Application Note 88

Considerations when implementing a design that uses an ARM hard macrocell

Document number: ARM DAI 0088A

Issued: March 2001

Copyright ARM Limited 2001

Application Note 88
Design Implementation Considerations of an ARM Hard Macrocell

Release information

The following changes have been made to this Application Note.

Change history

Date	Issue	Change
March 2001	A	Initial release

Proprietary notice

Confidentiality status

Feedback on this Application Note

If you have any comments on this Application Note, please send email to errata@arm.com giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- an explanation of your comments.

General suggestions for additions and improvements are also welcome.

ARM web address

http://www.arm.com

Table of Contents

1 Introduction

This application note explains important issues to consider when you integrate an ARM core hard macrocell into your design. The processor cores covered are:

- ARM7 family:
  - ARM7TDMI
  - ARM710T
  - ARM720T
  - ARM740T.
- ARM9 family:
  - ARM9TDMI
  - ARM920T
  - ARM922T
  - ARM940T.

The application note must be read in conjunction with the appropriate Technical Reference Manual and the *Reference Peripheral Specification* (RPS) (ARM DDI 0062) available at `http://www.arm.com`.

2 System considerations

System considerations described in this application note are:

- Vector table considerations on page 5
- Big-endian or little-endian on page 6
- Implementing memory on page 7
- Implementing peripherals on page 7

2.1 Vector table considerations

This section describes:

- Memory remapping on page 5
- FIQ handler location on page 6.

2.1.1 Memory remapping

You can locate the vector table in either ROM or RAM. You are advised not to locate the vector table in ROM, because it:

- is usually slower than RAM
- requires hardware breakpoints
- cannot be changed dynamically.

When you locate the vector table in RAM you must still be able to provide a valid reset vector after power-up. This is usually achieved by using an alternative “reset memory map” which has the vector table located in ROM. This is implemented by using a “remap register” to change how the system address decoder selects memory devices at the addresses occupied by the vector table. An example memory map is shown in Figure 1.

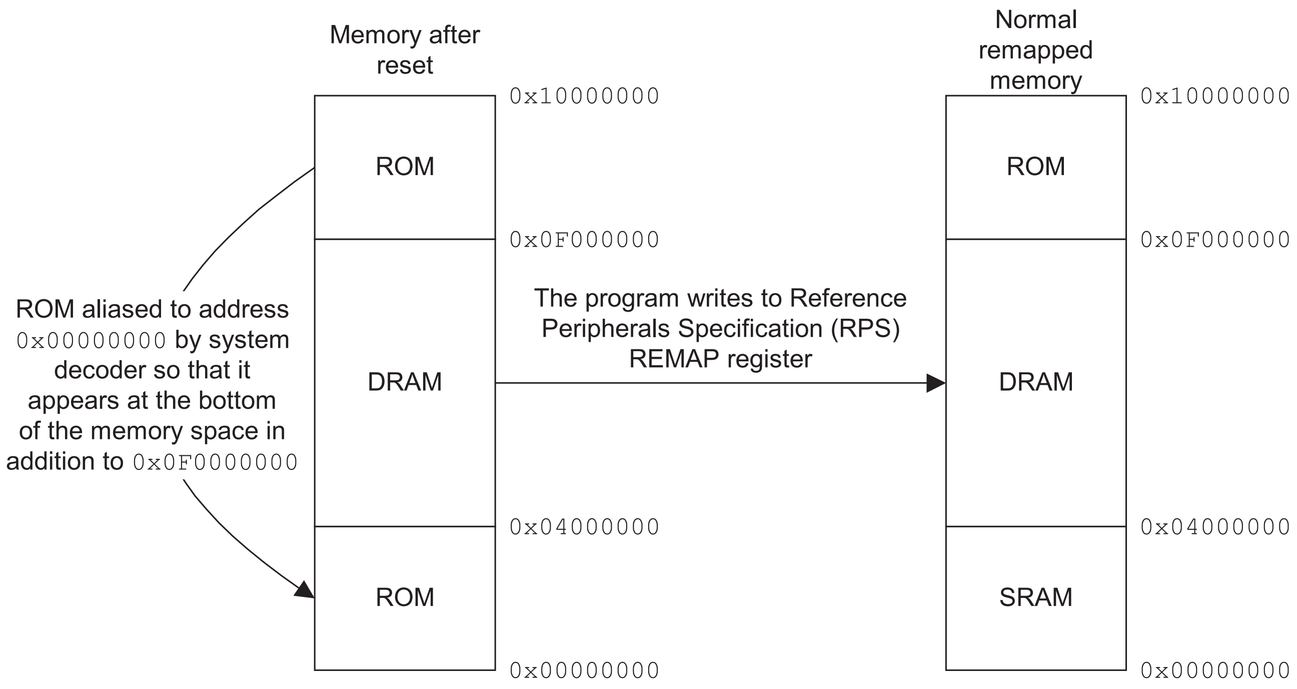
Memory after reset
Normal remapped memory
0x10000000
ROM
0x0F000000
DRAM
0x04000000
ROM
0x00000000
0x10000000
ROM
0x0F000000
DRAM
0x04000000
SRAM
0x00000000
ROM aliased to address 0x00000000 by system decoder so that it appears at the bottom of the memory space in addition to 0x0F0000000
The program writes to Reference Peripherals Specification (RPS) REMAP register

