AMULET3H

32-bit Integrated Asynchronous Microprocessor Subsystem

version 1.0
24 November 1999

AMULET3H

32-bit Integrated Asynchronous Microprocessor Subsystem

Features

- AMULET3 core:
  - compatible with the ARM architecture v4T instruction set;
  - support for the ‘Thumb’ 16-bit compressed instruction set;
  - debug hardware generates an interrupt or abort on programmable bus values.
- 8 Kbyte high-speed dual-ported SRAM memory.
- ‘MARBLE’ asynchronous macrocell bus, connecting to:
  - 32-channel DMA controller;
  - 16 Kbyte ROM;
  - bridge to Synchronous On-Chip Bus (‘SOCB’) which provides an interface to standard ‘off-the-shelf’ synchronous peripheral macrocells;
  - Asynchronous Event-Driven Load (‘AEDL’) for synchronizing software to external data rates.
- External memory interface:
  - direct connection to off-chip static and dynamic memory devices;
  - timing programmable using on-chip software-calibrated reference delay;
  - dynamic external bus sizing.
- Zero power idle mode, exited immediately on interrupt request.
- 0.35 micron CMOS, 3 layer metal process.
- Test interface supporting straightforward ‘design for test’ strategy.
- 3.3V supply voltage.

Introduction

AMULET3H is an integrated asynchronous microprocessor subsystem based around AMULET3, a 32-bit asynchronous microprocessor compatible with the ARM instruction set. AMULET3H is designed to be suitable for embedded applications.

In addition to the AMULET3 processor, AMULET3H incorporates 8 Kbytes of static RAM, 16 Kbytes of ROM, a DMA controller and an interface to on-chip synchronous peripherals. It also includes memory interface logic which allows it to interface directly to a wide variety of memory and peripheral devices. The memory interface is directly configurable by the processor. Static memory devices, such as SRAM, EPROM and peripheral chips, can be connected directly to the processor with no extra logic. In addition, DRAM is supported, again with no external support logic.

Background

AMULET3H was designed at the University of Manchester within the EC-funded OMI-DE2 and OMI-ATOM projects. It was developed as a telecommunications controller subsystem with input from our industrial partners in these projects. The work would not have been possible without European Union funding, and the support of the EC and the industrial partners is gratefully acknowledged.

Intellectual Property rights

Disclaimer

Change history

version	date	changes
1.0	22/11/99	first full release

Contents

1. Block Diagram 6

2. Interface signals 7
- 2.1 External memory interface 7
- 2.2 Synchronous On-Chip Bus (SOCB) 7
- 2.3 DMA controller interface to synchronous subsystem 7
- 2.4 ADC/AEDL interface 8
- 2.5 Debug interface 8
- 2.6 Interrupt requests 8
- 2.7 Notes 8

3. AMULET3 core 9
- 3.1 Features 9
- 3.2 AMULET3 core organisation 9
- 3.3 Overview 10
- 3.4 Instruction set architecture 10
- 3.5 Reorder buffer 10
- 3.6 Delayed abort response 10
- 3.7 Branch prediction 11
- 3.8 Fast interrupt response 11
- 3.9 Halt 11
- 3.10 Thumb 11
- 3.11 Debug support 11
- 3.12 Implementation dependent features 11
- 3.13 Programming hints 12
- 3.14 Notes 13

4. CPU and BTB control registers 14
- 4.1 CPU control register - CCR - at ffe00008 14
- 4.2 BTB control register - BCR - at ffe0000c 14

5. Debug architecture 15
- 5.1 Features 15
- 5.2 Debug control registers 15
- 5.3 Debug scan registers 16
- 5.4 Memory map 19
- 5.5 Debug interrupt requests 19
- 5.6 Debug operation 19
- 5.7 Power-efficiency 20
- 5.8 Notes 20
- 5.9 Code examples 20

6. 8Kbyte SRAM module 22
- 6.1 RAM block diagram 22
- 6.2 SRAM processor and MARBLE connections 22
- 6.3 Notes 23

7. DMA controller 24
- 7.1 Features 24
- 7.2 DMA controller block diagram 24
- 7.3 Programming the DMA controller 25
- 7.4 Chaining 28
- 7.5 Global registers 29

7.6 Sequence of events during a single transfer 31
7.7 Prioritisation 31
7.8 Request timing 31
7.9 Power On Initialisation 33
7.10 Reset 33
7.11 Clocking 33

8. 16 Kbyte ROM module 34
8.1 Features 34
8.2 MARBLE interface 34
8.3 Notes 34

9. The ADC/AEDL interface 35
9.1 Introduction 35
9.2 Interface signals 35
9.3 Interface timing 35
9.4 Circuit 36
9.5 Memory map and registers 38
9.6 Action of the AEDL 38
9.7 Programming sequence 38

10. The External Memory Interface (EMI) 39
10.1 Features 39
10.2 Introduction 39
10.3 Interface signals 39
10.4 Supported memory types 40
10.5 Memory decoding 40
10.6 Bootstrap 40
10.7 Action of reset 41
10.8 Timing reference 41
10.9 Memory cycle types 42
10.10 Memory width, byte selection and funnelling 43
10.11 DRAM interface 43
10.12 Static interface 44
10.13 Control registers 47
10.14 Test mode 47
10.15 Notes 47

11. Memory Map 49
11.1 Overall memory map 49
11.2 On-chip RAM 49
11.3 On-chip ROM 49
11.4 Action at reset 49
11.5 AEDL/ADC registers (base at ff800000) 50
11.6 SOCB registers (base at ffa00000) 50
11.7 DMA controller registers (base at ffc00000) 50
11.8 CPU debug and control registers (base at ffe00000) 51
11.9 EMI control registers (base at fff00000) 51

12. The MARBLE bus 52
12.1 Features 52
12.2 Architecture 52
12.3 Signal List 53
12.4 Implementation Specifics 53
12.5 Initiator Bridge 54
12.6 Target Bridge 56

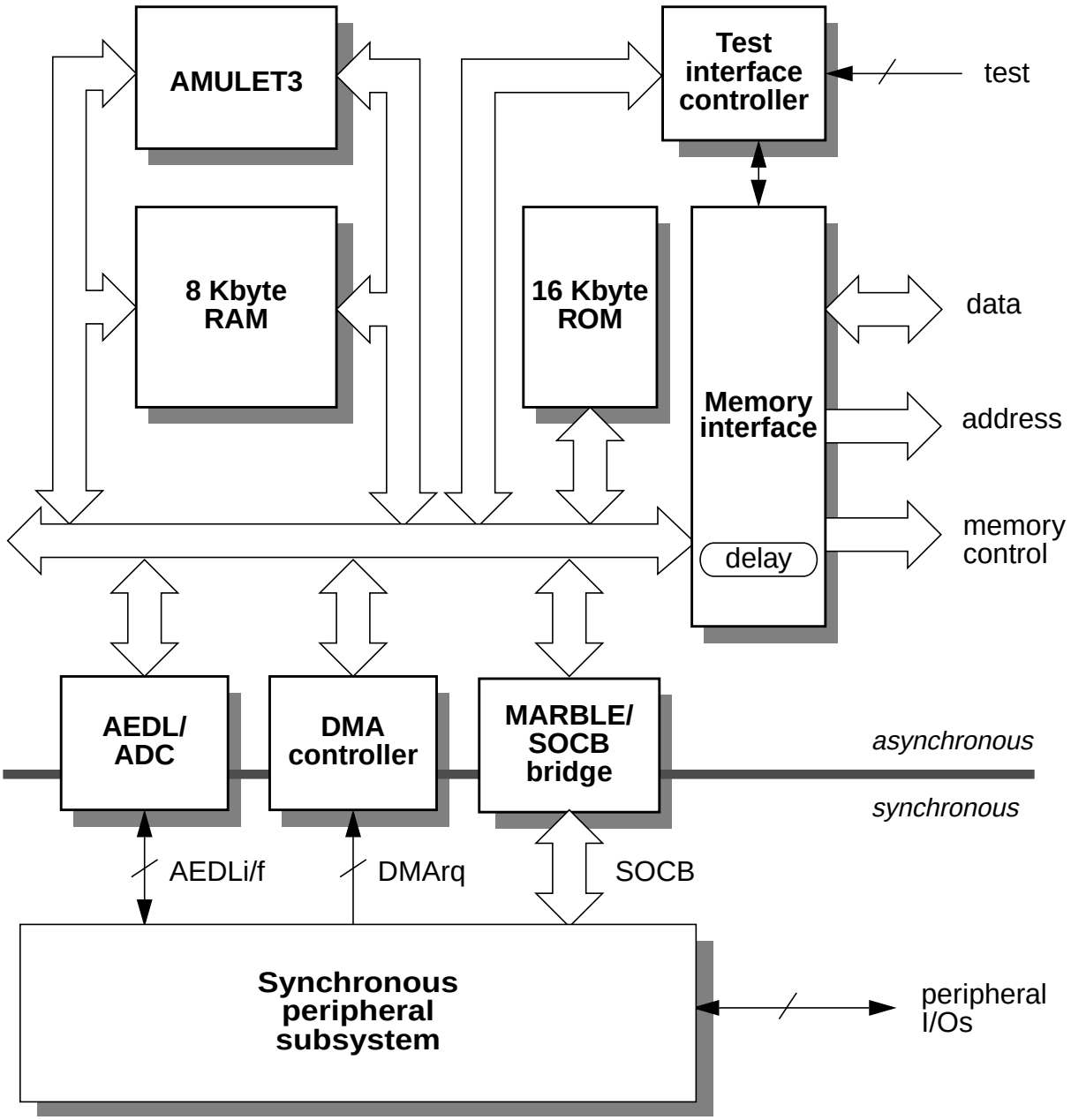
13. The MARBLE/SOCB Bridge (MSB). 58
13.1 Features 58
13.2 Interface signals 58
13.3 The SOCB 58
13.4 Operation of Nwait 58
13.5 Synchronisation and bus timing 59

14. The AMULET3H Test Interface Controller (TIC). 60
14.1 Features 60
14.2 Test interface signals 60
14.3 External pin mapping 60
14.4 Test interface operation 61
14.5 TIC registers 61
14.6 TIC operations 61
14.7 TIC timing diagram 62
14.8 Notes 62

15. Test facilities and procedures. 63
15.1 Testing the AMULET3 core 63
15.2 Testing the BTB 64
15.3 Testing the on-chip debug logic 67
15.4 Testing the 8 Kbyte dual-port RAM 68
15.5 Testing the 16 Kbyte ROM 68
15.6 Testing the DMA controller 68
15.7 Testing the external memory interface (EMI) 68
15.8 Testing the MARBLE bus 69
15.9 Testing the MARBLE/SOCB bridge 69
15.10 Testing the AEDL 69
15.11 Testing the synchronous peripheral subsystem 70

1. Block Diagram

The major components of AMULET3H and their relationship to the synchronous peripheral subsystem are illustrated in the figure below.



2. Interface signals

2.1 External memory interface

Signal	Type	Function
d[15:0]	IOZ	16-bit off-chip data bus
a[28:0]	O	29-bit off-chip address bus[a]
rNw	O	read/not write
Noe	O	output enable (active low)
Nms[7:0]	O	memory select/row address strobes ($\overline{RAS}$)[b]
Nbs[1:0]	O	byte select/byte write strobes
Ncas[1:0]	O	column address strobes ($\overline{CAS}$)
bt[1:0]	I	boot mode select
Ntest	I	test mode enable (high for normal operation)

a. a[28:20] are multiplexed with peripheral I/O signals on DRACO.
b. Nms[7:4] are multiplexed with peripheral I/O signals on DRACO.

2.2 Synchronous On-Chip Bus (SOCB)

Signal	Type	Function
Nwait	I	informs bridge that the shared RAM is busy
NIOCS	O	SOCB peripheral access signal
NSBrd	O	read from SOCB
NSBwr	O	write to SOCB
SA[19:0]	O	SOCB address lines
SD[31:0]	IOZ	SOCB data lines
SIZE[1:0]	O	SOCB size of data transfer
CLK	I	SOCB clock

2.3 DMA controller interface to synchronous subsystem

Signal	Type	Function
DMArq[15:0]	I	DMA requests from synchronous peripherals
DMAirq	O	IRQ request from DMA controller
DMAfiq	O	FIQ request from DMA controller

2.4 ADC/AEDL interface

Signal	Type	Function
AEDL_en	I	Active high enable for AEDL
AEDL_sync	I	Selects level (0) or edge (1) AEDL synchronization
ADC_rd	O	ADC read, active high
AEDL_ch[1:0]	O	AEDL channel number (reflect processor A[7:6])
AEDL_rdy[2:0]	I	AEDL event signals
AEDL_ack[2:0]	O	AEDL acknowledges
ADC_din[9:0]	I	ADC data (9 bits, [8:0]) & time-out (bit [9])

2.5 Debug interface

Signal	Type	Function
INTA	O	breakpoint (instruction) interrupt request
INTD	O	watchpoint (data) interrupt request

2.6 Interrupt requests

Signal	Type	Function
Nreset	I	global system reset
Nirq	I	IRQ request to AMULET3H core
Nfiq	I	FIQ request to AMULET3H core

2.7 Notes

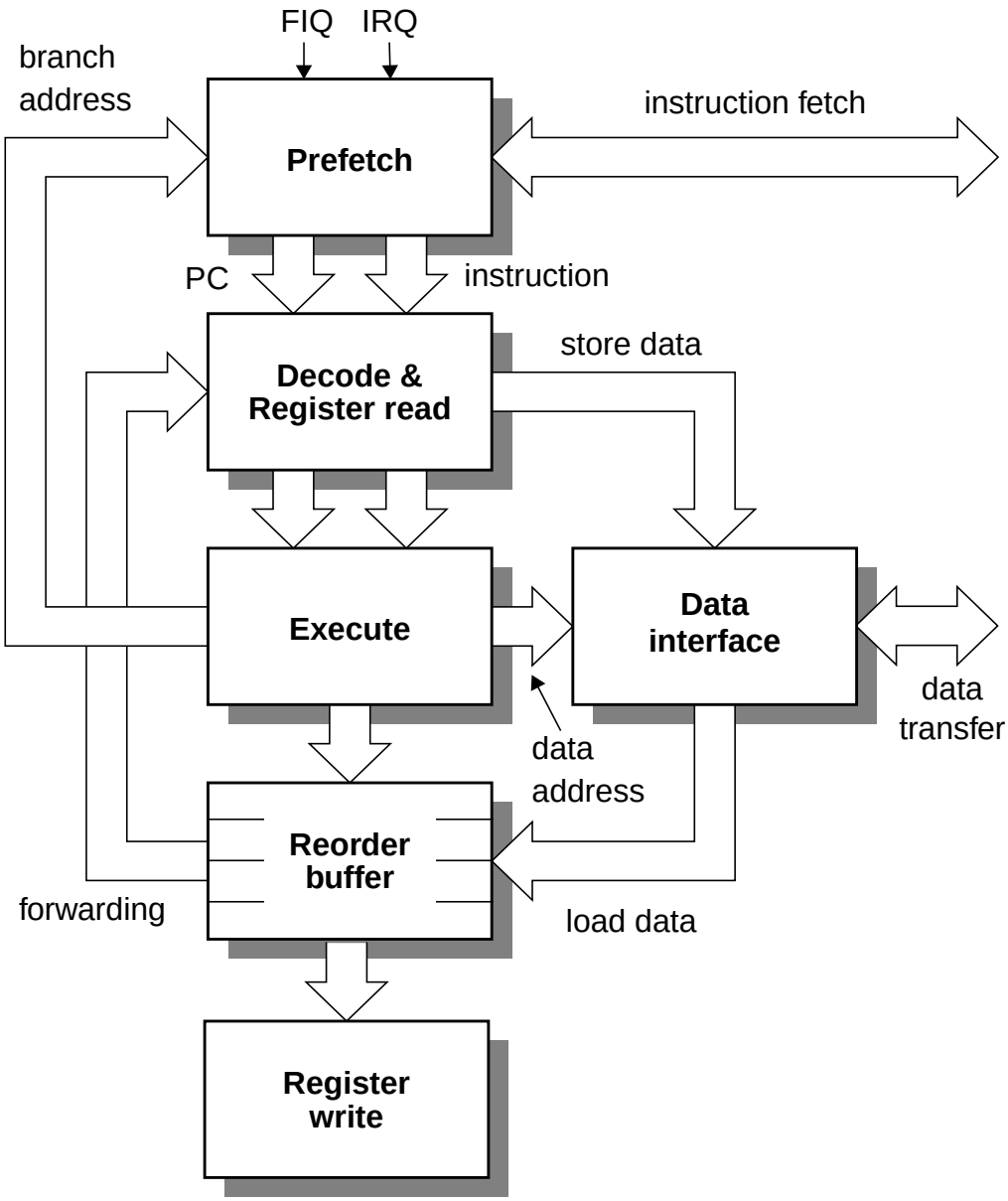
- in test mode a[19:0] (in the external memory interface) are used as inputs (see ‘The AMULET3H Test Interface Controller (TIC)’ on page 60).
- the main interrupt controller is in the synchronous peripheral subsystem; this is why the DMA and Debug interrupt requests are outputs from, and Nirq, Nfiq inputs to, AMULET3H.

