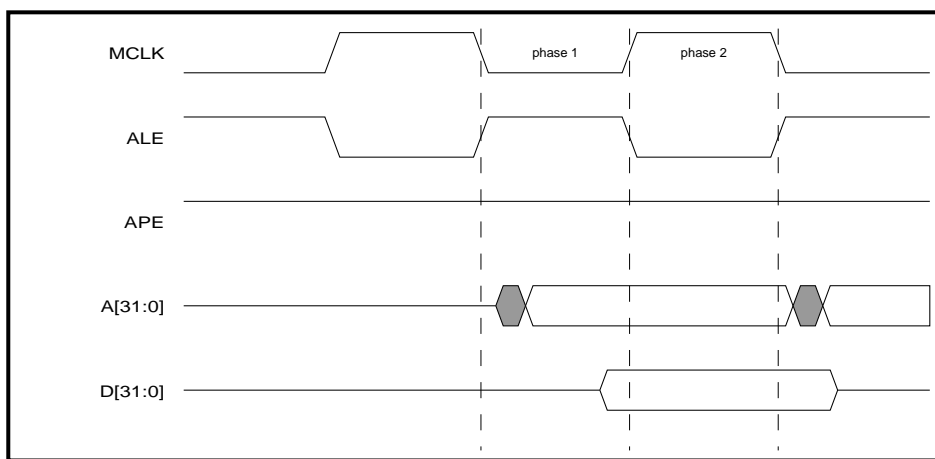
# Interfacing a Memory System to the ARM7TDM Without Using AMBA

## Bus control signals

**ALE:** Address Latch Enable.

This signal controls transparent latches on the address bus. **ALE** allows the ARM7TDM to be interfaced to ROM and SRAM. Holding **ALE** LOW stretches the valid period of the address. Addresses can only therefore change with **ALE** HIGH. Note that **APE** offers a more convenient mechanism to address ROM and SRAM on the ARM7TDM and the use of **ALE** is not recommended.

This timing is shown in *Figure 4: Use of ALE*.



*Figure 4: Use of ALE*

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

## 2.3 Memory access control

### Memory access control signals

**nMREQ:** Memory request.

This is an active LOW signal which indicates a memory access in the following cycle.

**SEQ:** Sequential Addressed Access.

This is an active HIGH signal which indicates that the address used in the following cycle is either the same as or one operand greater than the current address. The indicator is applicable to both word and halfword accesses.

**nRW:** Not Read/Write.

This signal differentiates between memory read and write accesses.

**LOCK:** Locked Operation.

This signal indicates that a swap instruction is in progress, and that the following two cycles are indivisible.

**MAS[1:0]:** Memory Access Size.

The core allows byte, halfword and word accesses. The size of the transfer is determined by the **MAS[1:0]** pins as follows:

MAS[1:0]	Transfer Size
00	Byte
01	Halfword
10	Word
11	Reserved

**Table 1: MAS[1:0] transfer size**

Note that data can be any size and the processor always produces a byte address, but instructions are either word (4 bytes) or halfwords (2 bytes). When word instructions are fetched, address bits **A[1:0]** are undefined, and when halfword instructions are fetched, **A[0]** is undefined.

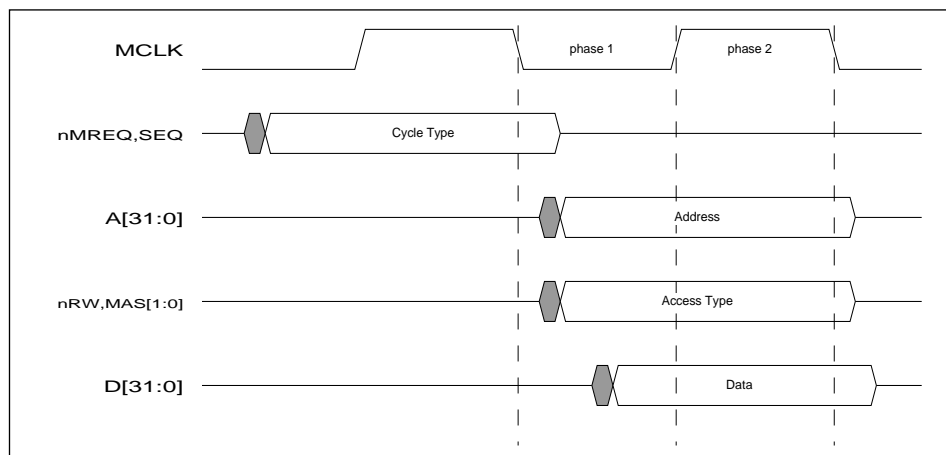
When data reads are performed on byte and halfword data, the memory system controller may ignore the sizing information and present the whole word. The ARM7TDM then extracts the correct byte or halfword from the data.

When byte or halfword writes occur, the ARM7TDM broadcasts the byte or halfword across the whole of the bus. The memory system controller must then examine address bits **A[1:0]** to enable writing to the addressed byte or halfword.



# Interfacing a Memory System to the ARM7TDM Without Using AMBA

The timing of these memory access control signals is shown in **Figure 5: Memory access control signal timing**.