Figure 1 Memory remapping

An example of the HDL code for memory remapping is shown in Example 1.

Example 1 Decoder design in HDL

```
case ADDR(31:24) is
  when "0x00"
    if REMAP = '0' then
      select ROM
    else
      select SRAM
  when "0x0F"
      select ROM
  when .............
```

An example of the sequence to step through as the system powers up is shown in Example 2.

Example 2 System power up

```
Power On
Fetch RESET vector at 0x00000000, this is the alias of ROM
Execute RESET vector
  LDR PC, = 0x0F000004, this is the real address of next ROM
instruction
Write to REMAP register
  set REMAP = '1'
Run exception vector copy routine
  copy slow ROM to fast SRAM
```

2.1.2 FIQ handler location

You are advised to put the FIQ handler on top of the vector table so that it starts at `0x1C`. This improves the performance (assuming that there is fast memory for the vector table) and avoids a branch to access the FIQ handler.

2.2 Big-endian or little-endian

Ensure that you configure the **BIGEND** input signal, where provided, correctly. The states for this signal are listed in Table 1.

Table 1 BIGEND signal states

BIGEND	State
LOW	Little-endian
HIGH	Big-endian

Processors without a **BIGEND** input signal default to little-endian operation. You must write to the system control coprocessor CP15 C1 as part of the initialization code to select big-endian state, if required. Such processors usually have a **BIGEND** output to allow the memory system to be set to the same endian configuration as the application.

Note *Do not change the endian state of the processor outside the initialization routine. Current ARM tools do not support mixed endian systems.*

2.3 Implementing memory

When you use on-chip RAM, you must provide separate write enables for each byte, because if your system supports only word or halfword writes, it is possible for data to be corrupted.

Byte-write enables are required for:

- C structures
- EmbeddedICE.

2.4 Implementing peripherals

The AMBA Design Kit provides:

- a pause and remap controller
- an interrupt controller
- two timers.

When you implement peripherals, you are advised to comply with the *Reference Peripherals Specification* (RPS). This simplifies porting different *Real Time Operating Systems* (RTOS) to target systems.

The RPS specifies the address offsets of registers in peripherals such as timers and interrupt controllers. The base address of each peripheral is system-dependent.

2.4.1 Implementing an interrupt controller

When implementing an interrupt controller you must consider the following:

- It is usual to avoid multiple sources of FIQ. A single source of FIQ reduces the time required to identify the source. Other interrupt sources can be merged onto IRQ.
- If an interrupt source is slow to clear, then the interrupt handler can complete before the interrupt is deasserted, and consequently the interrupt routine could be entered again in error.
- Interrupts are best cleared by a memory access to the interrupt source, such as reading a character out of the UART.