3. AMULET3 core

3.1 Features

- Compatible with ARM architecture v4T instruction set
- 'Thumb' 16-bit compressed instruction set
- Branch target prediction and branch fetch suppression
- Low latency interrupts
- Dual ('Harvard') bus interface
- Zero power sleep mode.

3.2 AMULET3 core organisation



3.3 Overview

AMULET3 is the third generation asynchronous ARM microprocessor core. It implements ARM architecture version 4T and supports the 16-bit Thumb compressed instruction set.

The processor core may be described by six major functional blocks, as shown above. The normal flow of operation is for the prefetch unit to generate instructions which flow down the pipeline and eventually cause changes to the registers. In AMULET3 the data interface is 'sidelined' from the main instruction flow which allows the decoupling of data transfer operations (especially multiple register moves) from purely internal operations.

The processor core communicates with memory using separate instruction and data buses, but expects a unified memory model, for example through the use of a dual-ported RAM or cache.

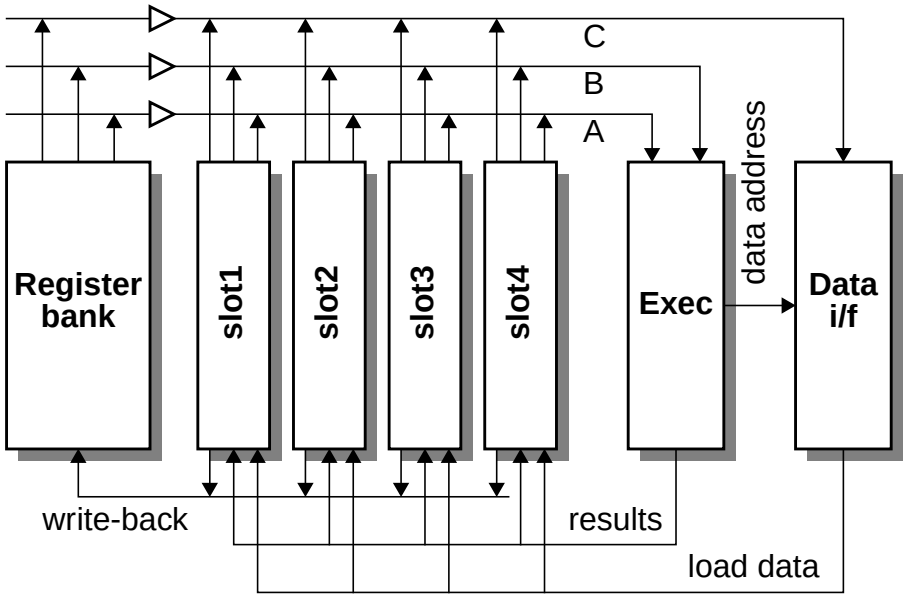
3.4 Instruction set architecture

The AMULET3 instruction set architecture is not described here as it is fully compatible with the 32-bit ARM architecture v4T which is documented elsewhere. Older, 26-bit address space ARM modes are NOT supported by AMULET3.

3.5 Reorder buffer

The processor supports out-of-order completion and employs a reorder buffer to allow register forwarding while preserving the exact exception model assumed by the ARM architecture.

The following diagram shows how results can return to one of the four reorder buffer 'slots', with internal results and loaded data values being interleaved arbitrarily. Once in a slot, the value can be forwarded for use as an operand by any following instructions. The write-back process returns results cyclically to the register file, in order, some time later; memory faults are detected during the write-back process and an exception raised with the register file in a consistent state.



The processor core contains a number of other novel features to enhance performance:

3.6 Delayed abort response

The data interface is loosely coupled to the rest of the processor. Data operations are executed largely on a 'fire and forget' basis, loaded values being returned and reordered into the register bank asynchronously and at a later time. The reordering process allows data aborts to occur after

subsequent instructions have been resolved by delaying the final register writeback until the data transfer has completed correctly. This enables significant speculation following load and store instructions without paying penalties for slow memory.

3.7 Branch prediction

The branch prediction unit is able to predict correctly a proportion of previously encountered branches. This increases performance and lowers power consumption by reducing the number of erroneous prefetch cycles. Additional gains are made by resynthesizing the predicted branch instructions internally, further reducing the number of memory cycles. Unconditional branches may be predicted and executed by the prefetch unit without any other processor intervention.

3.8 Fast interrupt response

Unusually the interrupt signals are injected not into the instruction decoder but into the prefetch unit. This feature allows an interrupt to be treated as a predicted branch, and the interrupt service code may begin to be fetched as soon as the interrupt occurs, rather than waiting for the execution unit to ‘branch’ in the normal way. This feature also simplifies the halt mechanism.

3.9 Halt

The asynchronous nature of AMULET3 enables the processor to halt – with a consequent reduction in power consumption to near-zero – immediately on receiving the relevant ‘B .’ instruction. This is done by suspending the prefetch until an interrupt occurs, subsequent instructions rapidly draining from the pipeline.

3.10 Thumb

Unlike some earlier, synchronous, implementations of Thumb AMULET3 fetches Thumb instruction data as 32-bit words, so two instructions are fetched per bus cycle. This reduces the number of bus cycles performed by almost a factor of two (when using 32-bit memory) and thus should reduce system power significantly. The Thumb decoder therefore normally receives two 16-bit instructions as a single packet.

The ARM instructions output by the Thumb decoder are determined by combinatorial look-up. The output instructions are actually slightly extended from conventional ARM operations because not all Thumb instructions have direct ARM counterparts. The appropriate extensions are also added in this stage if the 32-bit instruction set is in use.

3.11 Debug support

The AMULET3 core is equipped with value and mask registers which are used to detect particular address, data and control values on the instruction memory port (to set breakpoints) and on the data memory port (to set watchpoints). When a match (under mask) is detected an interrupt request is generated or an abort signalled, giving a basic debug capability.

The AMULET3H debug architecture is described further in ‘Debug architecture’ on page 15.

3.12 Implementation dependent features

Although AMULET3 implements the full ARM v4T instruction set some parts of this ISA are ‘implementation defined’. AMULET3 obeys the following conventions:

- When an instruction uses the PC as a source operand the value is always the address of the next instruction but one. This means that the value is PC+8 in ARM mode and PC+4 in Thumb mode; there are no cases where PC+12 is obtained.
- If a data abort occurs during a load or store instruction with base writeback the base register remains unchanged (i.e. the correct value to restart the instruction). This is true of all memory

operations.

- The mode of ‘early termination’ on multiplications is based upon the value in Rs (bits[11:8] of the ARM instruction, which corresponds to Rd, bits [3:0] of the Thumb MUL instruction). The multiplier uses up 8 bits of the Rs multiplier per (self-timed) cycle, using 4 cycles to compute the full product. If the top 24, 16 or 8 bits of Rs are all zeros or all ones the multiplier will cycle faster for the last 3, 2 or 1 cycles respectively.
  (Multiplier cycles are significantly faster than full datapath cycles and the ‘early termination’ speed-up is less than a factor of 2, so the benefits of early termination will not be spectacular!)

3.13 Programming hints

This section contains a few random hints for the programmer to get the most out of AMULET3.

1. The execution path is pipelined and therefore register dependencies between instructions affect performance (hardware interlocks ensure correct results). An instruction which requires an operand from the immediately preceding instruction may be delayed until the value is available. This delay is small or zero for internally derived values, considerably more (memory speed dependent) for externally loaded values. Instruction issue is in the written order, so for best performance instructions which depend on the result of a ‘load’ should not immediately follow the load. There is no such penalty imposed by the flag register, so - for example - a conditional branch can immediately follow a comparison without penalty.
2. After the address calculation LDM/STM instructions run independently of the execution unit. It is therefore possible to execute the succeeding instruction largely in parallel, providing dependencies do not occur. Dependencies are resolved on a register by register basis and so will be more severe for the later registers in the load sequence. (As in all ARMs, registers are loaded with the lowest register number first.)
3. Loading R15 from memory is a special case and it is performed using the instruction fetch port. This allows it to be done in parallel with other register loads, reducing the latency of the ‘LDM style’ procedure returns.
4. Many operation timings are data dependent. For example an ADD operation occupies the execution stage for perhaps 15%-20% longer than a MOV. Using logical operations (e.g. TEQ rather than CMP) when possible may save a small amount of time.
5. Data accesses with post-increment addressing modes are despatched earlier than those with pre-increment addressing modes, thus reducing overall latency. However, specifying base writeback on any addressing mode imposes a small timing penalty in the decode stage.
6. The branch predictor is simplistic and assumes that (known) branches are always taken. Performance will be improved if code is arranged so branches in frequently executed code (such as loops) are normally taken.
7. MSR - for example changing mode or interrupt enable status - incurs a penalty similar to an unpredicted branch.
8. When running Thumb code the instructions are fetched as 32-bit words. This reduces any penalties imposed by the speed of the memory system. For best performance branch targets in Thumb code should be whole word addresses when running from internal 32-bit memory.
9. When a multiplicand is known to have fewer than 24 bits it should be put in the Rs field in the ARM multiply instruction (Rd in Thumb) to maximize the benefits of early termination. (See ‘Implementation dependent features’ on page 11)
10. When using the PC (R15) as a source operand (including branch operations) it must be incremented to PC+8 (PC+4 in Thumb mode); this is done by a serial incrementer to reduce the hardware and power requirements. This will have no noticable effect in most cases, but instructions which read R15 will be slower if they are located at certain addresses. The incrementer time is proportional to the number of contiguous "1" bits at the least significant end of the instruction address, beginning at bit 3 (bit 2 in Thumb mode). This implies the following code properties:

```
7FFFF7F8     MOV    R9, R15    ; Possible small penalty  ...
```

```
7FFFFFF4      MOV    R0, R15     ; Negligible penalty
7FFFFFF8      MOV    R1, R15     ; Severe penalty
7FFFFFFC      MOV    R2, R15     ; Negligible penalty
```

11. Note that a few other instructions which have encodings that imply the PC is read (notably BX) also suffer from this.
12. There is a penalty associated with initiating an LDM/STM instruction, thus for single register loads/stores LDR/STR should be used in preference.
13. An LDM/STM instruction which fails its condition code check is aborted after the first cycle.
14. Memory operations that fail their condition code test bypass the memory interface and go straight to the reorder buffer.

3.14 Notes

1. SWAP instructions will only work correctly to addresses in the on-chip RAM. If a SWAP instruction targets an address across MARBLE the write phase of the instruction will be ineffective (the addressed memory location will be unchanged by the instruction) but the correct read data will be returned to the target register.
2. Most undefined instruction codes are fully trapped. An exception is those opcodes which correspond to an LDRH with register offset where bits[11:8] are not zero; these instructions are not trapped but behave as if bits[11:8] were zero.

4. CPU and BTB control registers

These registers control assorted detail functions of the AMULET3 processor core. Normally they will be configured at start-up and then left alone. All of the writable control bits are reset to zero by the system reset signal.

4.1 CPU control register - CCR - at ffe00008

	7	6	5	4	3	2	1	0
CCR	TstInt	TstClk	TstNrst	Turbo	CCen	MulET	BigEnd	HaltEn
reset to:	0	0	0	0	0	0	0	0

- Turbo - controls the processor's pipeline speed/economy setting. This should normally be set to 0 for optimum power efficiency.
- CCen - Counterflow colour enable - when 1 enables an optimisation in the core which allows the flags to percolate back up the pipeline to enable instructions to be discarded more efficiently.
- MulET - Multiplier early termination. In AMULET3H this control bit is ineffective and the multiplier always uses early termination.
- BigEnd - Big endianism - when 1 causes the processor to take a big endian view of memory. As the AMULET3H subsystem is little endian, this bit should be left set to 0.
- HaltEn - Halt enable - when 1 enables the processor's halt function.
- TstInt, TstClkn, TstNrst- test signals (See 'Testing the AMULET3 core' on page 63 and 'Testing the synchronous peripheral subsystem' on page 70). Must be left at zero during normal operation.

4.2 BTB control register - BCR - at ffe0000c

	7	6	5	4	3	2	1	0
BCR	bt[1:0]		-	BTBcnt	BTBlkp	BTBpsv	BTBflsh	BTBen
reset to:			0	0	0	0	0	0

- BTBcnt - BTB count control - when 0 a pseudo-random counter chooses the next slot which will be used in the BTB; when 1 a sequential counter is used. The former has better statistical properties for mean program performance, but the latter is more deterministic and may give better control for critical real-time code. (In practice the difference may be very small!)
- BTBlkp - BTB look up - when 1 enables BTB lookups on subroutine returns and other uncacheable PC changes, when zero disables such lookups. It can only be enabled when the BTB power save logic is disabled (BTBpsv = 0). If BTBpsv = 1 BTBlkp is ignored and treated as 0.
- BTBpsv - BTB power save - when 1 enables the BTB power save logic. This should normally be enabled. Enabling the BTB power save logic disables BTBlkp automatically.
- BTBflsh - BTB flush - when 1 causes all the BTB locations to be cleared whenever a SWI is executed. This is an automatic mechanism to prevent errors when an old program is overwritten with a new one (assuming that program load involves a SWI operation).
- BTBen - BTB enable - when 1 enables the BTB in normal operating mode.
- bt[1:0] - the external boot control pins - are readable through BCR[7:6]. This has nothing to do with controlling the BTB, but may be useful for test or other general programming purposes.