**Figure 5: Memory access control signal timing**

Having examined the memory access control signals, it is now possible to provide a classification of the various memory cycles types. These can be identified by the state of the signals **nMREQ** and **SEQ**. The result of enumerating and classifying the combinations is shown in **Table 2: Memory cycle types**.

nMREQ	SEQ	Next Cycle
0	0	nonsequential (N)
0	1	sequential (S)
1	0	internal (I)
1	1	coprocessor (C)

**Table 2: Memory cycle types**

non-sequential bus cycle	is an access to a memory location whose address is unrelated to the address used in the preceding cycle
sequential bus cycle	is an access to or from an address which is either the same as the address in the preceding cycle, or is one word or halfword after the preceding address.
internal cycle	indicates that the processor is not requesting an access, and is performing some internal function
coprocessor cycle	indicates communication via the data bus with a coprocessor, but requires no action from the memory system.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

---

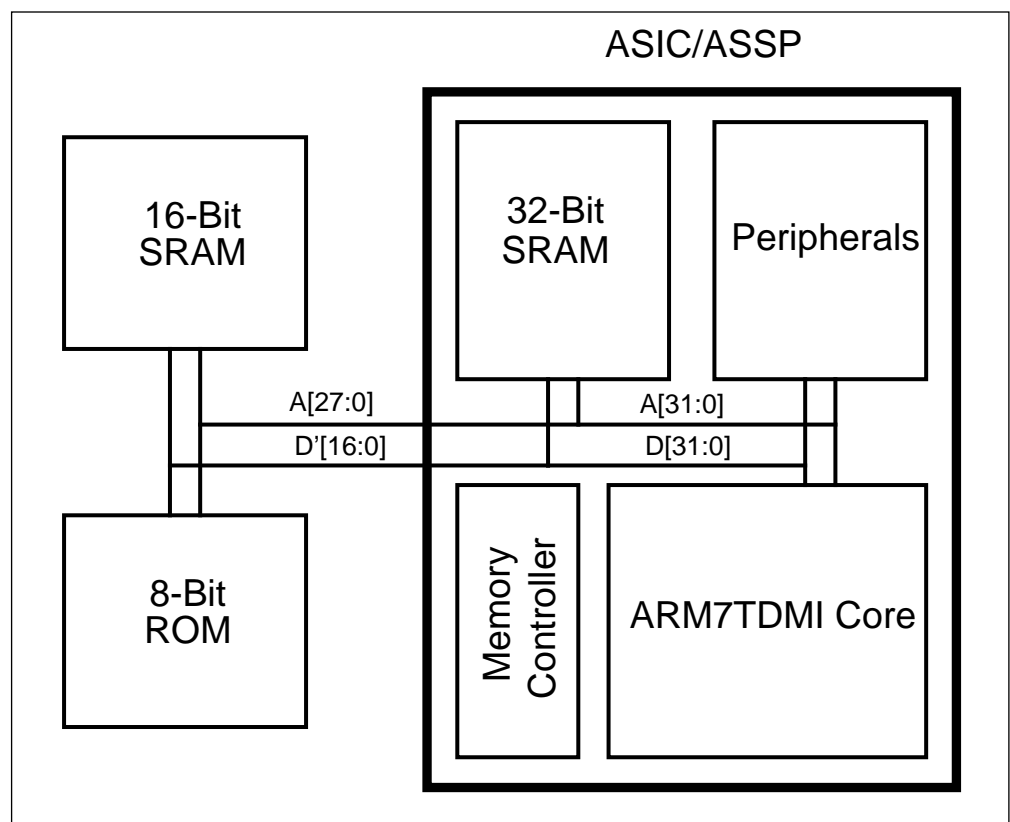
## 2.4 Interfacing ARM7TDM to 16-bit or 8-bit memory

ARM7TDM has architectural features that allow it to be successfully interfaced to 32, 16 or 8-bit wide memory systems. Interfacing to 32-bit wide memory is straightforward; how the processor may be interfaced to sub32-bit memory systems is more interesting. A memory controller can use the **MAS[1:0]** signals to control the number of fetches to or from the memory system. The ARM7TDM has internal latches to allow 32-bit instructions to be built up from sequential fetches to sub32-bit memory. The byte latch strobes **BL[3:0]** control these latches. When the appropriate byte lane strobe is HIGH, the data is latched for that byte on the falling edge of **MCLK**. The use of **nWAIT** does not affect the operation of the byte lane strobes.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

## 3 Overview of an Example Memory System

A typical memory system (see **Figure 6: Typical memory system**) consists of a small amount of fast on-chip 32-bit SRAM, usually located at the bottom of the ARM memory map. This is ideal to provide the performance required to handle exceptions (especially the fast interrupt, FIQ). In addition to the on-chip SRAM, there may be a larger amount of off-chip SRAM, only 8 or 16 bits wide to save system costs. Also, there will usually be an area of ROM holding the application software. This may only be 8 bits wide. A system such as this, utilizing only SRAM and ROM, may have the **APE** signal tied LOW, such that the address pipeline is disabled. The timing presented by this configuration is suited to these types of memory devices.



**Figure 6: Typical memory system**

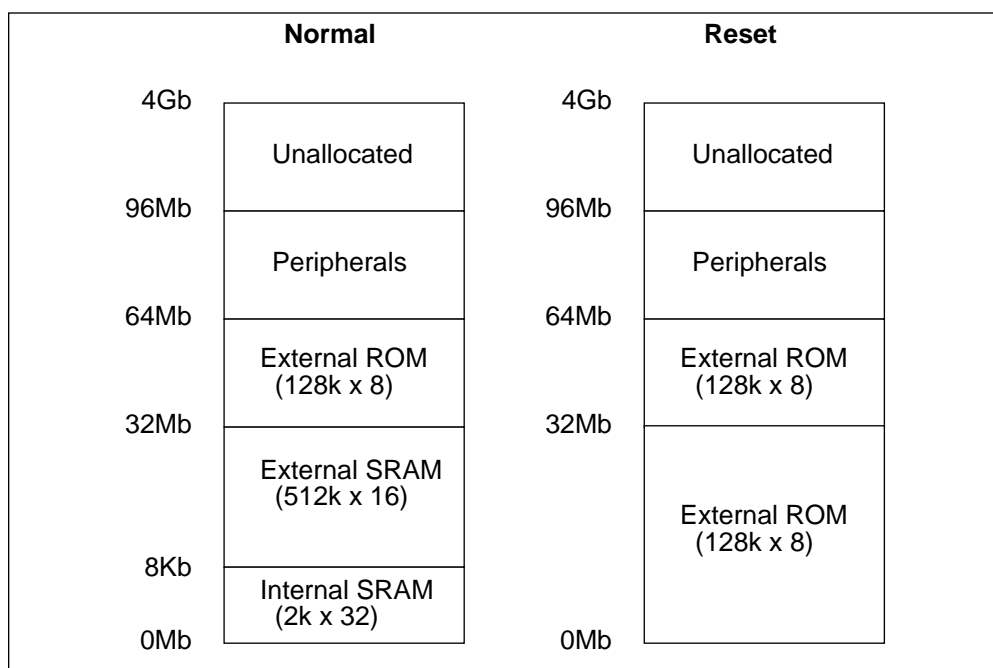
The three different memory types described above will all have different performance characteristics. ROM is essential for a stand alone product, but generally suffers from poor access speeds. This will be exaggerated by the narrow size of the ROM, as two accesses are required to read one Thumb instruction from an 8-bit wide device, and four to read one ARM instruction.