Note ***nFIQ** and **nIRQ** are level sensitive, not edge sensitive.*

Debug communications

It is normally better to have interrupt driven communication over the Debug Communications Channel, in which case you must connect **COMMRX** and **COMMTX** to two inputs of your system interrupt controller.

If you cannot use the interrupt driven communications signals, **COMMRX** and **COMMTX**, you must use a polled communication method. However, this results in lower performance.

Synchronous and asynchronous interrupt sources

ISYNC is an input to the processor. When LOW, **ISYNC** indicates that the **nIRQ** and **nFIQ** inputs are to be synchronized by the ARM core. This increases interrupt latency by one clock cycle.

When HIGH, **ISYNC** disables this synchronization logic. This is useful for inputs that are already synchronous.

Note *For cores that use single or synchronous clock domains, you are advised to perform synchronization in the interrupt controller, outside the core.*

If wait states (using ***nWAIT*** *or* ***BWAIT****) are added then the processor clock is being gated internally to delay state change. This gated clock is also used by the interrupt synchronization logic, which results in a delay to the interrupt being recognized.*

For cores operating from more than one asynchronous clock domain, you are advised to set the core to synchronize the interrupt internally. This ensures that the correct clock is used.

3 Power consumption

To minimize power consumption:

- Do not drive the external bus dynamically unless necessary. Instead, leave it in the last state. If the external bus is used frequently, avoid switching to tristate because switching can consume more power and reduce the operating frequency.

Note *Tristate is required if the external bus is shared with other bus masters.*

- If the clock is slowed down or stopped (sleep mode), make sure that external memory devices are not:
  - left enabled
  - enabled for all of the slow memory access.
- Make sure that when the clock is stopped, memory is disabled.

Note *CMOS consumes power only when clocked, but SRAM consumes power when enabled. Therefore, systems that run at a very low clock speed can consume more power than systems that run at a high clock speed but where the clock is stopped when necessary.*

- If the battery power is low when you try to restart the clock, consider that the following items cause high power consumption:
  - uninitialized logic, such as programmable logic arrays
  - clashing tristate buses.
- If system power consumption is higher than normal (for one of the above reasons), a weak power source might not be sufficient to restart the clock. If the system reset synchronizes the clock then this can mean that the system is never reset. To resolve this, apply **nRESET** asynchronously to the system and remove **nRESET** synchronously.

4 Timing

Design considerations for timing are:

- Check the data-in hold time. A positive hold time is required on some ARM cores. You must check the timing data for the appropriate ARM core and production process.
- For the ARM7TDMI, avoid using the **BL[3:0]** inputs. These cannot be used in AMBA-compliant systems. If you decide to use **BL[3:0]**, check the timing of **BL[3:0]** with respect to **MCLK**. This can affect the data latching point. Latch timing should be dependent on **MCLK**, not on change of **BL[3:0]**. Figure 2 shows how the **BL[3:0]** inputs are used to control the data input latches.

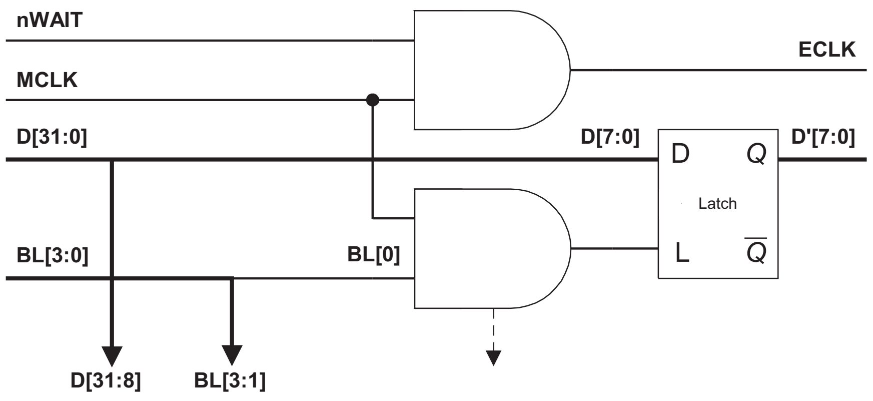

Figure 2 Controlling the timing of BL

- Check worst case clock jitter and stability of the clock generator, which can affect relative clock edges. For hard macrocells the minimum high and low phase times must not be violated. The T_{min} period is often calculated as $T_{min\ high} + T_{min\ low}$.