5. Debug architecture

The AMULET3H subsystem includes breakpoint and watchpoint hardware which can cause an interrupt or abort to be generated whenever a particular instruction or data access occurs. The debug registers specify values for the address, data and control signals on each of the AMULET3 instruction memory and data memory interfaces, and they also specify mask values which cause any subset of these signals to be ignored in the matching logic.

5.1 Features

- set a breakpoint on a particular instruction fetch address, address range, data value, control value, or combination of address, data and control;
- set a watchpoint on a particular data transfer address, address range, data value, control value, or combination of address, data and control;
- breakpoints and watchpoints generate interrupts or aborts, allowing the processor state to be examined.

5.2 Debug control registers

The debug registers are implemented as serial scan chains, one for the instruction memory interface and one for the data memory interface. They are loaded by software via 8-bit control registers, the BreakPoint Control register (BPC) and the WatchPoint Control register (WPC).

	7	6	5	4	3	2	1	0
BPC	BPmout	BPdout	BPmin	BPdin	BPabt	BPden	BPen	BPck
reset to:	-	-	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
WPC	WPmout	WPdout	WPmin	WPdin	WPabt	WPden	WPen	WPck
reset to:	-	-	0	0	0	0	0	0

The functions of the individual breakpoint control register bits are as follows:

Signal	R/W	Function
BPck	R/W	Breakpoint register clock
BPen	R/W	Breakpoint address and control enable
BPden	R/W	Breakpoint read data comparator enable
BPabt	R/W	Breakpoint abort enable
BPdin	R/W	Breakpoint register serial data in
BPmin	R/W	Breakpoint register serial mask in
BPdout	R	Breakpoint register serial data out
BPmout	R	Breakpoint register serial mask out

- The watchpoint control register bits are defined similarly. (WPen also enables the write data comparators.)

- The scan registers are loaded by repeatedly clocking (using BPck, WPck) data bits in from BPdin and WPdin and mask bits in from BPmin and WPmin. The rising edge of the clock is active, so the ‘data in’ and ‘mask in’ values should be valid as the clock rises (see code samples later).
- The ‘data out’ bits (BPdout, WPdout) and ‘mask out’ bits (BPmout, WPmout) are for test use only - they enable the debug scan registers to be observed directly.
- The enable bits (BPen, BPden, WPen, WPden) should be zero during scan-in of the corresponding register to prevent spurious interrupts, and BPen/WPen should be pulsed to zero after a debug interrupt to clear the appropriate interrupt request.

5.3 Debug scan registers

The signals matched by the debug registers are listed below. The scan chain signal and corresponding mask orders will match on both chains.

- the breakpoint signals are:

Signal	Function
IA[31:0]	Instruction memory address
Iread	Instruction read (must be 1 or ‘don’t care’)
Iwrite	Instruction write (must be 0 or ‘don’t care’)
Isize[1:0]	Instruction size - see encoding below
Imode[2:0]	Instruction mode
ILdPC	Instruction load indirect PC
ID[0:31]	Instruction data

- the watchpoint signals are:

Signal	Function
DA[31:0]	Data memory address
Dread	Data read
Dwrite	Data write
Dsize[1:0]	Data size - see encoding below
Dmode[2:0]	Data mode
Dcopro	Data coprocessor transfer
DD[0:31]	Data data

Note that both chains have the same number and order of signals. Where signals are irrelevant (such as Iread and Iwrite) they should be set to ‘don’t care’, i.e. the mask bit should be 1.

Note also that the data bus is in reverse bit order (most significant bit entered first). This reduces the cost (in silicon area) of the debug hardware.

Size encoding

The encoding of the Isize[1:0] and Dsize[1:0] fields is shown below.

Note that, unlike the clocked ARM cores, AMULET3 performs most Thumb instruction fetches as

word accesses, fetching two Thumb instructions at a time. Half-word Thumb accesses are only used when a branch target is at an odd half-word address.

- The Dsize[1:0] encoding corresponds to the MARBLE size encoding and is the same as the MAS[1:0] (memory access size) encoding used on clocked ARMs.

Dsize		Transfer Size
[1]	[0]	
0	0	BYTE: 8-bit transfer
0	1	HALFWORD: 16-bit transfer
1	0	WORD: 32-bit transfer
1	1	Reserved

- The Isize[1:0] encoding is similar, but the byte case does not apply:

Isize		Transfer Size
[1]	[0]	
0	0	Reserved
0	1	HALFWORD: 16-bit Thumb instruction fetch
1	0	WORD: 32-bit ARM or 2 x 16-bit Thumb instruction fetch
1	1	Reserved

Mode encoding

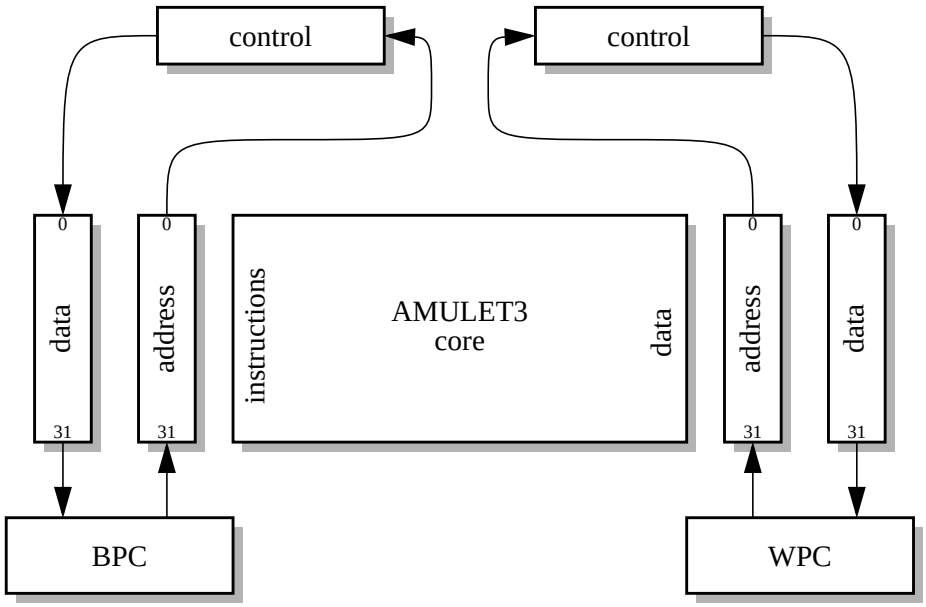
The ARM mode information is encoded as follows:

I/Dmode[2:0]			ARM Mode
0	0	0	User_32
0	0	1	FIQ_32
0	1	0	IRQ_32
0	1	1	SVC_32
1	0	0	System_32
1	0	1	Undef_32
1	1	0	reserved
1	1	1	Abort_32

(AMULET3 only supports the 32-bit ARM modes; it does not support the older 26-bit modes.)

Debug chain organization

Each debug chain is organized in three sections, as illustrated below:

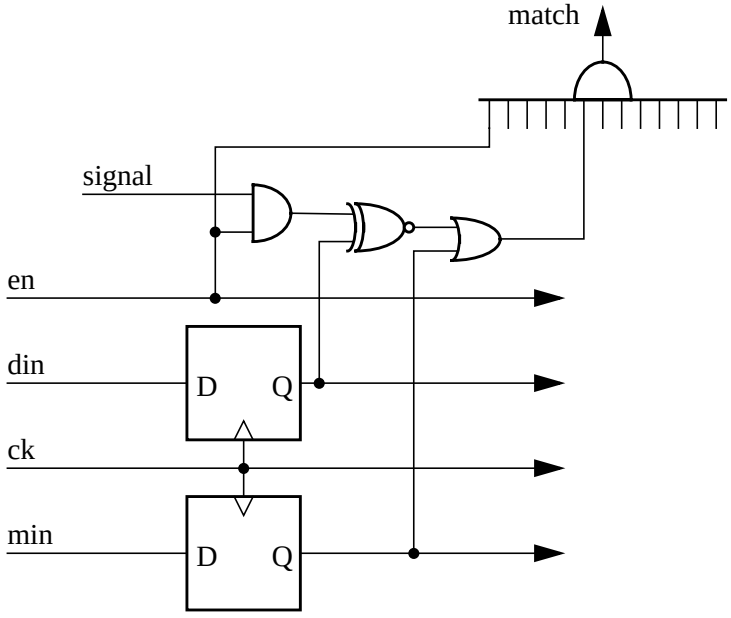


The signals which pass around the breakpoint chain are BPck (to each register/comparator unit in parallel), BPen (to the address and control comparators; WPen also goes to the write data comparator, which is automatically disabled on a load cycle), BPden (to the read data comparators) and BPdin and BPmin (from each register/comparator unit to the next in series, finally to BPdout and BPmout).

- Note that for the data section the enable signal is separate (BPden) - this is to optimise peformance under debug as explained below.

Register/comparator units

For each signal that is tested there is a register/comparator unit as illustrated below:



Debug timing

The address, control and write-data signals are valid at the start of the memory access. Therefore the results of these matches are combined and passed through the appropriate memory control pipeline to be merged with the read-data match signal where appropriate.

Because read data is time critical, it is desirable not to delay sending it to the processor while the debug comparison takes place. However, when abort is used to signal a breakpoint event, some delay is unavoidable. Therefore read data will be delayed when the following events all occur on the same access:

- the read data debug chain is enabled (WP/BPden=1), and
- the address and control debug registers have reported a match or are disabled (WP/BPen=0).

In all other cases the processor's timing is unaffected by the debug hardware.

Load and store data

There is only one data register in the watchpoint unit and this may be used to check loaded or stored values, but not both at the same time (in other words, it is not possible to detect the load or store of a particular value with the same debug register setting). The write data value is automatically ignored on a load instruction, but the read data value comparison must be explicitly disabled by setting WPden=0 if a store data value is to be detected, otherwise it will behave unpredictably.

Enabling the read data comparators on a write cycle will give unpredictable results if the address, control and write data comparators report a match. This can be avoided by using the control comparators to check that the cycle is a read by setting Dread to 1 and unmasking it whenever the read data comparators are enabled (WPden=1).

5.4 Memory map

Address	R/W	Function	Location
ffe00000	R/W	Breakpoint control register (BPC)	MARBLE
ffe00004	R/W	Watchpoint control register (WPC)	MARBLE

5.5 Debug interrupt requests

These interrupt requests are passed to the interrupt controller in the synchronous subsystem where they may individually be enabled/disabled and returned as IRQ or FIQ requests.

Signal	Type	Function
INTA	O	breakpoint (instruction) interrupt request
INTD	O	watchpoint (data) interrupt request

The debug interrupt requests are generated within the debug subsystem by enabling the address/control comparators (WP/BPen=1) and, if required, the read data comparators (WP/BPden=1) and then, after a short delay to allow the comparators to enter stable operation, setting the scan register data input (WP/BPdin) to 1. Example code is given at the end of this section.

5.6 Debug operation

Following system reset both debug registers are disabled (BPen, WPen are zero). To activate one of the registers (for example, the breakpoint register), the following sequence of steps should be followed:

1. Set BPck to zero, BPdin, BPmin to the values required for the furthest signal in the scan register.
2. Set BPck to one leaving BPdin, BPmin unchanged.
3. Set BPck to zero, BPdin, BPmin to the values required by the next signal in the scan path.
4. Repeat steps 2 & 3 until the scan register is fully loaded (72 bits have been clocked in)
5. Set BPen high to activate the breakpoint.
6. Delay until the breakpoint hardware is stabilised.
7. To enable an interrupt on the breakpoint event: set BPdin to one and enable INTA in the interrupt controller.
8. To enable an abort on the breakpoint event: set BPabt to one.

When there is a match on the breakpoint register INTA will be set high. INTA is cleared by disabling the breakpoint (BPen = 0). To re-enable the same breakpoint, set BPen high again. To set a different breakpoint, leave BPen low and repeat the above programming sequence.

5.7 Power-efficiency

The debug comparators will use some power. To minimize power in normal operating mode, the comparators are disabled whenever BPen, BPden, WPen and WPden are zero. As system reset sets these bits to zero, the debug hardware will be in a minimum power configuration unless it has been explicitly enabled.

5.8 Notes

- Indirect PC load (the data transfer phase of LDR pc,<address> and LDM <base>,{...pc}) uses the instruction memory interface to load the PC value and hence should be trapped with a breakpoint, not a watchpoint. (This is different from the ARM model.)
  Note: the same address will also appear on the data port so that a data abort can be generated if the address is bad. The PC value will *not* appear on the data bus here; the data memory is bypassed on this cycle.
- Breakpoints on instructions in a branch shadow will fire even though the breakpointed instruction is not executed.
  If abort signalling is used the breakpoint will be ignored (just as prefetch aborts on instructions in the branch shadow are ignored).
  If interrupt signalling is used an interupt will be generated whether or not the instruction is executed.
- If branch prediction is enabled a branch instruction can be executed without being fetched from memory, and therefore a breakpoint set on its address may not trigger.. The first time the branch is executed it will be fetched, but thereafter it will be held in the BTB for an unpredictable period of time during which it will be accessed from the BTB and not fetched from memory.
- The debug hardware becomes active a little while after the enables (BPen, BPden, WPen, WPden) are set and the behaviour is unpredictable immediately after they have been set. Thereore a little time must be allowed after setting the enables and before enabling the interrupt or abort outputs (see the sample code at the end of this section).