To get round the problem of slow ROM, the initialization routines could copy the application from the ROM into the 16-bit wide off-chip RAM. The RAM will generally be faster than the ROM, and the number of accesses required will be halved due to the increased bus width.

## Interfacing a Memory System to the ARM7TDM Without Using AMBA

Performance critical routines benefit from being written or compiled in ARM code, rather than Thumb code (assuming that 32-bit memory is available). The fast 32-bit on-chip SRAM allows these routines to run at the maximum possible speed, for the best performance. Placing this RAM at the bottom of the ARM memory map allows exceptions to benefit from the performance, because any exception will branch through the vector table, in ARM state. Indeed, the FIQ handling routine can be placed at the top of the vector table, saving the need for a further branch.

An example ARM memory map is shown in **Figure 7: Memory map**. Note that the areas of the memory map are not necessarily fully decoded. For example, the external ROM may only be 128Kb, but will be multiple-addressed in the 32 to 64 Mb range.



**Figure 7: Memory map**

Using noncontiguous memory can be an advantage, because sequential memory access can be assumed to be accessing the same memory device(s) and as such, the decode time for the *chip select* signals need not be considered. This allows a faster memory access cycle. Any nonsequential memory accesses (such as branches) could indicate different memory devices will be selected and hence additional time may be required to allow for the decoding of *chip select* signals. This could result in the necessary addition of a wait state.

If contiguous memory were used, the memory controller/decoder would need to know the address of the boundaries, so that additional time can be added to sequential memory accesses that cross these boundaries. The additional logic required to check for this may even slow the system sufficiently that wait states will need to be added for every cycle. Alternatively, every memory access could be considered nonsequential and the decoding time of the chip select signals added to every memory cycle. This may also result in reduced performance due to added wait states.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

---

In the example, however, the internal 2k x 32 internal SRAM overlays the external RAM allowing code or stacks to flow from internal memory to external memory, if necessary. Additional logic in the memory controller can check for the last memory location in internal memory, and force a nonsequential memory access on the following access cycle.

One disadvantage of having RAM at the bottom of the memory map, is that the exception vectors (in particular the Reset vector) will not be set up at power-up. For this reason, a *reset* memory map is specified, where the ROM is mapped into the bottom of memory (in addition to its usual memory location). This should contain a set of initialization vectors, which are used after reset. The Reset vector should branch to the real (normal) ROM area, where the initialization code will begin. Some mechanism (such as a write to a specific memory location) can be used to revert the memory map to normal, and the required exception vectors can then be copied into the RAM for use.

## 3.1 32-bit Internal SRAM

Interfacing a small amount of on-chip SRAM is relatively straight-forward, and is not discussed in detail here.

The SRAM is usually very fast, and benefits further from not incurring delays introduced by I/O pads when driving off-chip. However, as it may be a precious resource, it should be used wisely for high performance code or frequently accessed variable storage.

If the RAM is placed at the bottom of the memory map, some mechanism must exist to provide the ARM with a valid Reset vector after power-up. Therefore, either an internal or external area of ROM must be mapped over the RAM, and the RAM must be temporarily disabled. During initialization, the RAM can be restored, and valid exception vectors copied into the vector table at the bottom of the RAM.

Alternatively, the reset vector may be loaded via the JTAG interface, eliminating the requirement for ROM.

An important point to remember is that the memory controller must be able to identify sub-word writes to the SRAM, and ensure that the byte write strobes (usually called **BWE[3:0]**) are only activated for the relevant bytes. There is no need to take specific action on sub-word reads, as the ARM extracts the required byte(s) from the word.

A memory controller will be required to drive the appropriate control signals. This will usually contain a state machine similar to the one shown in **Figure 8: Memory controller state machine** on page 12. Notice that **nMREQ** and **SEQ** are identified to determine whether the next memory cycle will be sequential, nonsequential or internal, and hence whether to allow time for address decoding. This is discussed in a later section.

## 3.2 16-bit wide external SRAM

With 16-bit wide SRAM, each ARM instruction requires two memory accesses; Thumb instructions can be read in a single memory access. However, off-chip RAM is generally slower than the on-chip SRAM, so wait-states may need to be added at high clock speeds. This can be achieved either through the **nWAIT** input (which can only add an integer number of clock cycles) or by modulating the **MCLK** clock through

## Interfacing a Memory System to the ARM7TDM Without Using AMBA

some other logic (where it is possible to change the clock cycle time seen by the ARM by fractions of the clock period, and so is more time efficient). This example will consider the former, simpler method.

The diagram in **Figure 9: 16-bit RAM arrangement** shows the arrangement of external 16-bit SRAM connected to a device containing an ARM7TDM core, an appropriate memory controller and some peripherals.