  The duty cycle on a crystal is often assumed to be 50/50, but can be nearer to 60/40 as shown in Figure 3.

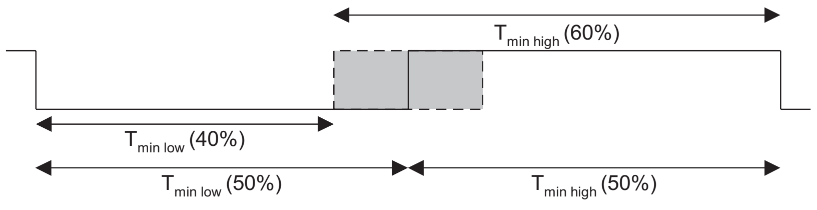

Figure 3 Jitter

- For a 50% duty cycle with a 10ns minimum high and low phase, then the minimum period is 10+10=20ns, providing a maximum frequency (F_{max}) of 50MHz.
- If the duty cycle. Is 40/60 worst case, then for a 20ns period, the worst case phase times will be 8ns and 12ns. This would violate a minimum phase time of 10ns.

The best possible combination is a T_{min_low} of 10ns, which requires a T_{min_high} of 15ns. The equation for this is shown below:

$$T_{min\ low} = 10ns\ (40\%)$$

$$\therefore\ T_{min\ high} = 15ns\ (60\%)$$

$$F_{max} = \frac{1}{(10ns+15ns)}$$

$$F_{max} = \frac{1}{25ns}$$

$$F_{max} = 40MHz$$

5 Test strategies

The test strategies described in this section are:

- Boundary scan testing on page 12
- Parallel testing on page 13
- AMBA testing on page 14.

Full scan insertion is not supported on hard macrocells.

5.1 Boundary scan testing

Boundary scan testing offers a low silicon overhead, but has a number of disadvantages:

- The vectors are serial
- it has a long test time
- the test patterns are complex
- the core must be exercised to provide visibility on the boundary
- it is difficult to debug and track faults.

Scan chains at the core boundary are included in some hard macrocells, such as the ARM7TDMI and ARM9TDMI cores. ARM can supply serialized test vectors for these cores so that you can use the scan chains. For more information contact ARM Customer Support.

Note *ARM does not supply serialized test vectors for cached cores such as the ARM720T and ARM922T cores because of the high numbers of vectors required.*

A boundary scan block diagram is shown in Figure 4. This implements scan chains around the boundary of the test logic only.