5.9 Code examples

Routines to load the 72-bit BP or WP scan register are given overleaf. The source of mask data uses 6 complete words to hold a 32-bit data mask and a 32-bit data value (both bit reversed), an 8-bit control mask and an 8-bit value (each occupying the bottom byte of a word) and a 32-bit address mask and a 32-bit value.

```
; veneers to make LdScn specific to BPC or WPC

LdBPC        LDR  r8, BPC
             B    LdScn
LdWPC        LDR  r8, WPC

; subroutine to load a complete 72-bit scan chain
; r7 -> source of mask & data; r8 -> BPC or WPC
; r1, r2, r5, r6, r12, r13 (& r14) are used.

LdScn        MOV  r12, r14                  ; save LR
             LDMIAr7!, {r1-r2}              ; load 'data' mask & data
             MOV  r13, #32                  ; scan out 32 bits
             BL   ScanP
             LDMIAr7!, {r1-r2}              ; load 'ctrl' mask & data
             MOV  r13, #8                   ; scan out 8 bits
             BL   ScanP
             LDMIAr7!, {r1-r2}              ; load 'address' mask & data
             MOV  r13, #32                  ; scan out 32 bits
             BL   ScanP
             MOV  pc, r12                   ; return to saved LR

; subroutine to shift data into a scan register
; r1 = mask; r2 = data; r8 -> BPC or WPC; r13 = count
; r5, r6 are used

ScanP        MOV  r5, #0
             STR  r5, [r8]                  ; zero B/WPC
sclp         AND  r5, r1, #1
             AND  r6, r2, #1
             MOV  r5, r5, LSL #5            ; mask in (B/WPmin) bit
             ORR  r5, r5, r6, LSL #4        ; value in (B/WPdin) bit
             STR  r5, [r8]
             ORR  r5, r5, #1                ; clock (B/WPck) bit
             STR  r5, [r8]
             MOV  r1, r1, ROR #1
             MOV  r2, r2, ROR #1
             SUBS r13, r13, #1
             BNE  sclp
             MOV  pc, r14
```

The following routine can be used to enable first the comparators and then, after a delay, the abort generation logic:

```
; subroutine to enable BP or WP for abort generation
; r8 -> BPC or WPC; r5 is used
; turn on B/WPabt and B/WPdin (enables interrupt) some time after B/WPen:

EnAbt        MOV  r5, #&06     ; enable B/WP address, control and
             STR  r5, [r8]     ;    read data comparators (en & den)
             STR  r5, [r8]     ; ensure comparators enabled
             STR  r5, [r8]     ;    before enabling abort & int!!
             ORR  r5, r5, #&18 ; enable B/WPabt (& hold B/WPen, den)
             STR  r5, [r8]     ;    & set din=1 to enable interrupts
             MOV  pc, r14
```

If the data comparator is not to be enabled the immediate value in the first 'MOV' instruction above can be changed from #&06 to #&02. If interrupt generation only is required the 'ORR' immediate should be #&10; if abort generation only then it should be #&08.

The repetition of the STR instruction is merely wasting time to ensure that the comparators are operating correctly before the abort and interrupt outputs are enabled. Any NOOP would probably do, though using a STR ensures that the critical pipeline (the data store pipeline) is flushed through as needed.

6. 8Kbyte SRAM module

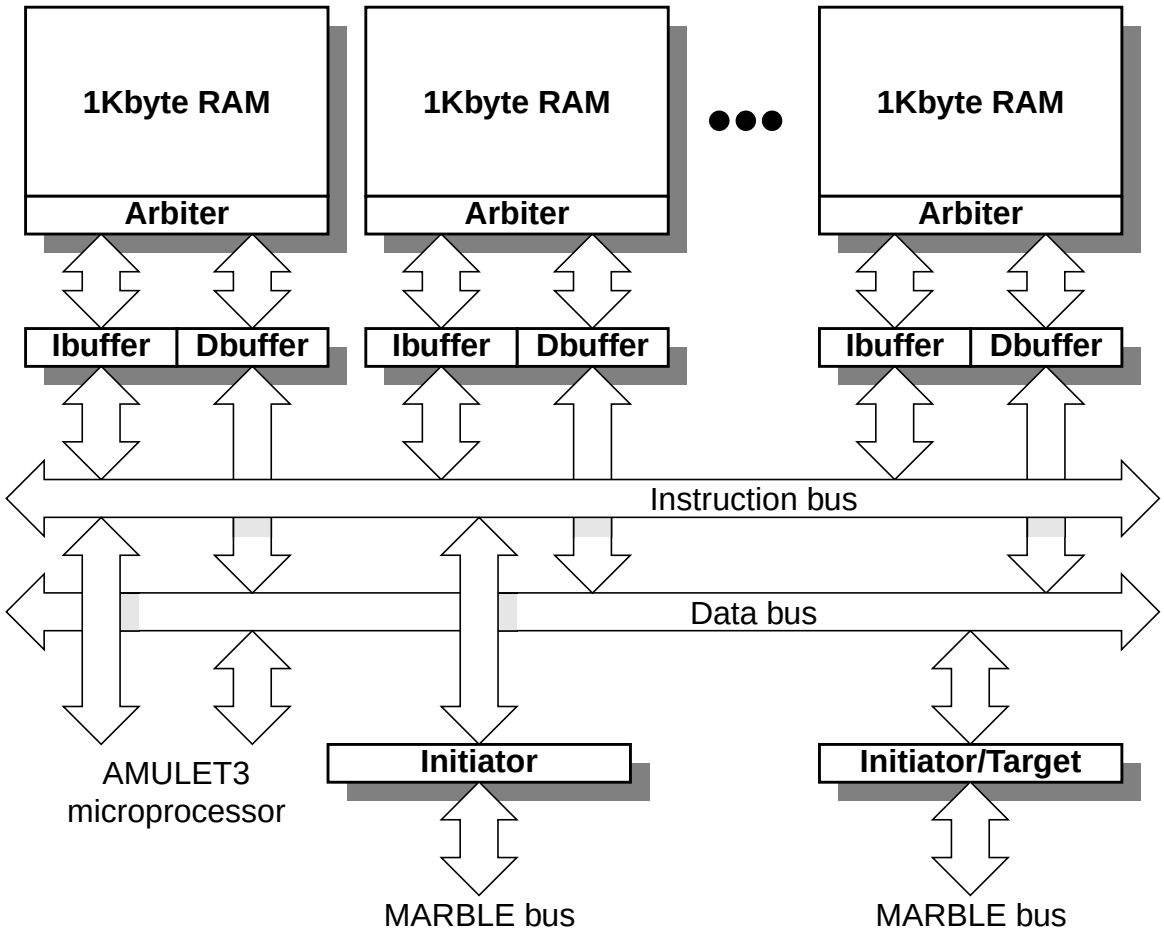
The AMULET3H RAM is an 8 Kbyte static RAM which is divided into eight 1Kbyte blocks. Each block contains 64 lines of 4 words (one word is 4 bytes). Two separate ports are used to handle instructions and data concurrently (see the block diagram). Each block has an internal arbiter to resolve contention between data and instruction references which require access to the same block. Splitting the RAM into eight separate blocks has the following benefits:

- it reduces the power consumption of the AMULET3H RAM, since only one 1K RAM block is activated by an instruction or data reference;
- it reduces the probability of clashes between instruction and data accesses.

Separate instruction and data quadword line buffers are present in each RAM block to:

- improve the overall performance of the RAM, since the majority of the instruction references are sequential which allows the address decode phase to be bypassed in a sequential cycle;
- reduce the total power consumption by minimising the number of accesses to the RAM itself.

6.1 RAM block diagram



6.2 SRAM processor and MARBLE connections

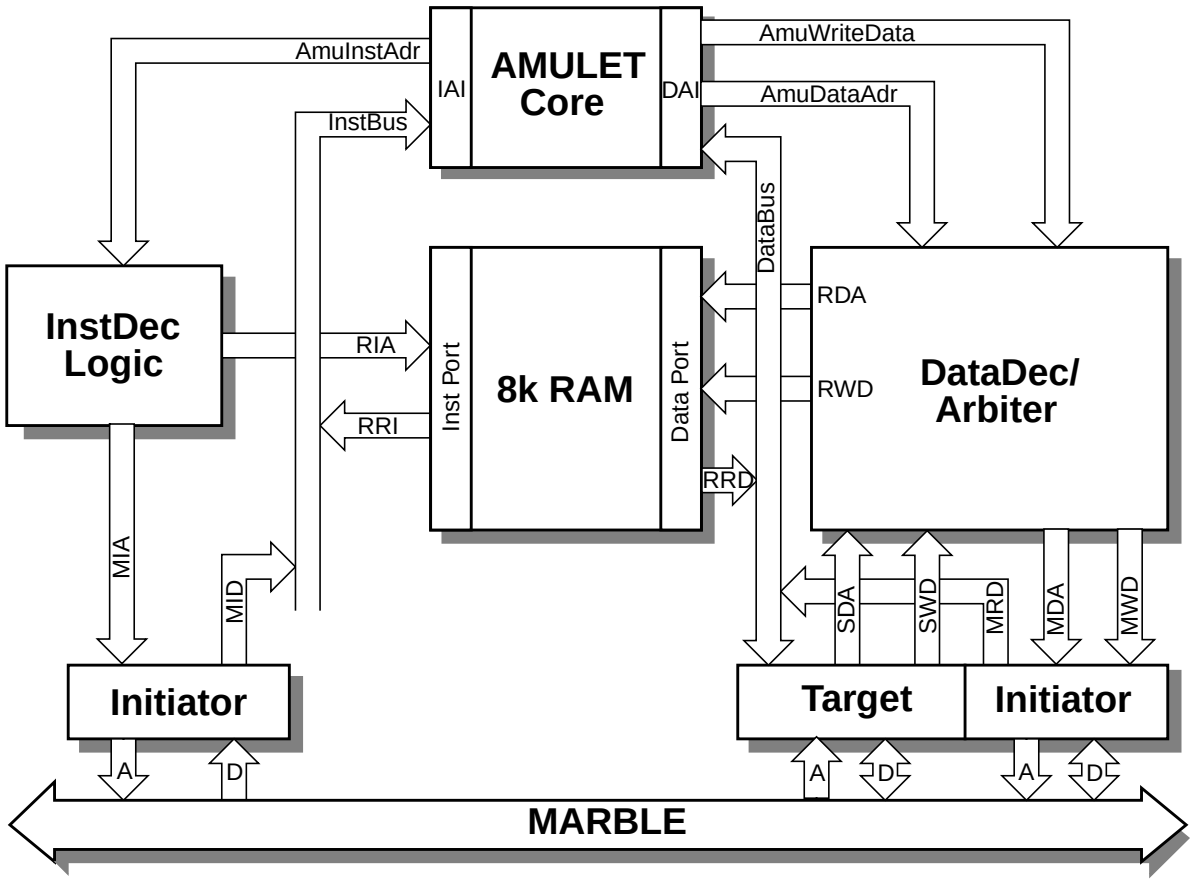
The AMULET3H RAM can be accessed from the AMULET3 processor core, or from the DMA controller which can generate a data read or write request through the target interface of the MARBLE bus. Since the RAM has two separate ports which provide for the supply of instructions and data to the processor independently, two decode logic blocks are required to monitor accesses to these two ports. The main functions of the instruction and data decode logic blocks are:

- to perform the memory management function;
- to direct instruction and data requests to the required target: MARBLE or RAM.

If there is contention between the AMULET processor and the target port of the MARBLE bus for data access to the RAM, arbitration is required. This is incorporated into the data decode logic.

Instructions are supplied to the processor through the instruction bus, which is shared between the RAM block and the initiator port of the MARBLE bus. Data is delivered either to the AMULET3 core or to the MARBLE bus target port using the data bus, which can be occupied by either the RAM or the MARBLE initiator port, but not both.

At the instruction port of the RAM instruction addresses and output data are treated as two separate bundled data interfaces. The same is true for read operations of the RAM through its data port. During the write operation to the RAM the write address and the write data are bundled; i.e. the processor or the DMA requests the write operation only when the write data and the write address are stabilised on their outputs.



6.3 Notes

- Instruction accesses are read only and are handled by the RAM or passed to the MARBLE bus.
- Data accesses are read/write and may also be passed to the MARBLE bus. The RAM may also be accessed by remote MARBLE initiators through the data port, so the data bus must arbitrate between AMULET3 and MARBLE accesses.

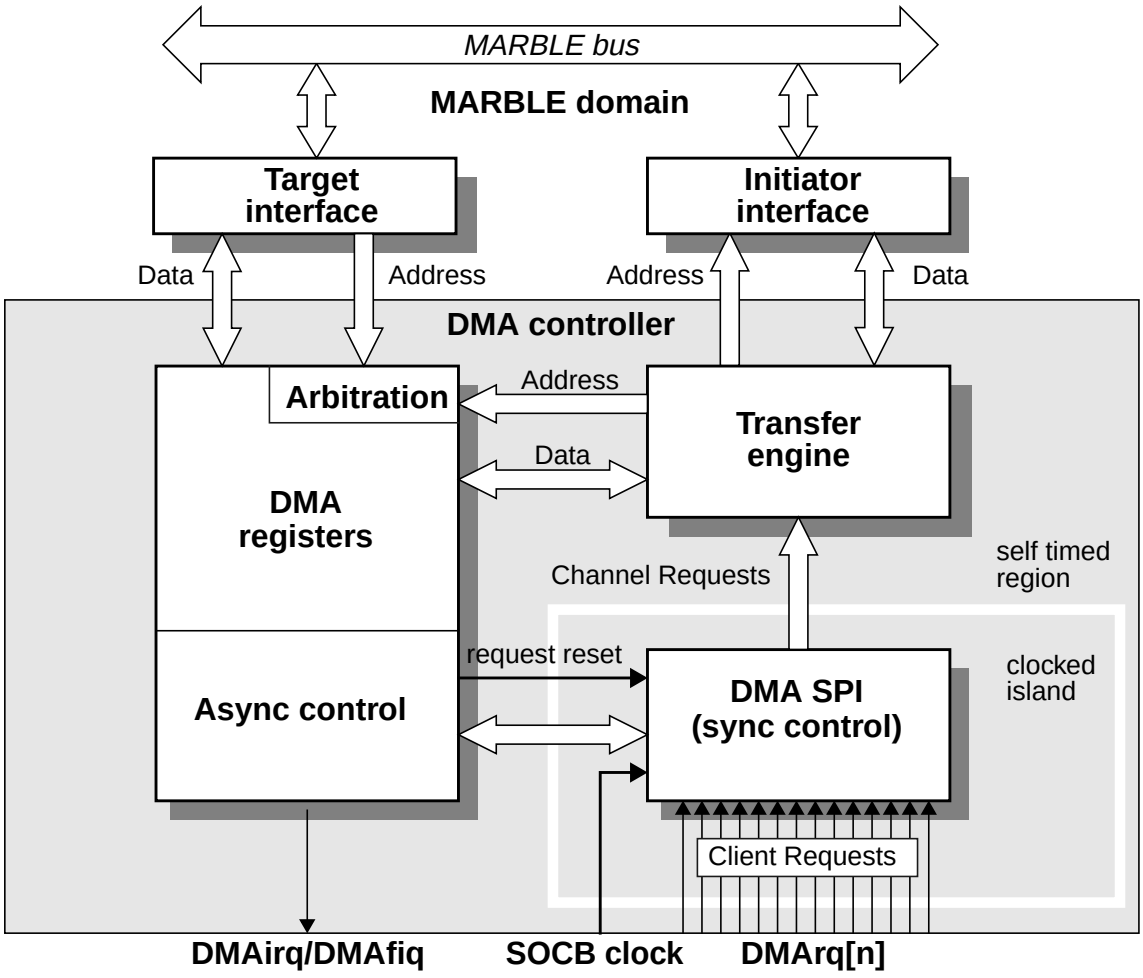
7. DMA controller

The AMULET3H DMA controller is a module which automatically controls data transfers across the MARBLE bus without the intervention of the processor core. The DMA controller has 32 independently programmable channels each of which can perform memory to memory, memory to peripheral, peripheral to memory or peripheral to peripheral transfers. IRQ and FIQ interrupt requests can be signalled by the completion of transfers on each channel.