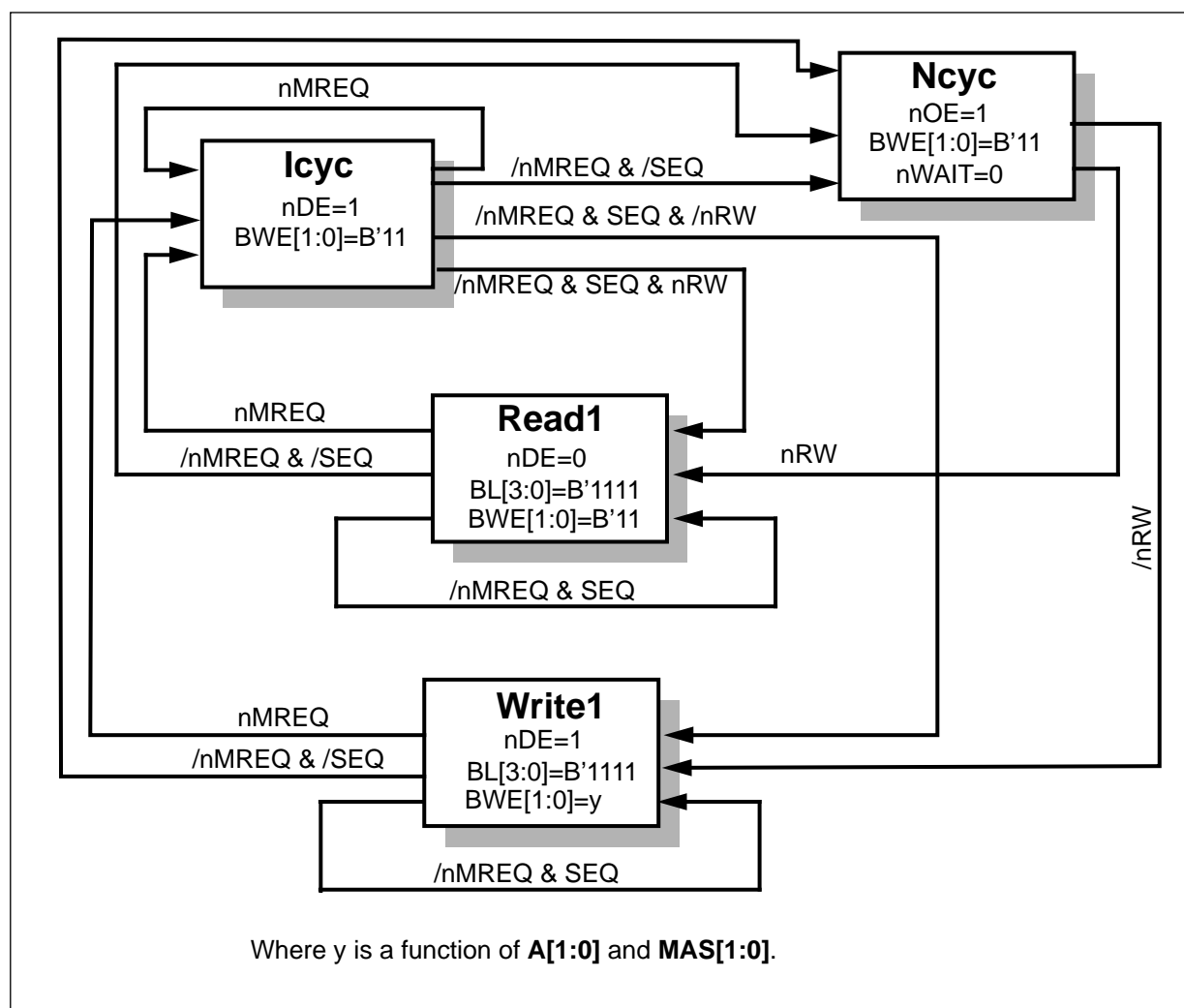


Figure 8: Memory controller state machine

## Interfacing a Memory System to the ARM7TDM Without Using AMBA

The state diagram only shows the case of single cycle memory (32-bit on-chip SRAM). The state machine will need to be extended to allow the slower access times of the off-chip SRAM to be considered. In this example, two clock cycles are required for a sequential access, and one further cycle for a nonsequential access. A separate counter may be used to hold the state machine in the read or write state for the required time, and to drive **nWAIT**.

The state machine will also need to consider **MAS[1:0]** to determine whether one memory access (for byte or halfword) or two accesses (for word) will be required, and hence how long to hold the ARM7TDM while the word is built up.