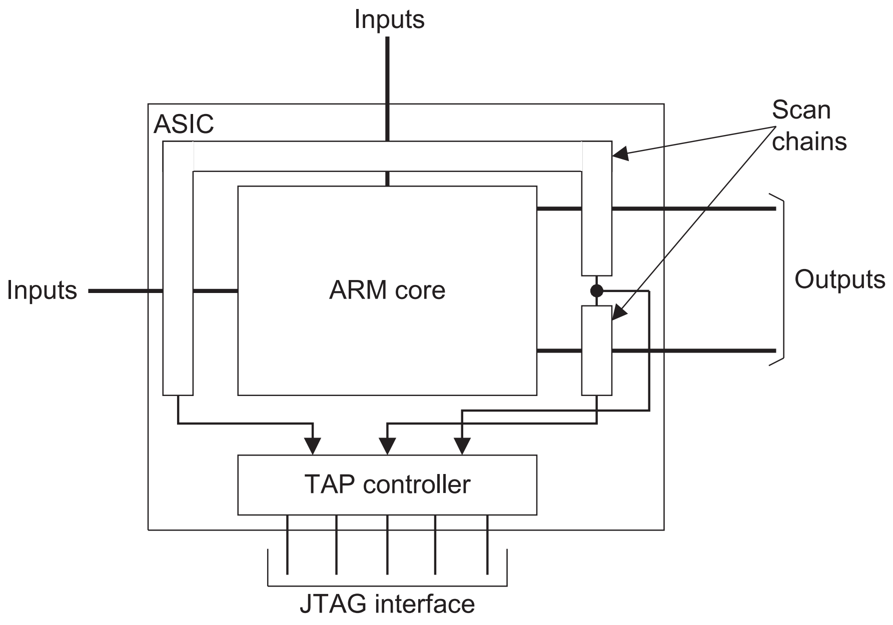

Figure 4 Boundary scan test strategy

5.2 Parallel testing

When you perform parallel testing on an ASIC block or peripheral, multiplexing signals has the advantages that:

- the test time is short
- it is easy to generate and debug test vectors.

They also have the following disadvantages:

- high silicon overhead
- possibility of top level routing problems
- possibility of high pin cost
- difficult to change the design because the multiplexer structure must be changed.

Note *ARM Customer Support can supply parallel test vectors for ARM hard macrocells, but in some cases they cannot be used due to the pin count requirements of the test strategy.*

A block diagram of the test strategy for multiplexing signals is shown in Figure 5.