7.1 Features

- 32 channels: 6 ‘long head’ channels with full memory addressing, 16 ‘short head’ channels with restricted addressing and 10 ‘short chain-only’ channels
- 16 DMA client request sources
- byte, half-word and word wide transfers
- single buffer, store and forward transfers: request-driven or free-run operation
- static, channel-ordered prioritisation
- arbitrary mapping of client requests to 22 ‘long’ and ‘short head’ channels
- request chaining – one request can initiate transfers on multiple channels

7.2 DMA controller block diagram



The DMA controller comprises two main blocks which are connected to the MARBLE bus and the incoming peripheral requests by three interface units. The main blocks are:

- The register bank: contains all of the controller registers, a substantial portion of the overall control circuitry of the controller and a means for arbitrating in an asynchronous fashion between register read/write requests from the MARBLE bus and the transfer engine.
- The transfer engine: performs transfers to/from memory initiated by requests coming from the Synchronous Peripheral Interface (SPI). The transfer engine contains control which sequences data transfer reads and writes and buffers a single word of data moving between source and destination addresses.

The interfaces to MARBLE and the synchronous peripherals are:

- The target interface: connects the register bank to the MARBLE bus allowing the processor core to program the DMA controller.
- The initiator interface: performs the memory reads and writes which actually transport data from one place to another.
- The Synchronous Peripheral Interface (SPI): gathers client requests, maps these client requests onto channels using the channel CTRL register CLIENT fields – see ‘The control (CTRL) registers’ on page 26 – masks off requests made for disabled channels, conditions the request signal to identify request events and presents those *channel* requests to the target interface.

The transfer engine and register bank are connected together to allow the transfer engine to receive channel information prior to initiating a transfer. A connection between the register bank and the SPI allows DMA requests to be set when setting up memory to memory (continuous, non client request driven transfers) transfer runs and for transferring CTRL.ENABLE and CTRL.CLIENT register values between register bank and SPI.

7.3 Programming the DMA controller

The DMA controller is programmed by manipulating the word wide registers between addresses ffc00000 (DMABASE) and ffc00214. The registers are split into two sections: those associated with individual channels, the channel registers, and those which are global to all channels, the global registers.

Channel Types

There are three types of channel:

- Short head channels (0…15) intended for peripheral to peripheral and peripheral to/from memory transfers; these channels have 16-bit long SRC, DST and COUNT registers. The most significant 16 bits of source and destination addresses for transfers performed by these channels are taken from the BASE portion of the GENCTRL global register. A maximum of 2^{16} transfers may be performed in any one run where the CTRL.COUNTINC bit is used.
- Long head channels (16…21) with full 32-bit addressing intended for peripheral to/from memory and memory to memory transfers. Up to 2^{32} transfers can be performed in a transfer run when CTRL.COUNTINC is set.
- Short chain-only channels (22…31) which cannot directly receive client requests, but are controlled by head channels. Short chain-only channels have 16-bit SRC and DST registers like other short channels but have no COUNT register (reading a short chain-only COUNT register returns -1) and fewer control bits in their CTRL registers. Chaining is discussed in ‘Chaining’ on page 28.

Channel registers:

Each channel has a set of four registers associated with it. Each register is a word long and will only (usefully) support word-wide reads and writes. The four registers are:

Address	Register	Function
REGADDR+0	SRC	Source address - peripheral or memory address
REGADDR+4	DST	Destination address - peripheral or memory address
REGADDR+8	COUNT	Remaining transfers counter
REGADDR+12	CTRL	Control register

The value of REGADDR for a channel *n* is DMABASE + 16*n. The SRC, DST and COUNT registers are 16 bits wide for short channels and 32 bits wide for long channels. The COUNT register is sign extended when read from MARBLE and the SRC and DST registers have the contents of GENCTRL.BASE (the BASE portion of the GENCTRL global register) in their most significant half words when read so giving the full 32 bit values of that channel's addresses.

The control (CTRL) registers

Each channel has a 17-bit control register determining the behaviour of that channel. Not all bits are used by chain-only channels. The fields of the control register are:

Bits	Field	Function
CTRL[0][a]	ENABLE	When set: transfers are enabled for this channel.
CTRL[4:1][a]	CLIENT	Client DMArq number which will initiate a transfer on this channel when ENABLE and USEDRQ are both set. A consistent behaviour cannot be guaranteed where two or more channels share the same CLIENT value and use that client with USEDRQ set.
CTRL[6:5]	SIZE	Transfer size 0: byte 1: half-word 2: word 3: reserved
CTRL[7]	SRCINC	When set: the source address (SRC register contents) will be incremented (by 1, 2 or 4 depending on the contents of the SIZE field) after each transfer.
CTRL[8]	DSTINC	Similar to SRCINC but for the destination (DST register) address.
CTRL[9][a]	COUNTINC	When set: the transfer count (COUNT register contents) will be incremented by 1 after each transfer.
CTRL[10][a]	USEDRQ	When set: transfers are initiated by the arrival of a client request from DMArq[CLIENT] ; when reset: a transfer may always be performed.
CTRL[15:11]	NEXTCHAN	The channel number of the next channel in this chain, see 'Chaining' on page 28.
CTRL[16]	USECHAIN	When set: a transfer on channel NEXTCHAN is to be performed after any transfer on this channel irrespective of that channel's ENABLE status bit.

a. Not used by chain-only channels - these fields read back as 0

A run of transfers on a particular channel can be programmed by writing appropriate values to that channel's SRC, DST and COUNT registers and then writing an appropriate CTRL word with the CTRL.ENABLE bit set.

Each of the options below performs a fixed number of transfers which are determined by the

contents of the COUNT register at the time at which the CTRL.ENABLE bit was set. The end of a sequence of transfers of this type occurs when the COUNT register reaches zero. The test for zero is performed after incrementing the COUNT register (if CTRL.COUNTINC = 1) after each transfer. The CTRL.ENABLE bit is reset when COUNT equals zero irrespective of whether CTRL.COUNTINC was set or not, this allows single transfer runs to be performed by initialising COUNT to zero and resetting the CTRL.COUNTINC bit. Common control options are:

Memory to memory transfers

- fixed transfer count, continuously initiated transfers

Register	Contents
SRC	Source memory base address
DST	Destination memory base address
COUNT	2's complement of no. of transfers to perform. (COUNT is incremented between transfers)
CTRL.CLIENT	Not used
CTRL.SIZE	Transfer size (0: byte, 1: half-word, 2: word)
CTRL.SRCINC	1 - increment the source address after each transfer
CTRL.DSTINC	1 - similarly with the destination address
CTRL.COUNTINC	1 - increment the COUNT also
CTRL.USEDRQ	0 - don't use the client request

Memory to peripheral

- fixed transfer count, initiated by a client request

Register	Contents
SRC	Source memory base address
DST	Peripheral address
COUNT	2's complement of no. of transfers to perform. (COUNT is incremented between transfers)
CTRL.CLIENT	Peripheral client request number
CTRL.SIZE	Transfer size (0 : byte, 1: half-word, 2: word)
CTRL.SRCINC	1
CTRL.DSTINC	0 - peripheral address does not change
CTRL.COUNTINC	1
CTRL.USEDRQ	1 - use the client request

Peripheral to memory

- fixed transfer count, initiated by a client request

Register	Contents
SRC	Peripheral address
DST	Destination memory base address
COUNT	2's complement of no. of transfers to perform. (COUNT is incremented between transfers)
CTRL.CLIENT	Peripheral client request number
CTRL.SIZE	Transfer size (0 : byte, 1: half-word, 2: word)
CTRL.SRCINC	0 - peripheral address does not change
CTRL.DSTINC	1
CTRL.COUNTINC	1
CTRL.USEDRQ	1 - use the client request

Peripheral to peripheral

- fixed transfer count, initiated by a source peripheral client request

Register	Contents
SRC	Source peripheral address
DST	Destination peripheral address
COUNT	2's complement of no. of transfers to perform. (COUNT is incremented between transfers)
CTRL.CLIENT	Peripheral client request number
CTRL.SIZE	Transfer size (0 : byte, 1: half-word, 2: word)
CTRL.SRCINC	0 - peripheral address does not change
CTRL.DSTINC	0 - peripheral address does not change
CTRL.COUNTINC	1
CTRL.USEDRQ	1 - use the client request

7.4 Chaining

Channels can be chained together to perform a number of transfers on different channels all initiated by a single request source. A single channel – the head channel – controls the transfer: subsequent channels in the chain are joined in a linked list structure using the NEXTCHAN and USECHAIN fields in the CTRL registers. The settings in the CTRL register of the head channel determine most of the properties of the transfer: the number of transfers, whether transfers are initiated by a client request or are free-running. Signalling interrupts is, however, controlled by the IRQ/FIQMASK and CHANSTATUS bits of the final channel in the chain. On completing a chain of transfers only the CHANSTATUS bit of the final channel is set (the head and subsequent channels have clear CHANSTATUS bits). Only if the corresponding bit in the IRQ/FIQMASK is set will an interrupt be signalled.

Transfers can be initiated either by a single client request to the head channel or by enabling an free-run transfer on that channel. Transfers on the second and subsequent channels in a chain will occur irrespective of the values of COUNT, CTRL.ENABLE, CTRL.USEDRQ or the CHANSTATUS bit for those channels.

Any channel, including a head channel, may be specified as a chained channel; however channels 22…31 are designated chain-only channels and as such have no ENABLE, CLIENT, COUNTINC, and USEDRQ fields in their CTRL registers; any information written to those fields is discarded. If non chain-only channels are included as non head channels in a chain, it is essential that their ENABLE bit be reset so as to prevent that channel from being inadvertantly activated as an independent transfer.

As an example, a chained transfer of 10 words between peripheral addresses P1 and P2 and memory addresses M1 and M2 using channels 7 and 14 initiated by client request 4 could be setup by setting the registers in the following order:

```
set up chained channel(s) first...
CHAN[14].SRC = P2
CHAN[14].DST = M2
CHAN[14].CTRL.ENABLE = 0      - important to prevent accidental transfers
             .SIZE = 2        - word transfers
             .SRCINC = 0      - fixed source address
             .DSTINC = 1      - changing destination address (memory)
             .NEXTCHAN = 0    - not used (end of chain)
             .USECHAIN = 0    - end of chain

...then set up head channel
CHAN[7].COUNT = -10           - 10 passes along this chain of transfers
CHAN[7].SRC = P1
CHAN[7].DST = M1
CHAN[7].CTRL.ENABLE = 1       - enabled to receive requests
            .CLIENT = 4       - client request 4
            .SIZE = 2         - word transfers
            .SRCINC = 0       - fixed source address
            .DSTINC = 1       - changing destination address (memory)
            .COUNTINC = 1     - use the count register
            .USEDRQ = 1       - receive requests for client 4
            .NEXTCHAN = 14    - the 2nd channel in the chain
            .USECHAIN = 1     - use chaining
```

Note that it is important that the chained channels be setup before the head channel is enabled, otherwise the transfers may start before the chained channels are fully initialised.

7.5 Global registers

There are seven global registers which affect the behaviour of and reflect the status of all 32 channels; they are:

Reg Name	R/W	Reg Address	Offset Address
GENCTRL	R/W	ffc00200	DMABASE+200
CHANSTATUS	R	ffc00204	DMABASE+204
IRQMASK	R/W	ffc00208	DMABASE+208
FIQMASK	R/W	ffc0020c	DMABASE+20C
IRQREQ	R	ffc00210	DMABASE+210
FIQREQ	R	ffc00214	DMABASE+214
SETREQ	R/W	ffc00218	DMABASE+218

All of the channel and global registers can be read or written to at any time (except the read only registers for which a write is ignored). The transfer engine takes a copy of the registers for the channel for which it is performing a transfer, these values (after incrementing) are returned back to the register bank in parallel with the transfer is being performed. Channel CTRL register writes and global register writes are guaranteed to successfully update the register bank irrespective of whether the controller or any particular channel is enabled.

GENCTRL Register

Bits	Field	R/W	Function
GENCTRL[15:0]	BASE	R/W	16 MS bits of short channel SRC and DST addresses
GENCTRL[23:16]			Reserved - reads back as 0
GENCTRL[28:24]	TFRCHAN	R	The channel number of the current/last transfer
GENCTRL[30:29]			Reserved - reads back as 0
GENCTRL[31]	GENABLE	R/W	When set: enables transfers on all channels having CTRL.ENABLE set. When the bit is reset from the target interface, the DMA controller client requests (as visible on the asynchronous side of the SPI) and all channel CTRL.ENABLE bits are reset. This bit is used for system initialisation. Clearing the bit whilst the transfer engine is in use will not abort current transfers, but in some circumstances may cause the transfer engine to burn unecessary power whilst declining new client requests.

CHANSTATUS Register

This register contains one bit per DMA channel (at the bit position of that channel's channel number) indicating whether a run of transfers on that channel has recently been completed (i.e. when COUNT reaches zero and ENABLE is reset by the DMA controller). 'Recently' means since the last time the CTRL register for that channel was accessed via MARBLE. Reading or writing to a channel's CTRL register will reset the CHANSTATUS bit for that channel (after some delay).

The CHANSTATUS word is not zeroed on power up, reset, or after resetting GENABLE. Writing to all the channel CTRL registers to clear CHANSTATUS bits is, therefore, a necessary power up initialisation step.

IRQMASK Register

Interrupt masks for IRQ: each bit in these registers corresponds to an individual DMA channel (in a similar way to CHANSTATUS). A set bit indicates that the appropriate interrupt, DMAirq or DMAfiq, should be signalled when the corresponding bit in the CHANSTATUS register becomes set, i.e. a channel transfer run is completed. The status of the interrupt lines DMAfiq and DMAirq (see 'DMA controller block diagram' on page 24) is reassessed each time IRQMASK is written to and after each CTRL register read or written.

FIQMASK Register

Interrupt mask for FIQ (similar to IRQMASK).

IRQREQ Register

IRQ interrupt request status: the bit-wise AND of CHANSTATUS and IRQMASK. A set bit indicates the signalling of an interrupt for the corresponding channel. IRQ is set when IRQREQ $\neq 0$. DMAirq and DMAfiq are always de-asserted when GENABLE is low. IRQREQ is calculated on demand from the current values in CHANSTATUS and IRQMASK and so constitutes no extra testable state to the DMA controller.

FIQREQ Register

FIQ interrupt request status (similar to IRQMASK).

SETREQ

This register contains one bit per DMArq (22 bits, the remaining MS 10 bits are ignored on a write and read back as 0). A set bit forces a channel request for that client request. In order to be recognized by the SPI, the same timing rules as those for true client requests must be obeyed – see 'Request timing' on page 31. Programming a previously reset bit to a '1' will cause the fake client request to be recognised. Normally all bits should be reset, otherwise normal DMArq requests are blocked. The primary purpose of the register is to facilitate testing (see 'Testing the DMA controller' on page 68).