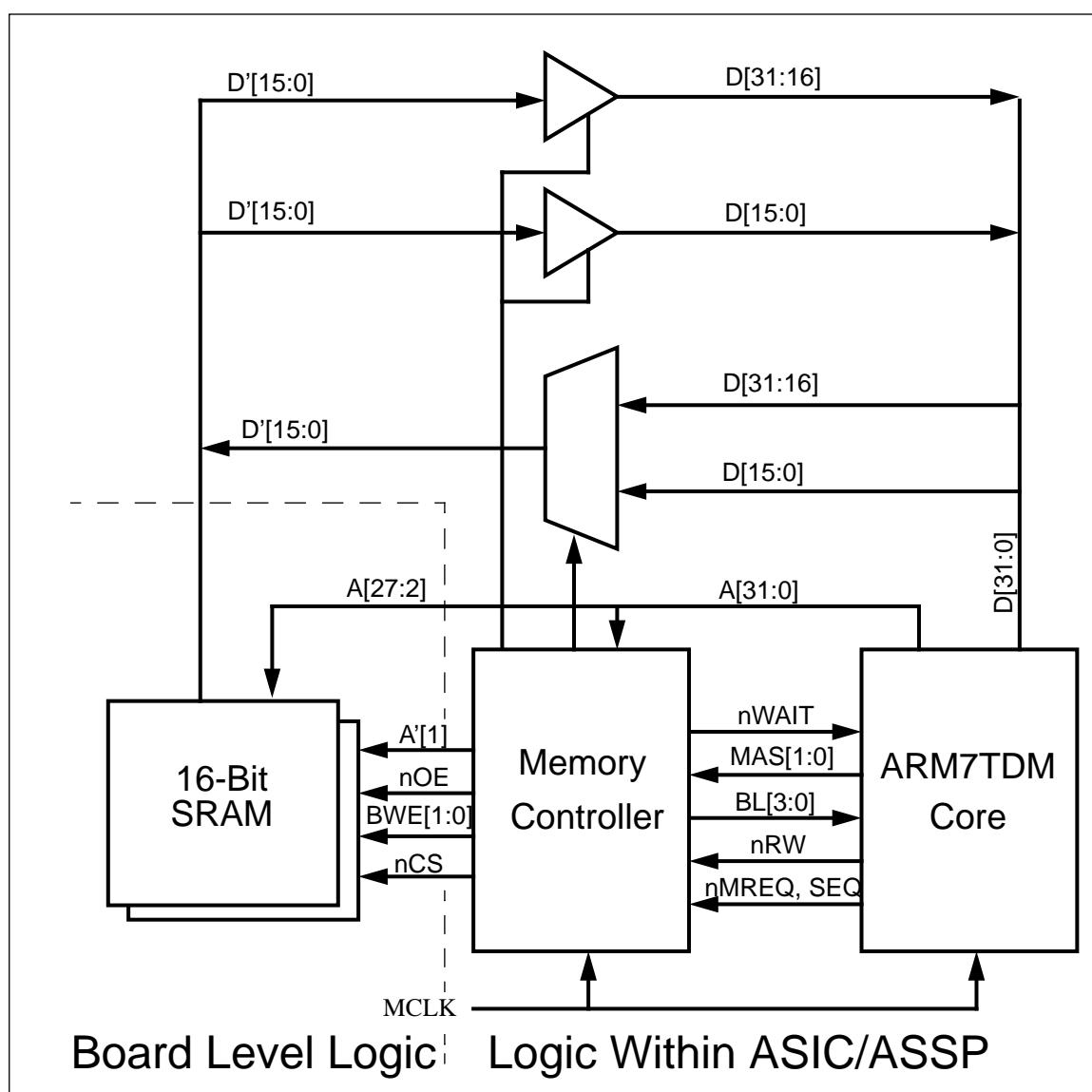


Figure 9: 16-bit RAM arrangement

## Interfacing a Memory System to the ARM7TDM Without Using AMBA

---

Extensions to the state machine to access 8-bit wide ROM are discussed in **3.3 8-bit wide external ROM** on page 19.

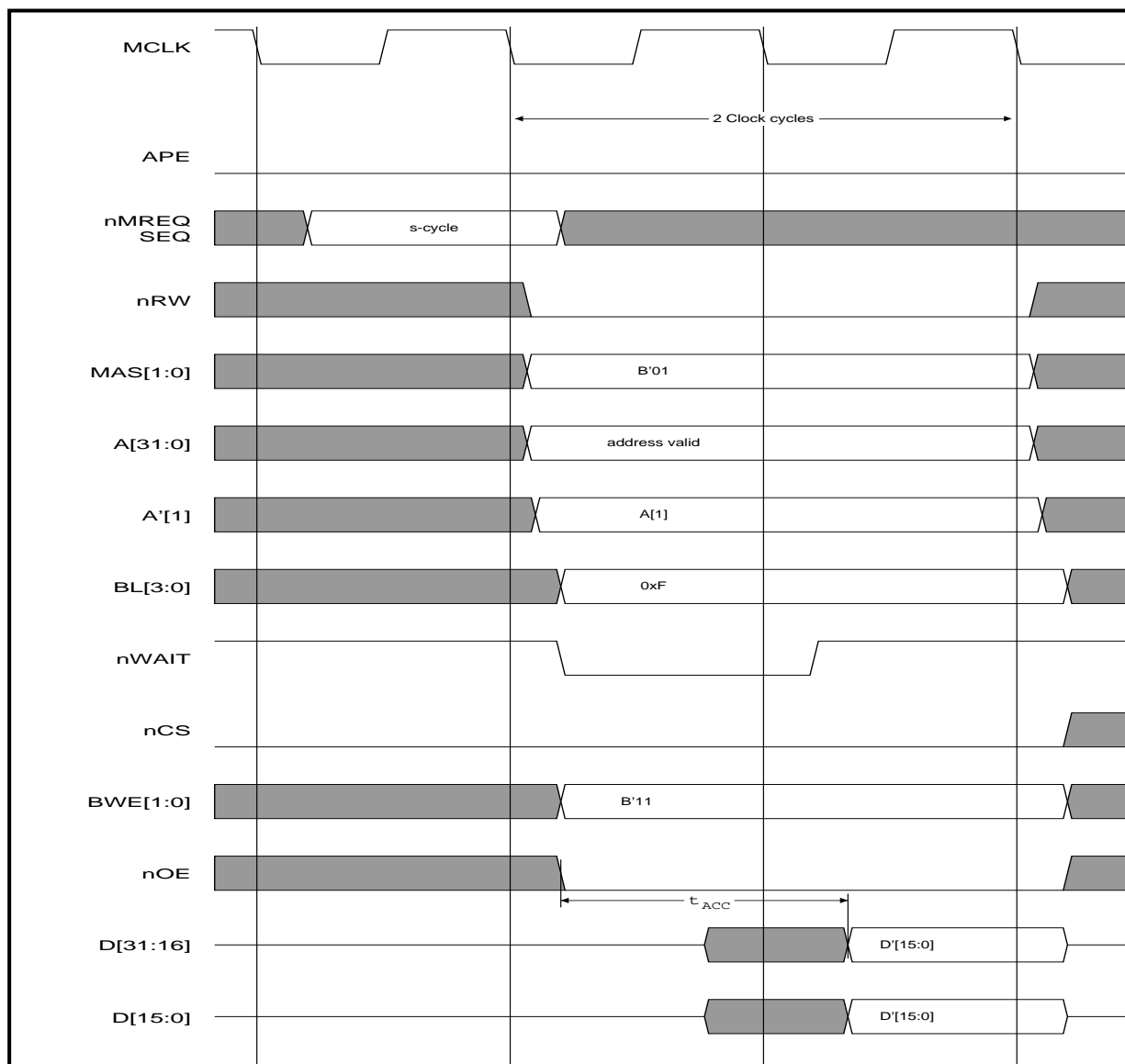
It is important that the memory controller state machine is operating synchronously with the ARM7TDM core, to ensure **nWAIT** is changed within the timing constraints with respect to **MCLK**, ie. while **MCLK** is LOW.