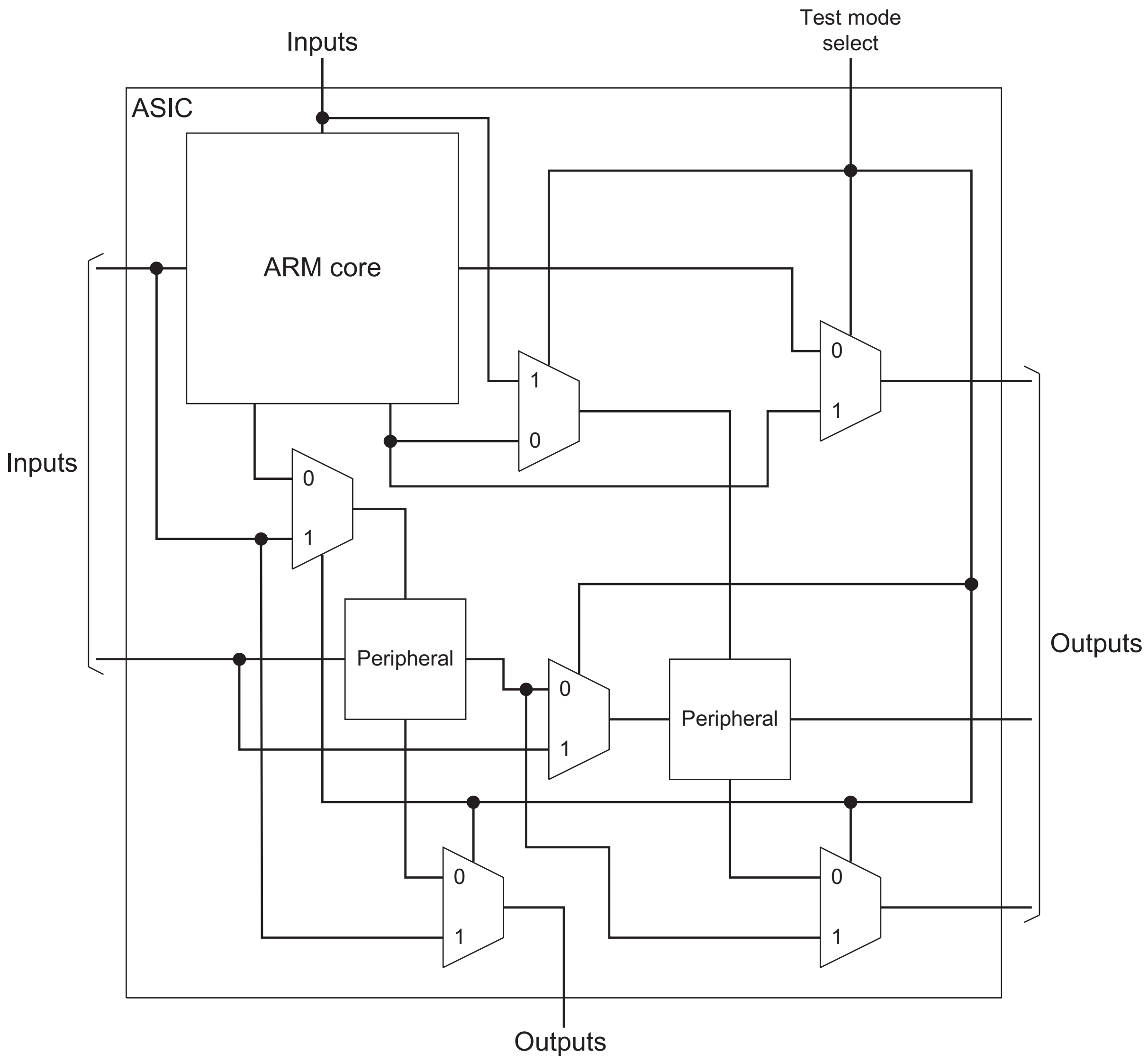

Figure 5 Parallel testing

5.3 AMBA testing

For the AMBA test strategy shown in Figure 6, you have to add only the *Test Interface Controller* (TIC) and test harness around each module. AMBA test strategy advantages are:

- the silicon overhead is moderate
- the test time is short
- it is easy to generate vectors
- there are no top-level buses dedicated to test
- the vectors are portable when peripherals are reused in a new design
- it is easy to debug test vectors.

However, you can only test the functionality, and not the connectivity, of non-AMBA signals.

ARM Customer Support can supply AMBA *Test Interface Format* (TIF) vectors for a wide range of cached and uncached hard macrocells. A block diagram showing an example of AMBA strategy is provided in Figure 6.

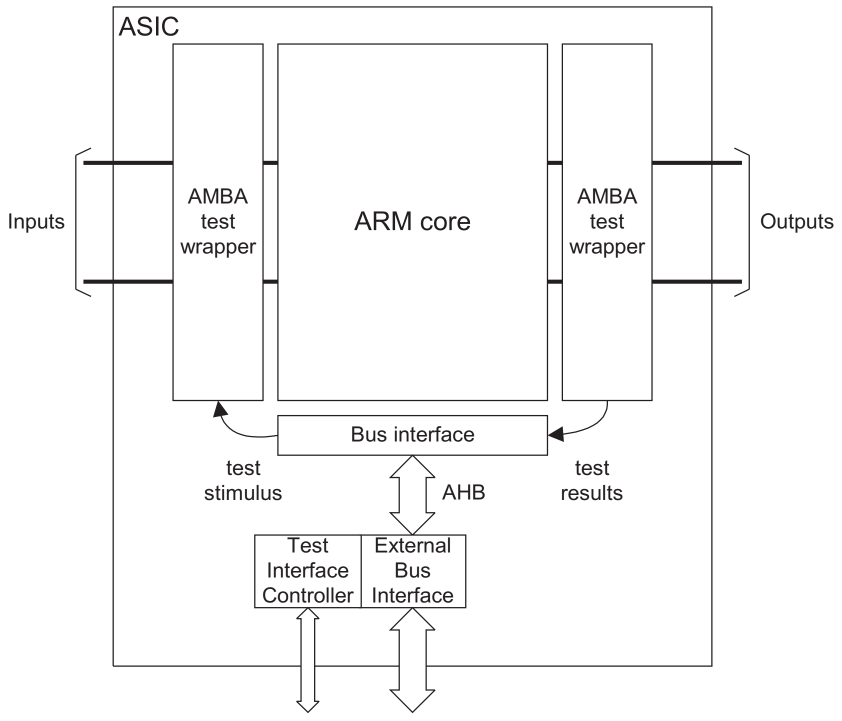

Figure 6 AMBA test strategy

Refer to the *AMBA Specification* (ARM IHI 0011) for further information.