7.6 Sequence of events during a single transfer

A complete DMA transfer is a repeated sequence of transfers of individual data items. An individual transfer (with or without chaining) consists of the following sequence of events:

1. The Transfer Engine awaits internal channel requests from the SPI. These are generated synchronously with the SOCB clock either by external DMA client requests or by channels which are programmed to ignore DMArqs (CTRL.USEDRQ = 0) and which are therefore always ready to run.
2. The Transfer Engine chooses the highest priority channel ready to run (see 'Prioritisation' on page 31) and takes a copy of that channel's SRC, DST and COUNT registers.
3. The transfer engine reads, then writes the source and destination addresses transferring a byte, half-word or word datum.
4. In parallel with 3; the register bank returns the updated SRC, DST and COUNT values to the registers.
5. Steps 2, 3 and 4 are repeated for chained channels.
6. The register bank updates channel CTRL.ENABLE based on the COUNT value. The channel request is cleared for request-driven channels or cleared at the end of a transfer run for free running channels. The CHANSTATUS bit for the channel is updated.
7. If all transfers in a run (including any chained channels) are complete, IRQ and FIQ signals are generated if unmasked.
8. The sequence is repeated from step 1.

7.7 Prioritisation

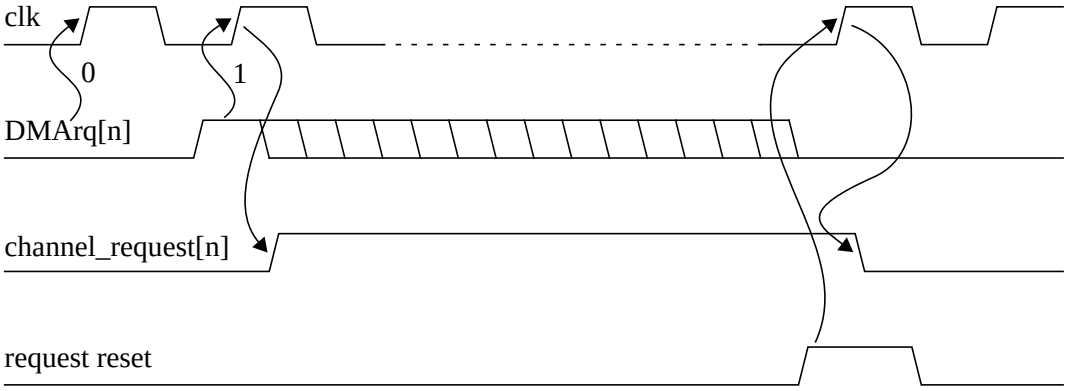
When more than one channel request is simultaneously active at the time that the transfer engine becomes free to perform the next transfer, a decision must be taken as to which request to service next. This is possible as requests are resolved by the clocked SPI and so the notion of simultaneity of requests is defined by the cycle period. Simultaneous requests are therefore resolved by imposing a static prioritisation on channel requests. Channel 0 has the highest priority and will *always* be serviced first when a request is signalled for channel 0. Each of the channels 1 to 31 have descending priorities making the long channels (usually used for memory to memory transfers) the lowest priority channels. No attempt is made to make this system more fair by, for example, rotating priorities between channels. Care must therefore be taken to avoid setting up free running transfers on high priority channels which are likely to starve client request driven transfers on lower priority channels. There is no distinction, in terms of prioritisation, between channel transfer requests initiated by DMArq and those that result from free-running transfers.

7.8 Request timing

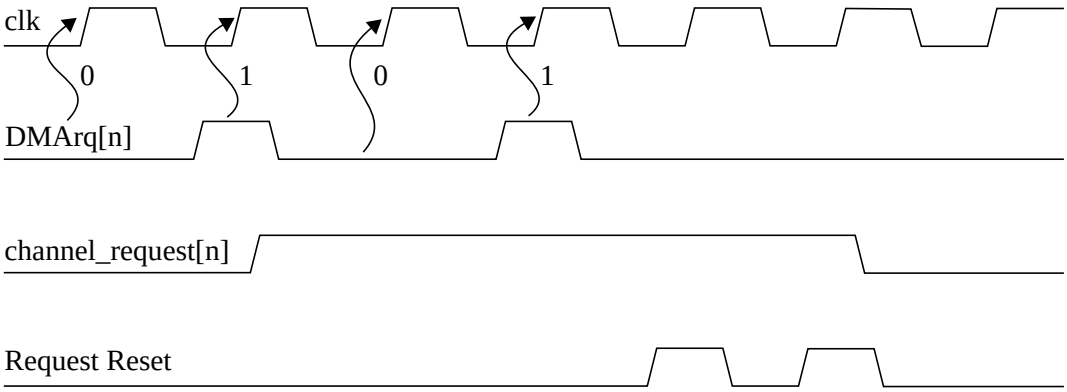
The SPI works by observing a positive going edge on an incoming DMArq over a period of two SOCB clock cycles: i.e. it must be low on one active clock edge and high on the subsequent active clock edge. A *channel request* to the Transfer Engine is generated which, if a second channel

request for the same channel has not occurred, will be reset by a *request reset* signal from the register bank.

It is the responsibility of the initiating peripheral to manage the handshaking of data transfers. The perpiheral may remove DMArq as soon as it is guaranteed that it has been recognised. Normally, the peripheral will not assert DMArq again until it recognises that the transfer of a piece of data is complete. If the peripheral is relying on notification from the CPU, as a result of a DMAfiq/ DMAirq condition that the transfer is complete (COUNT=0), it is possible that it may generate a spurious DMArq after the last transfer. This will be recognised by the SPI and lie dormant waiting to be activated when that channel is next enabled. If this is a problem, the latched request may be cleared by writing CTRL.USEDRQ=0, CTRL.ENABLE=0 to the channel's control register.[1]



It is allowable for the incoming request DMArq to fall as soon as it has been recognized and the next rising edge of that request will (after two cycles) indicate the next request for that client. In that case, the request to the transfer engine channel request remains high until a matching request reset signal has been received from the register bank



1. Of course, this action will also explicitly reset any pending requests if this channel is currently programmed for a free-running memory to memory transfer.

7.9 Power On Initialisation

At power on, the only user visible status bits with guaranteed values are the CTRL.ENABLE bits for each channel and the GENCTRL.GENABLE, they are all reset. The SPI will also contain no pending requests but although the CTRL.ENABLE bits are reset, applied client requests can propagate through the SPI as far as the enable gating which is immediately before the channel request handling state machines. This is because the client to channel mapping which will be randomly set in the CTRL.CLIENT control register fields.

Whilst CENCTRL.GENABLE is reset no transfers can take place and no interrupts can be signalled although if a channel CTRL.ENABLE bit is set a channel request can get as far as the transfer engine and can initiate a request for registers from the register bank. This is the first stage of performing a transfer and, although that transfer will be aborted by the register bank noticing the GENCTRL.GENABLE status, power will be dissipated. It is for these reasons that GENCTRL.GENABLE should be viewed as an initialisation feature rather than a power saving mode.

At this point the DMA controller can be initialised by setting the remaining control registers to ‘safe’ states and then setting GENABLE to wake the controller up.

7.10 Reset

Hardware Reset: A hardware reset will cause exactly the same GENCTRL.GENABLE/ CTRL.ENABLE bits reset as power on although the DMA controllers registers will retain their other control bits in the same states as before the reset.

Programmed reset: GENABLE can be reset by the programmer as a software reset or “emergency stop” feature. All channel CTRL.ENABLE bits will be reset and any pending requests in the SPI will be discarded. Note though that if GENCTRL.GENABLE is reset whilst a transfer is taking place that that transfer will **not** be aborted although no subsequent transfers will take place until GENCTRL.GENABLE is set. Resetting and then setting GENCTRL.GENABLE is the easiest way to clear all ENABLE bits and cease new transfers without losing other control information.

7.11 Clocking

Because some DMA functions interact with the SOCB clock, they will not function unless the clock is running. Some simple register transactions, such as writing GENCTRL, synchronize with the clock to ensure reliable operation, and they will therefore take a variable time depending on the phase and frequency of the clock when they are initiated.

8. 16 Kbyte ROM module

The AMULET3H ROM module is a simple MARBLE target device. It contains application software.

8.1 Features

- 4K 32-bit words
- access time limited by MARBLE transfer latency

8.2 MARBLE interface

The ROM connects to the MARBLE bus through a simple target interface. It can therefore be accessed by any MARBLE initiator, though only the AMULET3 processor is likely to be interested in the ROM in normal operation.

8.3 Notes

- the ROM interface is read-only. It gives an Abort response to any attempt to write to the ROM.
- the ROM is implemented as word (32-bit) read only. Halfword and byte accesses will give the expected result, but will not save power compared with a word access.

9. The ADC/AEDL interface

9.1 Introduction

The ADC/AEDL interface is a MARBLE target which provides an interface to an analogue/digital converter (ADC) which is on-chip but not part of the AMULET asynchronous block. The ADC produces a 9-bit result. One of the functions of the interface is to allow reading of the ADC to be synchronised to a signal which indicates that the ADC is ready. This is termed an Asynchronous Event Driven Load (AEDL). In this case the processor reads the ADC but the interface does not return the data until the ADC has valid data. The processor is stalled until the data is available.

9.2 Interface signals

The following signals are provided between the MARBLE target and the ADC/AEDL logic block. The directions shown are relative to the MARBLE target.

Signal	Direction	Function
AEDL_en	In	Active high enable for AEDL
AEDL_sync	In	Selects level (0) or edge (1) AEDL synchronization
ADC_rd	Out	ADC read, active high
AEDL_ch[1:0]	Out	AEDL channel (reflects processor A[7:6])
AEDL_rdy[2:0]	In	AEDL event signals
AEDL_ack[2:0]	Out	AEDL acknowledges
ADC_din[9:0]	In	ADC data (9 bits, [8:0]) & time-out (bit [9])

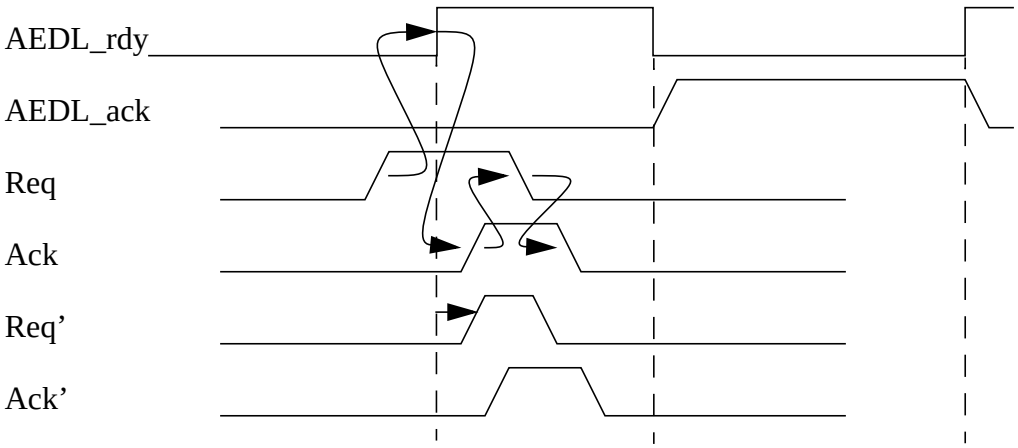
The role of each of these signals is descibed in detail later in the text, but in brief:

- AEDL_en is used to disable and reset the AEDL interface when low and to enable normal operation when high.
- AEDL_sync selects between two synchronization modes (described later) and should only change when AEDL_en is low and the unit is idle.
- ADC_rd indicates that a read of the ADC is in progress. The falling edge of ADC_rd may be used to start the next ADC conversion.
- AEDL_rdy[2:0] are the 3 signals used for synchronization; when a synchronized ADC read is requested the response will be held up until there is a rising edge or high level (as selected by AEDL_sync) on the addressed AEDL_rdy[] line.
- AEDL_ack[2:0] are responses confirming that a particular AEDL_rdy[] event has been recognised.
- ADC_din[9:0] is the ADC data and error bus.

If an ADC read operation is requested by the software (visible to the synchronous subsystem as ADC_rd being high) but the requested AEDL_rdy[] response is not generated within some time-out period (several microseconds) the synchronous interface is responsible for generating a time-out on all of AEDL_rdy[2:0] and setting ADC_din[9] to indicate that this has happened.

9.3 Interface timing

The key timing relationship is between the AEDL rdy/ack signals and the asynchronous Req/Ack handshake. AEDL_ack will be active for the AEDL_rdy low period following a successful synchronization. The synchronization event behaviour is determined by AEDL_sync:



- If AEDL_sync is high, synchronization takes place at the first positive edge of AEDL_rdy following Req going high as illustrated by the Req/Ack signals shown above.
- IF AEDL_sync is low, synchronization takes place when AEDL_rdy and Req are first high at the same time. The Req'/Ack' signals shown above illustrate what happens when Req' high follows AEDL_rdy high.

The original specification calls only for the first (edge) synchronization model. However, this requires that the software *always* completes within one AEDL_rdy cycle time. The level synchronization model requires that the software loop has an *average* cycle time no longer than the AEDL_rdy cycle time, but the *worst-case* software cycle time can be the AEDL_rdy cycle time *plus* the AEDL_rdy high time. Thus the level synchronization model better copes with occasional perturbations to the software cycle time as are likely to arise in any asynchronous processing system due to arbiter metastability delays and the like.

When the Req positive edge is very close to the AEDL_rdy positive edge (if AEDL_sync is high) or negative edge (if AEDL_sync is low) then metastability can ensue in the synchronization circuit. The decision about whether or not a valid synchronization has occurred is non-deterministic, but the circuit will make a valid decision using standard analogue arbitration techniques. As there is a fixed time this cannot be done with complete reliability, but the MTBF will be very high (comparable with the failure rate of a clocked synchronizer which has the same settling time) and in normal operation metastability should be very rare. To maximise the MTBF the AEDL_rdy high period should not be made too short, and giving it a 50/50 duty cycle will give good results in all modes of operation.

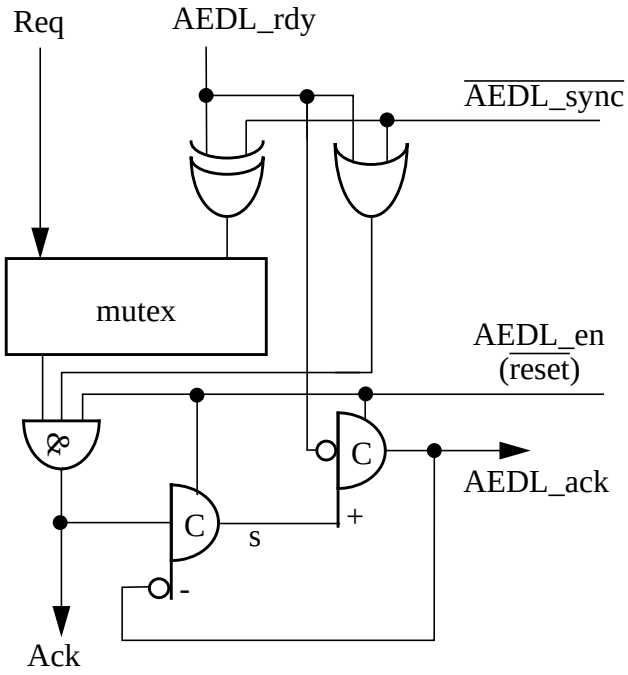
Note that the AEDL_ack active phase corresponds to the AEDL_rdy low period (to within a few gate delays) in both modes of operation. It is thus synchronous to the clock used to generate AEDL_rdy and can safely be sampled on any clock edge during the AEDL_rdy low period after the first, though sampling it towards or at the end of this period is best.