In addition to the memory controller, there is a multiplexer and buffer arrangement on the data bus. These ensure that the correct halfwords or bytes are presented to the narrow external memory (for writes), and that the 16-bit bus from the memory is presented to both the upper and lower halfwords of the ARM7TDM input bus (for reads).

During a 16-bit, sequential read (a Thumb instruction, for example), the memory controller determines from **MAS[1:0]** and **nRW** that a halfword is about to be read. **SEQ** HIGH indicates that the memory cycle is sequential. **Figure 10: Signal timing during a read** shows the timing of the signals during the read.



# Interfacing a Memory System to the ARM7TDM Without Using AMBA



**Figure 10: Signal timing during a read**

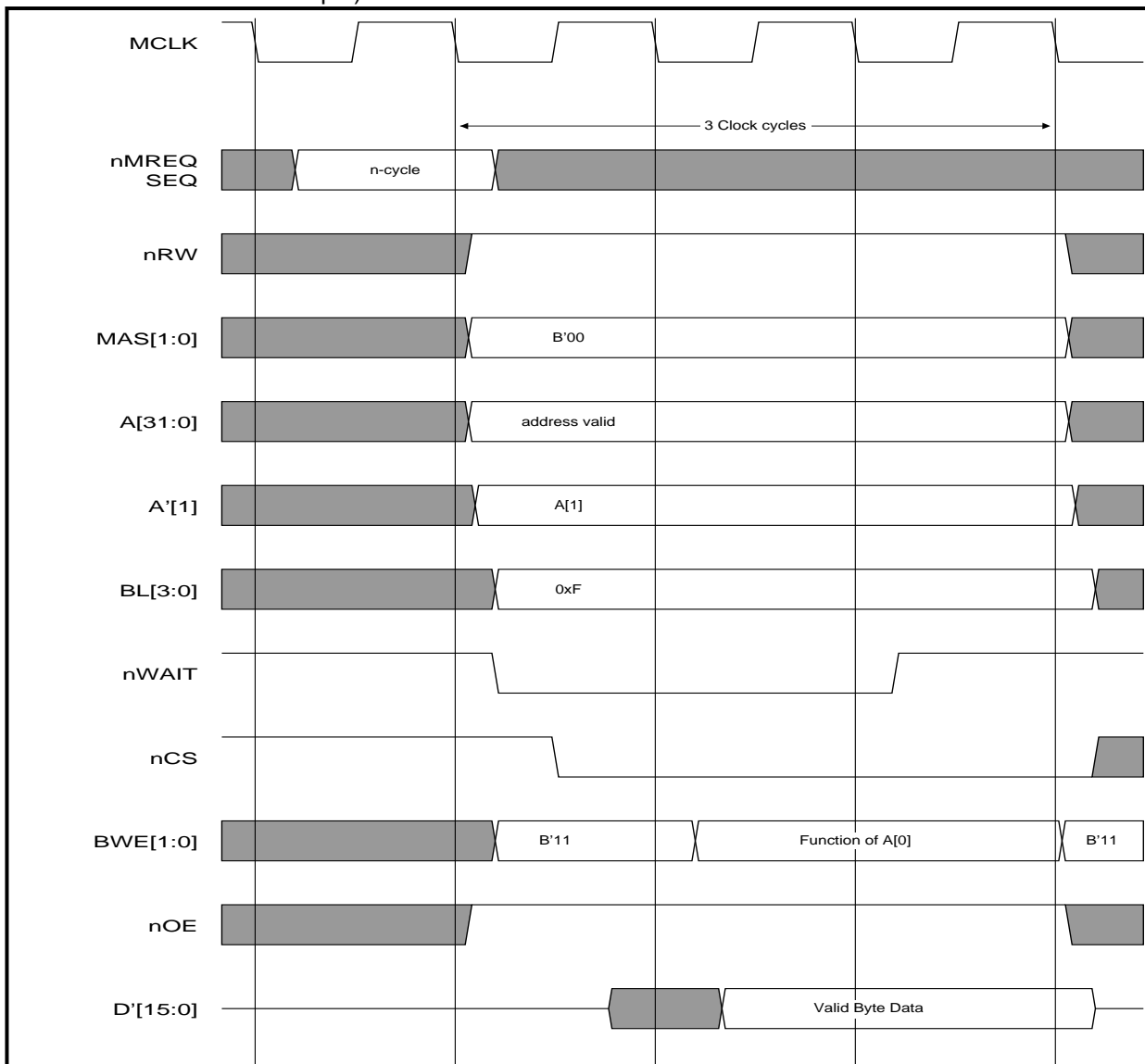
Note the following points:

- The 16-bit data word takes two **MCLK** cycles (one wait state) to be read into the core. The SRAM access time is such that two clock cycles are required. The time taken for address decoding can be ignored in this case, as this is only relevant to nonsequential accesses in this memory system.
- A single **OE** output enable signal is derived, to drive both byte-wide SRAM devices simultaneously. During a byte read, for example, the ARM core will select only the appropriate byte.