6 Testing

Test design considerations are:

- If device boundary scan is implemented, check that the implementation of the scan cells conforms with *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. Most hard macrocells provide outputs from the TAP controller to facilitate the addition of a boundary scan chain.
- The IEEE 1149.1 standard implies that **nTRST**, **TDI**, and **TMS** must have internal pullup resistors. To minimize static current draw, these resistors are not fitted to the ARM core because this should only be done at the device boundary. Accordingly, the four inputs to the test interface, the **nTRST**, **TDI**, and **TMS** signals, plus **TCK**, must all be driven to good logic levels to achieve normal circuit operation. This is shown for **TDI** in Figure 7.
- The ARM7TDMI core (up to Rev 3a) has no update stages, so signals can be seen to toggle during testing while data is scanned around the scan chain.
- The IEEE 1149.1 standard requires that **TDO** is high impedance when data is not being scanned. This is to allow **TDO** outputs from multiple devices to be bused together. The **TDO** output from ARM cores is always driven, even when no data is being scanned. Figure 7 shows how **nTDOEN** can be used to obtain the required behavior of **TDO** at the device boundary.

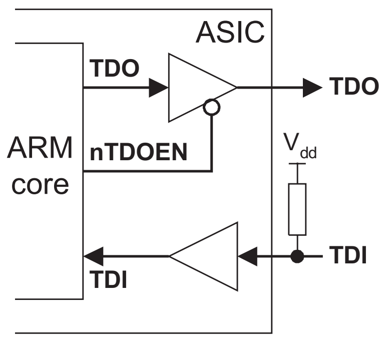

Figure 7 Use of nTDOEN and pullup resistor

7 Non-AMBA systems

Design considerations applicable to non-AMBA systems are:

- The **nRESET** signal must be removed from the core synchronously with the memory controller state machine. If not synchronized, the core and memory controller might detect **nRESET** on different cycles as shown in Figure 8. This will corrupt memory access.

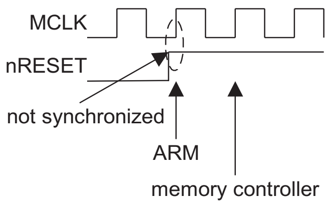

Figure 8 Unsynchronized nRESET signal

- Check the memory controller state machine to ensure that:
  - the merged I-S condition is correct
  - an I-cycle followed by an N-cycle, as occurs after LDR PC, is dealt with correctly.
- Make sure that **nMREQ** and **SEQ** are not sampled when **nWAIT** is low. Unless they are latched, this causes problems in the pipeline assuming that the state machine advances on falling **MCLK**.

  If, for example, a memory system is using **MCLK** but the ARM core is using **MCLK** gated by **nWAIT,** the memory controller state machine will not be synchronized with the ARM core. This is shown in Figure 9 where:
  - At point 1, the state machine samples **nMREQ** and **SEQ** for cycle A.
  - At point 2, the state machine samples **nMREQ** and **SEQ** again, but they now relate to the following cycle (B). This can cause the state machine to disable memory too soon as cycle A is being extended.
  - At point 3, **nMREQ** and **SEQ** are correctly sampled for the next cycle (B).

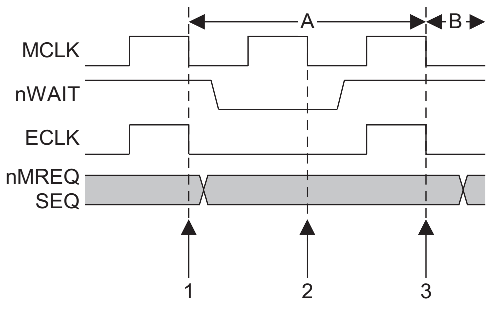

Figure 9 Sampling nMREQ and SEQ

8 Debugging

Debug considerations are:

- If you are debugging using the EmbeddedICE Logic, tie **DBGEN** HIGH.
- If you need to trace the ARM core for debugging, you must do one of the following:
  - add multiplexers to the output pins to allow buried nodes to be observed
  - activate external bus interfaces even for on-chip accesses
  - use the ARM Embedded Trace Macrocell.
- If additional logic, such as external watchpoint units, to increase the debug capability of the EmbeddedICE logic has not been added, **DBGRQ**, **BREAKPT**, and **EXTERN[1:0]** must be tied LOW.