9.4 Circuit

The synchronization circuit, which supports either mode under the control of AEDL_sync, is shown overleaf. Three of these synchronization circuits are required, one for each of AEDL_rdy[2:0].

The operation of the circuit can be understood by considering the two cases of AEDL_sync high and low separately (AEDL_sync should only change when this circuitry is in reset due to AEDL_en being low):

- AEDL_sync low: level synchronisation. When AEDL_rdy is high the mutex is released and if Req goes high it will pass through the mutex and give Ack immediately. Ack sets s, which in turn sets AEDL_ack as soon as AEDL_rdy goes low. s is cleared when Ack is low (the asynchronous Req/Ack handshake has completed), and AEDL_ack is reset when AEDL_rdy goes high.
  If Req goes high when AEDL_rdy is low, AEDL_rdy blocks the mutex and Ack is held until

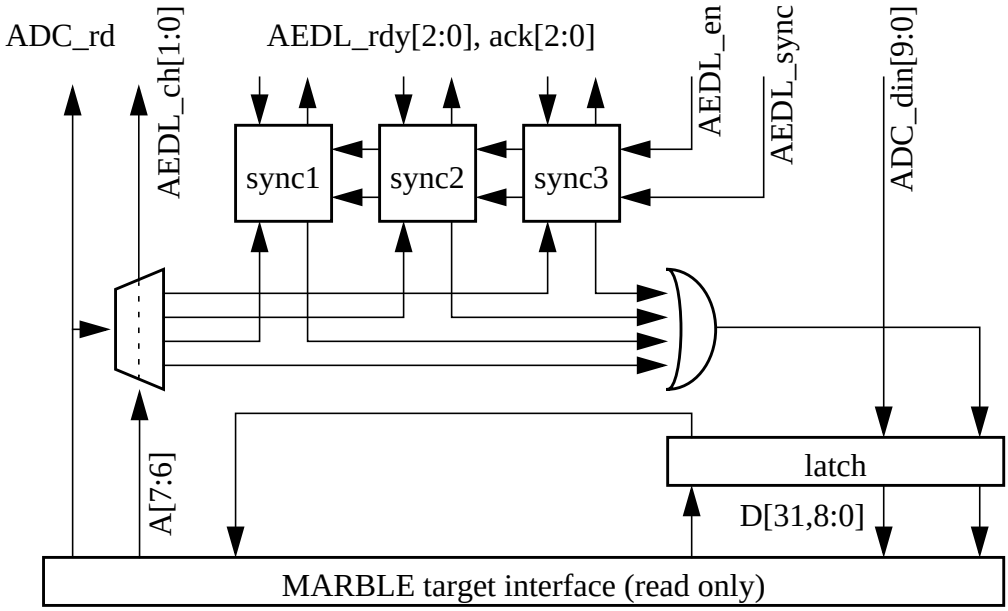


AEDL_rdy goes high, releasing the mutex.

- AEDL_sync high: edge synchronisation. Here Req must go high while AEDL_rdy is low in order to claim the mutex. Once the mutex is claimed, Ack is blocked by the AND gate until AEDL_rdy goes high, then Ack goes high and s is set, followed by AEDL_ack when AEDL_rdy goes low, as before.

The mutex will handle metastability safely with only a small delay in Ack occurring from time to time while the metastability is resolved.

The complete AEDL logic (on the asynchronous side) comprises three of the above circuits, a MARBLE (read-only) target interface, and a small amount of additional logic as shown below.



9.5 Memory map and registers

The base of the ADC/AEDL logic is at ff800000 and the register mapping is as follows:

Address	Read/Write	Function
ff800000	Rd	Read ADC, ignore AEDL
ff800040	Rd	Read ADC, sync to AEDL_rdy[0]
ff800080	Rd	Read ADC, sync to AEDL_rdy[1]
ff8000c0	Rd	Read ADC, sync to AEDL_rdy[2]

This set of locations repeats at ff800100, ff800200, etc., up to the top of the area at ff9fffff. A write to any of these locations writes to the AEDL test control register (see section 15.10 on page 69).

The data that is read will have the ADC values on bits [8:0] and the time-out signal on bit [31]. Bits [30:9] will read as zero (except in test mode).

9.6 Action of the AEDL

The AEDL_rdy[n] signal goes high each time the ADC produces valid data. When the chip is reset the AEDL_rdy[2:0] signals are all low. When the processor does a read the action is as follows: The read is blocked until AEDL_rdy[n] goes high. If the transition to high occurred before the read was attempted then either:

- the read will return the data immediately (level synchronization), or
- it waits for the next transition to high before returning the data (edge synchronization).

The choice between level and edge synchronization is selectable using AEDL_sync.

When AEDL_rdy[n] is recognised, AEDL_ack[n] will be returned. The synchronous system can therefore recognise exactly which AEDL_rdy[] pulses have been recognised and which have not, and an error condition may be generated when a pulse is missed.

It is expected that the ADC will generate a new sample as a result of its current sample being read using the ADC_rd signal. Thus a constant stream of samples may be produced and read by the processor by performing a sequence of reads.

The signal AEDL_en is provided to allow the AEDL synchronising logic to be re-initialised. When this signal is low the synchronising logic is reset and the AEDL_rdy[2:0] and AEDL_ack[2:0] signals must also go low at this time. When AEDL_en is high the synchronising logic is active. The ADC should not be accessed while AEDL_en is low (except that access via addresses that do not activate the AEDL is allowed). AEDL_sync should *only* be changed when AEDL_en is low.

9.7 Programming sequence

The software interface to the ADC is as follows:

1. Set signal AEDL_en low (chip reset should also do this). This initialises the AEDL interface and the AEDL_rdy[2:0] and AEDL_ack[2:0] signals should go low.
2. Write zero to the AEDL test control register (address ff800000) to ensure test mode is disabled.
3. Set signal AEDL_en high. This enables the AEDL interface.
4. Perform a read from the ADC (address ff800040, ff800080 or ff8000c0). This will wait for the transition on AEDL_rdy[n] and then return data from the ADC. Following the read the ADC will start another conversion.
5. Repeat step 3 until no more samples are required. The AEDL_rdy[n] signal will pulse for each sample.
6. When no more samples are required stop reading from the ADC.

10. The External Memory Interface (EMI)

The AMULET3H external memory interface supports the direct connection of external memory and peripheral devices. The timing of the accesses to these off-chip devices is defined by an on-chip timing reference delay (Dt), which is only activated when an off-chip access is required.

10.1 Features

- direct interface to conventional SRAM, DRAM, ROM and peripheral parts
- on-chip timing reference delay (calibrated against a reference signal by software)
- timing characteristics and bus width programmable separately for different memory regions

10.2 Introduction

The EMI is designed to interface to 8- and 16-bit wide memories. There is a 16-bit wide bidirectional data bus (d[15:0]) and a 29-bit address bus a[28:0]. Up to 8 different memory devices may be attached to the interface. Each one is selected by an active low memory select signal Nms[7:0] and each of these signals corresponds to a different part of the processor's address space.

In addition there is a read/write line (rNw) to indicate the direction of data transfers and an active low output enable (Noe) to enable read data onto the data bus. There are two column address strobes (Ncas[1:0]) for use with DRAM and two programmable byte strobes (Nbs[1:0]) which can serve as byte write strobes or byte select signals.

Two inputs (bt[1:0]) are used to control where the processor fetches its first instruction from following reset.

The processor and memory interface are configured for Little Endian operation.

10.3 Interface signals

The following signals are available to support the connection of off-chip devices:

Signal	Type	Function
d[15:0]	IOZ	16-bit off-chip data bus
a[28:0]	O	29-bit off-chip address bus[a]
rNw	O	Read/not write
Noe	O	Output enable (active low)
Nms[7:0]	O	Memory select/row address strobe (RAS)[b]
Nbs[1:0]	O	Byte select/byte write strobe
Ncas[1:0]	O	Column address strobe (CAS)
bt[1:0]	I	Boot mode select
Ntest	I	Test mode enable (high for normal operation)

a. a[28:20] are multiplexed with peripheral I/O signals on DRACO.
b. Nms[7:4] are multiplexed with peripheral I/O signals on DRACO.

10.4 Supported memory types

The following memory types should be adequately supported by the interface with no external logic:

- DRAM - 8 bit with RAS, CAS, RnW, optional nOE
- DRAM - 16 bit made of two 8 bit devices
- DRAM - 16 bit with RAS, UCAS, LCAS, RnW, optional nOE
- SRAM - 8 bit with nCE, nWE, optional nOE
- SRAM - 16 bit made of two 8 bit devices
- SRAM - 16 bit with nCE, nWE, nUB, nLB, optional nOE
- Peripheral - most 8 bit devices with SRAM type signals
- Peripheral - many 16 bit devices

10.5 Memory decoding

The 4GB address space of the processor is divided into 8 equally sized regions. These regions map to separate external memory devices and there is an active low select signal for each of regions 0 to 7 (Nms[7:0]). The addresses of each region are shown below:

Region	From	To	Memory select	Notes
0	00000000	000fffff	(on-chip RAM)	see below
0	00100000	001fffff	(on-chip ROM)	see below
0	00200000	1fffffff	Nms[0]	
1	20000000	3fffffff	Nms[1]	
2	40000000	5fffffff	Nms[2]	
3	60000000	7fffffff	Nms[3]	
4	80000000	9fffffff	Nms[4]	
5	a0000000	bfffffff	Nms[5]	
6	c0000000	dfffffff	Nms[6]	
7	e0000000	ff7fffff	Nms[7]	
7	ff800000	ffffffff	(control registers)	see below

Each externally accessible region has an address range of 512MB. Region 7 contains on-chip control registers and similar resources mapped into the top 8 Mbytes of the address space. The external memory is not accessible at addresses above 0xff7fffff. Region 0 contains on-chip ROM and RAM which are at addresses 0 to 0x00fffff (RAM) and 0x00100000 to 0x001fffff (ROM).

10.6 Bootstrap

When the processor is reset it starts fetching code from address 0. Normally, on-chip RAM is mapped to address 0 but for the first fetch (only) after reset the RAM is disabled and the first word is fetched from ROM. The boot select pins determine which ROM is used as shown in the table below.

If off-chip ROM is specified, the device attached to Nms[0] is used. In this case the on-chip timing reference must be used and because it will not have been configured at this point, the slowest

bt[1]	bt[0]	Boot ROM
0	0	Off-chip ROM (8 bit) in region 0
0	1	Off-chip ROM (16 bit) in region 0
1	0	(not used)
1	1	On-chip ROM

possible timing will be used and the first read from external ROM could take up to 20us.

10.7 Action of reset

When the memory interface is reset by the global chip reset signal all EMI control registers other than RTR0 are zeroed. When the chip comes out of reset the width field of RAR0 (see below) is set from the bt[1:0] pins.

10.8 Timing reference

All external memory operations are timed with respect to an on-chip delay line. The length of this delay line may be controlled by software. The basic delay of the delay line may be scaled by a factor of 1, 2, 4, or 8 to provide a wide range of delays. There are 54 delay elements in the delay line each providing a delay of around 250ps (typical silicon) so the basic delay varies from 0 to 13.5ns in 250ps increments. There is an overhead in the delay loop which is typically 5ns so the range is actually 5 to 18.5ns. The prescaler multiplies the delays by 1, 2, 4, or 8 and has a minor impact on the overhead so the delays which can be achieved are as follows:

Prescale	Minimum	Step	Maximum
1	5ns	0.25ns	18.5ns
2	6ns	0.5ns	33ns
4	7ns	1.0ns	61ns
8	8ns	2.0ns	116ns

These are preliminary figures based on SPICE simulation of typical silicon. Multiply the delays by 1.25 for slow silicon and 0.71 for fast silicon.

The delay line is controlled by a register mapped into the processor’s address space. It is expected that the prescale factor will be set when the system boots and not altered thereafter. The length of the delay line can be changed while the system is running to cope with changes in supply voltage and/or temperature.

The delay provided by the delay line (with prescale applied) is used as the basic timing reference (Dt) for the state machine which provides timing for the various types of memory cycle that the interface supports.

Delay control register - at fff00020

The delay control register (DCR) has the following format:

	7	6	5	4	3	2	1	0
DCR	CntEn	Cal	ClrCtr	C1	C0	L	M	R
reset to:	0	0	0	0	0	0	0	0

L, M and R (D2-D0) are signals which control the delay line. When they are all zero the delay line is set to its maximum delay; when they are all one the delay line is set to its minimum delay. C0 and C1 (D3, D4) control the prescaler as follows:

C1	C0	Prescale
0	0	8
0	1	4
1	0	2
1	1	1

Delay calibration

The CntEn bit (D7) enables a 16-bit binary counter (DTR0, DTR1) and the Cal bit (D6) causes the delay line to free run, clocking the counter after each cycle. This allows the delay to be measured by comparing the counter with a source of real-time information. While the delay line is free running in this mode it is still possible to make external memory accesses without significantly affecting the accuracy of the count, but the counter should not be read while the CntEn bit is set. The counter is zeroed when the chip is reset and also by taking bit 5 (ClrCtr) high. ClrCtr should be returned to low to enable counting. The counter cannot otherwise be written. (The counter may also be clocked for test purposes by toggling CntEn (D7), but only when the internal treq state is correct.)

The delay line is controlled by appropriate manipulation of L, M and R. At reset these bits are zero which initialises the delay line to its maximum length. Then the following sequence is used to reduce and increase the delay:

L	M	R	Effect
0	0	0	Initialise to maximum delay (reset state)
0	0	1	Idle
0	1	1	Reduce delay by 1 step
0	1	0	Idle
1	1	0	Reduce delay by 1 step
1	0	0	Idle
1	0	1	Reduce delay by 1 step
0	0	1	Idle
1	0	1	Increase delay by 1 step
1	0	0	Idle
1	1	0	Increase delay by 1 step
0	1	0	Idle

The delay line is stable with only 1 bit of L, M, R set and setting two bits in the appropriate order causes the delay to be increased or decreased. The processor must keep track of where the delay line is ‘tapped’ to avoid attempting to increase or decrease when at maximum or minimum delay.

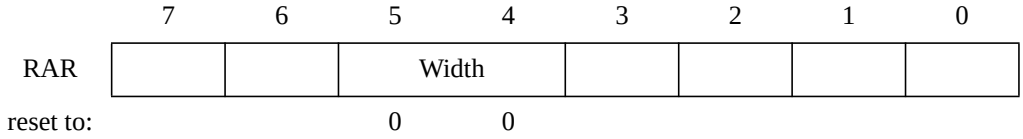
10.9 Memory cycle types

Two basic types of memory are supported, static and DRAM. The DRAM interface is used with

conventional page-mode DRAM devices and provides RAS and CAS signals. There is no hardware support for refresh - this must be performed by software. The static interface is designed to work with conventional static RAMs, ROMs and simple peripheral devices such as UARTs.