## Interfacing a Memory System to the ARM7TDM Without Using AMBA

- Two halfword buffers on the data bus input path drive the same halfword of data onto both halves of the data bus. Instructions are read from alternating halves of the input data bus, which can be determined from the value of address bit **A[1]** and the endianness of the system. It is often more convenient to drive the data onto both halves of the bus, but system power consumption may be increased.
- Address bit **A'[1]** from the memory controller is derived from **A[1]**. **A[1]** cannot be used directly, as the memory controller will need to control this signal for 32-bit memory accesses to the narrow memory system (see later).

**Figure 11: 8-bit nonsequential write** shows the timing diagram for an 8-bit, nonsequential write to the same memory system (caused by a store byte instruction, for example).



**Figure 11: 8-bit nonsequential write**

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

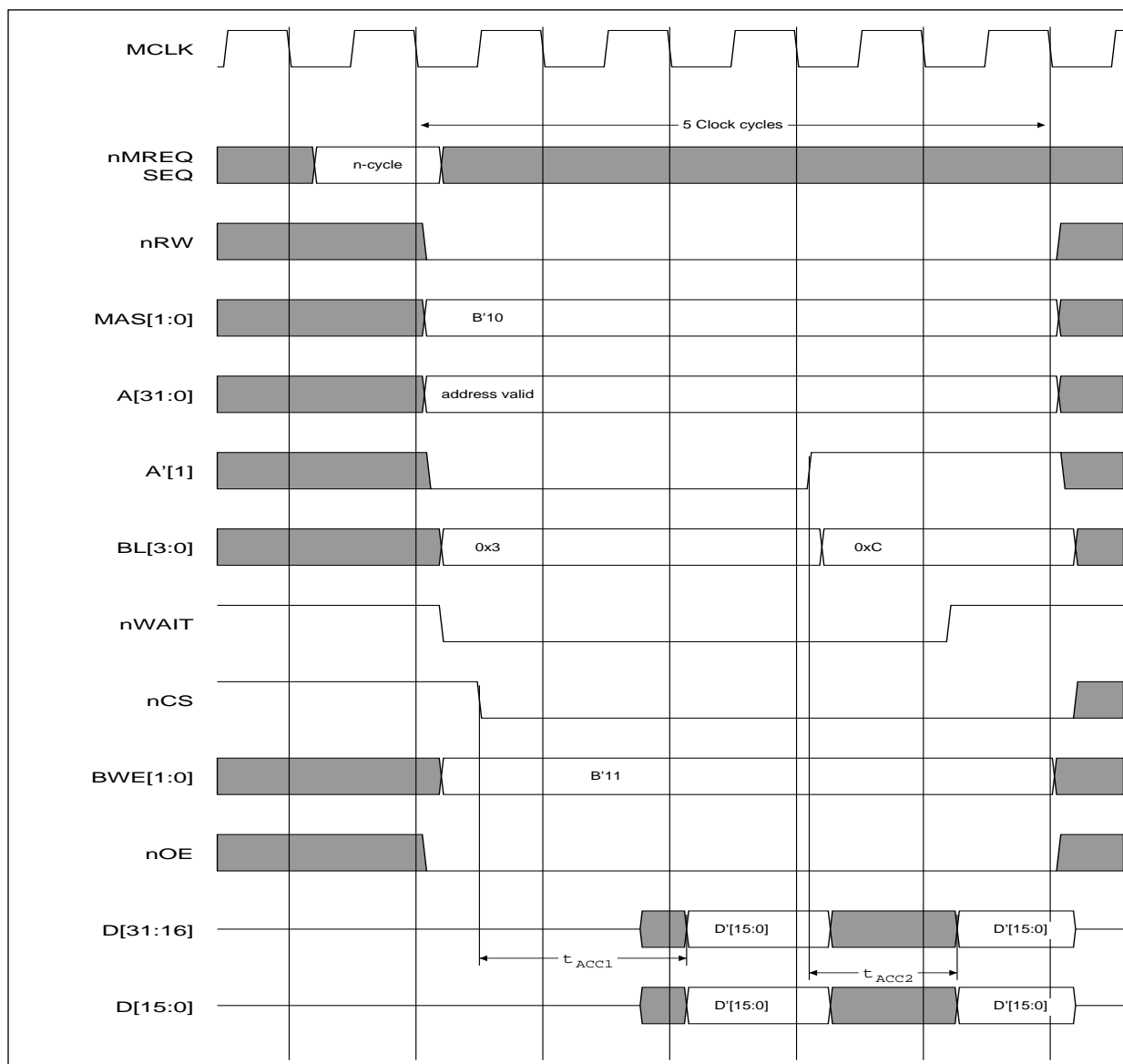
---

Note the following points:

- The 8-bit data word takes three **MCLK** cycles (two wait states) to be written to the memory system. The SRAM access time is such that two clock cycles are required, and the third clock cycle is required to allow for address decoding which would not be necessary for sequential memory accesses.
- The multiplexer is driven by the memory controller (as a function of **A[1]** and the system endianness), which directs the appropriate byte(s) onto the memory devices.
- Two separate Byte Write Enables, **BWE[1:0]**, are used to latch data into the memory. Both of these will be simultaneously active during a halfword write, but only one of these is active during a byte-wide write operation. The active byte write enable strobes are determined by **A[0]** and the size of the data transfer, identified by **MAS[1:0]**.
- The **BWE[1:0]** strobes are also derived from **MCLK** to ensure data setup and hold times are observed. This can be seen in the timing diagram, as the byte write enable strobes going (inactive) HIGH very soon after the falling edge of **MCLK**.

**Figure 12: 16-bit wide memory read** shows the more complex case where a 32-bit value is being read from 16-bit wide memory. This could be due to an ARM instruction being read, or a load into a 32-bit register.

## Interfacing a Memory System to the ARM7TDM Without Using AMBA



**Figure 12: 16-bit wide memory read**

Note the following points:

- MAS[1:0]** denote a word size data transfer. All word accesses must be word aligned, and so the values of **A[1:0]** from the ARM can be ignored (assumed zero). **A'[1]** from the memory controller is initially driven LOW (for a little-endian system). **nWAIT** is driven LOW (active) to add a number of wait states and to stop the core from trying to use the instruction during its assembly.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

- Once again, both halfword buffers on the data bus input path drive the same halfword of data onto both halves of the ARM data bus. During the first half of the memory access cycle, Byte Latch signals **BL[3:2]** are LOW (closed) and **BL[1:0]** are HIGH (open), which allow the lower two bytes into the ARM to be latched on the falling edge of **MCLK**. Note that **MCLK** is used to drive the byte latches before it is gated with **nWAIT** and so will operate the latches even when **nWAIT** is LOW. Data presented to the upper half of the ARM data bus will be ignored, as the latches are closed.
- After the first halfword has been read, the memory controller changes **A'[1]** to HIGH, allowing the memory system to make the upper half of the data word available. **BL[1:0]** will close (LOW) to protect the first halfword already latched, and **BL[3:2]** will open.
- **nWAIT** will go HIGH (inactive) at the end of the second memory access. This will allow the ARM core to finish the memory cycle, and the whole 32-bit word will be latched into the ARM.

32-bit writes to 16-bit memory operates in a similar manner to the read shown in **Figure 11: 8-bit nonsequential write**. The multiplexer drives the appropriate half of the ARM data bus onto the memory devices, and **A'[1]** will increment to ensure the halfwords are written into the SRAM correctly. Use of the byte latch control signals is not applicable during writes.

## 3.3 8-bit wide external ROM

We have shown how a 16-bit memory system can be used to build up 32-bit words in the ARM core, using the byte latch controls, **BL[3:0]**. This concept can be carried further, to build up words or halfwords from a single byte-wide device.

ROM devices are usually very slow, and may require several wait states to be added for a single ROM access. As well as the need to perform up to four ROM accesses to read one word, it is clear that many wait states may need to be added for each memory cycle.

A new sequence of states needs to be added to the memory controller state machine to ensure sufficient delays are introduced for each ROM access. Further, the number of ROM accesses required for each memory cycle needs to be determined from **MAS[1:0]**.

Since the memory controller needs to address four individual bytes in the ROM device, two low order address bits will now be required, **A'[1:0]**. In a little-endian system, these will generally increment from 0 to 3, opening each byte latch from **BL[0]** to **BL[3]** in turn, latching each byte into the ARM. In a big-endian system, the order of bytes will be reversed.

As with the 16-bit example, the data from the ROM can be broadcast across the whole of the ARM data bus. In this case, four byte-wide buffers will be needed to drive the same byte value on each byte lane of the ARM. Multiplexers can be used to allow both 8 and 16-bit memories to use the same ASIC/ASSP data bus, selected by the memory controller.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

---

Obviously there is no requirement to perform writes to the ROM (unless a Flash type system is employed), but this condition should be considered during the design to ensure there is no possibility of a bus clash.

## 4 Conclusion

This application note has highlighted some of the major considerations when designing a memory system around the ARM7TDM core. The points made should be deliberated when building an ARM7TDM based system. The memory controller state machine was also introduced, with pointers on how to extend the outline design to provide support for different types of memory system.

The following issues were covered:

- Use of fast on-chip SRAM for exceptions and high performance routines.
- Substitution of on-chip RAM with ROM, after a system reset.
- Using **nMREQ** and **SEQ** signals to improve the efficiency of address decoding.
- Addition of wait states using **nWAIT**, to allow for slower memory cycles.
- Use of the byte latches, **BL[3:0]** to build up larger operands from bytes or halfwords.
- Using **A[1:0]** and **MAS[1:0]** in the memory controller to generate the appropriate byte write enable (**BWE[3:0]**) strobe.
- The use of buffers and multiplexers to steer the correct bytes into the correct byte lanes between the ARM and the memory system.

# Interfacing a Memory System to the ARM7TDM Without Using AMBA

---



## Application Note 29

ARM DAI 0029A



---

#### **ENGLAND**

Advanced RISC Machines Limited  
Fulbourn Road  
Cherry Hinton  
Cambridge CB1 4JN  
England  
Telephone: +44 1223 400400  
Facsimile: +44 1223 400410  
Email: info@armltd.co.uk

#### **JAPAN**

Advanced RISC Machines K.K.  
KSP West Bldg, 3F 300D, 3-2-1 Sakado  
Takatsu-ku, Kawasaki-shi  
Kanagawa  
213 Japan  
Telephone: +81 44 850 1301  
Facsimile: +81 44 850 1308  
Email: info@armltd.co.uk

#### **GERMANY**

Advanced RISC Machines Limited  
Otto-Hahn Str. 13b  
85521 Ottobrunn-Riemerling  
Munich  
Germany  
Telephone: +49 (0) 89 608 75545  
Facsimile: +49 (0) 89 608 75599  
Email: info@armltd.co.uk

#### **USA**

ARM USA Incorporated  
Suite 5  
985 University Avenue  
Los Gatos  
CA 95030 USA  
Telephone: +1 408 399 5199  
Facsimile: +1 408 399 8854  
Email: info@arm.com

World Wide Web Address: <http://www.arm.com/>