10.10 Memory width, byte selection and funnelling

The interface supports 16- and 8-bit wide memories. Each region may be configured to one of these widths by writing to the Region Architecture Register corresponding to the particular region.



The width bits (D5 and D4) determine the width as follows:

D5	D4	Width
0	0	8 bits wide
0	1	16 bits wide
1	x	(reserved)

For region 0 the width bits are initialised from the boot select pins when the chip comes out of reset. For all other regions the bits are initialised to 0.

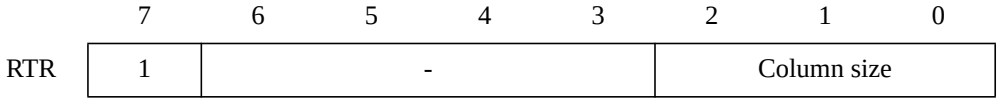
Where the memory is 16 bits wide and only a single byte is to be transferred it is necessary to indicate which byte is active. For a read cycle no byte selection information is provided unless the Nbs[] signals are configured as byte selects. In this case, one of these signals will go low to indicate which byte is to be read. Otherwise the full 16-bit quantity will be read and in this case the processor will select the appropriate byte.

For a write cycle it is essential that only the selected byte be accessed. If the memory system has separate write strobes for each byte, the Nbs[] pins should be configured as byte write strobes and one connected to the write strobe of each byte of memory. Otherwise, the rNw signal may be used as the overall write indicator to the memory and the Nbs[] pins configured as byte selects. These are then connected to the byte enable pins of the memory device(s).

Where the processor wishes to transfer a data item which is wider than the memory in a particular region the memory interface performs multiple accesses in order to transfer the data item. For example writing a 32-bit word to an 8-bit wide memory would cause 4 external memory cycles to take place. There is nothing special about these memory cycles and they are indistinguishable from multiple transfers of separate data items. Because it is always known that the transfers will be in the same page, page-mode accesses will be used if the memory is DRAM.

10.11 DRAM interface

The DRAM interface is selected for a particular region by setting bit 7 of the Region Timing Register (RTR). In this case, other bits select the column size of the DRAM so that the appropriate addresses are multiplexed onto the address pins.



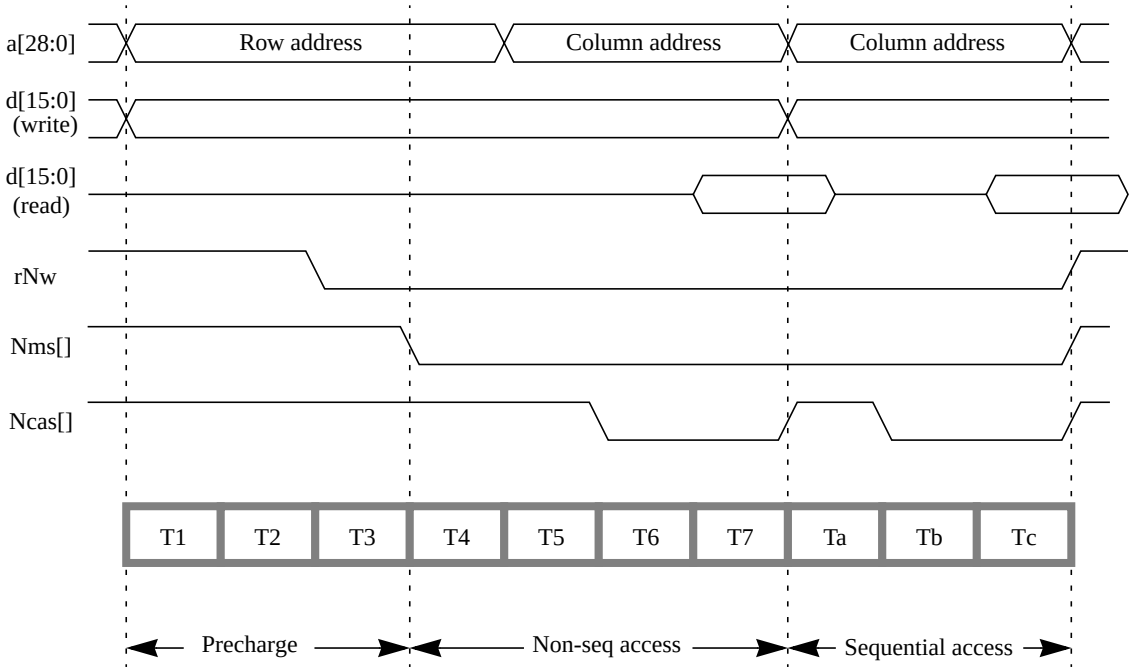


Figure 1: DRAM timing illustration

The column size encodings are as follows:

D2	D1	D0	Column size
0	0	0	8 bit
0	0	1	9 bit
0	1	0	10 bit
0	1	1	11 bit
1	x	x	12 bit

The DRAM interface uses the memory select signal Nms[] as row address strobe and Ncas[] as column address strobe. The rNw signal is connected to the DRAM write input and the Noe signal to the DRAM output enable. If the region contains a single 8-bit wide DRAM only Ncas[0] is used. If the region is 16 bits wide and contains two 8-bit wide DRAMs, Ncas[0] and Ncas[1] connect to the CAS strobes for each byte. If a single 16-bit wide DRAM is used, Ncas[0] connects to LCAS and Ncas[1] to UCAS.

The DRAM memory cycle takes a minimum of 7 timing reference delays (Dt) as shown in Figure 1. The RAS precharge is 3Dt (T1, T2, T3) and then Nms[] (RAS) falls. After another Dt (T4) the address lines switch to providing the column address and after another Dt (T5) the Ncas[] lines go low. They remain low for 2Dt (T6, T7) and then they rise again. At this point, Nms[] (RAS) may rise if the following memory cycle does not refer to the same page in the DRAM. If the next cycle is in the same page Nms[] will remain low and Ncas[] will fall again after Dt (Ta) and remain low for 2Dt (Tb, Tc). This cycle (Ta, Tb, Tc) continues as long as page mode access to the DRAM is possible, though a break in page mode access is forced every 256 bytes when a[7:2] are all zero.

10.12 Static interface

The static interface is selected for a particular region by clearing bit 7 of the Region Timing Register (RTR). In this case, other bits select the timing characteristics of the various control signals. RTR0 is reset to 0x1f; all other RTR registers are reset to zero.

The Nms[] signal is used as a chip enable/select for the memory device. Noe is an output enable for read cycles and rNw indicates the direction of the transfer. The Nbs[] signals may be configured either as byte write strobes or as byte select signals.

For read cycles the Noe signal is the active strobe; for write cycles Nbs[1:0] are the strobe signals. The length of the strobe period may be configured to a multiple of Dt and optional setup and hold delays may be specified which extend the Nms[] low time in front of and behind the strobes.

	7	6	5	4	3	2	1	0
RTR	0	BSmode	SSH	Setup	Hold	Slow	Timing	
RTR0 reset to:	0	0	0	1	1	1	1	1

The timing bits (D1, D0) determine the basic strobe width in terms of multiples of Dt. The slow bit (D2) allows this basic timing to be multiplied by 4. Setting the setup (D4) and hold bits (D3) specifies that the memory select signal should be extended around the active strobe signals. The SSH bit (D5) specifies the length of the setup and hold in conjunction with the Slow bit. See the diagrams below for more detail of this mechanism. Bit 6 determines how the Nbs[] signals behave. If low they behave as byte write strobes and pulse low in write cycles (only) when an individual byte is being written. If high they are byte select signals and go low in both read and write cycles to indicate which byte is active.

The encoding of D5, D2-D0 is as follows:

D5	D2	D1	D0	Strobes	Setup/Hold
0	0	0	0	1 Dt	1 Dt
0	0	0	1	2 Dt	1 Dt
0	0	1	0	3 Dt	1 Dt
0	0	1	1	4 Dt	1 Dt
1	0	0	0	4 Dt	1 Dt
1	0	0	1	8 Dt	1 Dt
1	0	1	0	12 Dt	1 Dt
1	0	1	1	16 Dt	1 Dt
X	1	0	0	4 Dt	4 Dt
X	1	0	1	8 Dt	4 Dt
X	1	1	0	12 Dt	4 Dt
X	1	1	1	16 Dt	4 Dt

A basic read cycle is shown in Figure 2. In this case the strobe width has been set to 4Dt and no setup or hold time is specified. If the Nbs[] signals are configured as byte select signals, they go low with the same timing as Nms[]. If configured as write strobes they will remain high throughout the cycle. Data is latched on the rising edge of Noe.

The same cycle with setup and hold adds states Ts and Th in which the strobe (Noe in this case) remains high with all other signals in their active state. This is shown in Figure 3.

Write cycles are slightly different in that each write cycle is preceded by a single Dt cycle (Tw) in which all the memory control signals return to their idle state. This is to prevent inadvertent writes while the control signals are changing. A basic write cycle is shown in Figure 4. In this case the

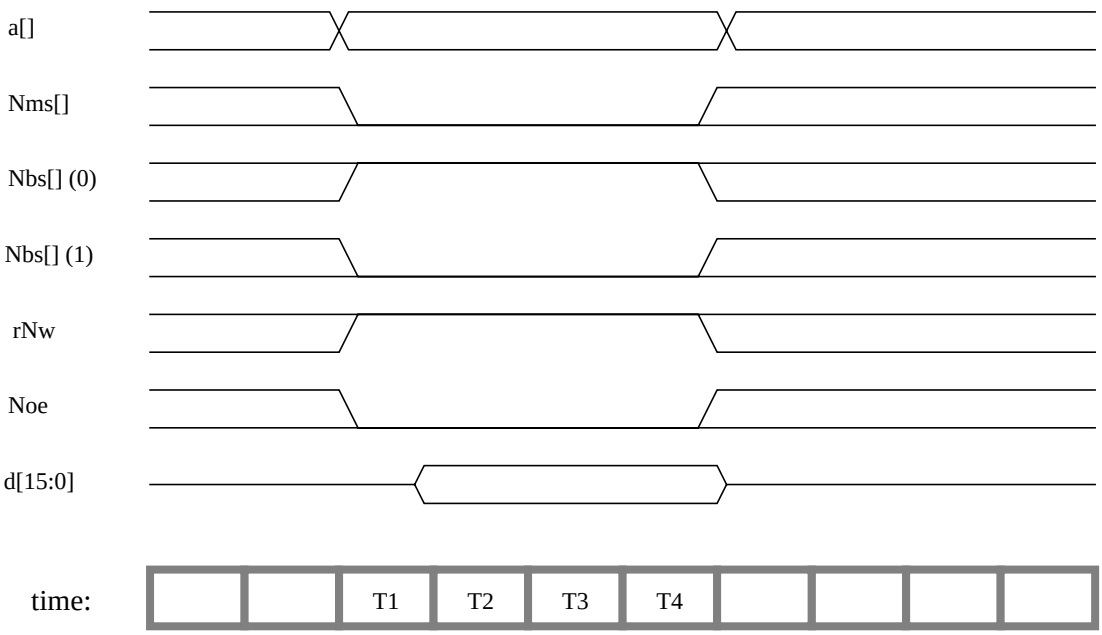


Figure 2: Read timing set at 4Dt

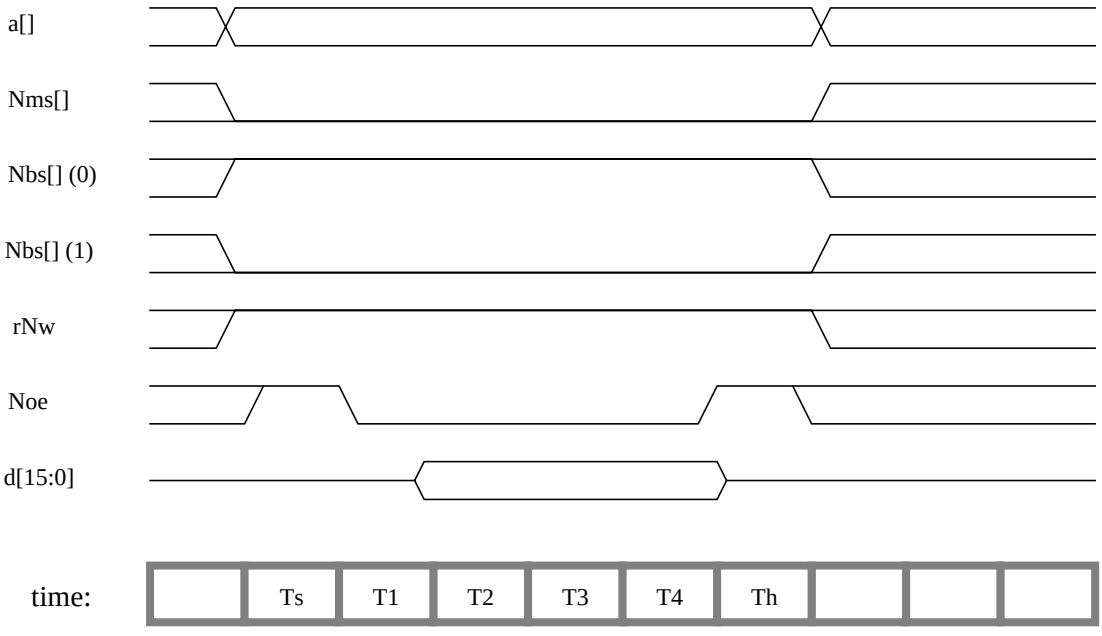


Figure 3: Read timing set at 4Dt, with setup and hold

strobe width has been set to 3Dt

Figure 5 shows a write cycle with setup and hold specified and a strobe time of 1. In this case, however, the slow bit has been set to multiply all times by 4 so that the setup and hold times are 4Dt and the strobe is 4Dt. Note that in a write cycle the Nbs[] signals behave similarly regardless of how they are configured.

10.13 Control registers

The control registers for the memory interface are mapped into region 7 at address 0xfff00000. There are two control registers per region which control the behaviour of the interface when the corresponding region is accessed. In addition to the region control registers, there is a small number of other registers. All registers are 8 bits wide.

Address	Read/Write	Register	Function
fff000f4	R/W	RAR7	Region 7 architecture
fff000f0	R/W	RTR7	Region 7 timing
fff000e4	R/W	RAR6	Region 6 architecture
fff000e0	R/W	RTR6	Region 6 timing
fff000d4	R/W	RAR5	Region 5 architecture
fff000d0	R/W	RTR5	Region 5 timing
fff000c4	R/W	RAR4	Region 4 architecture
fff000c0	R/W	RTR4	Region 4 timing
fff000b4	R/W	RAR3	Region 3 architecture
fff000b0	R/W	RTR3	Region 3 timing
fff000a4	R/W	RAR2	Region 2 architecture
fff000a0	R/W	RTR2	Region 2 timing
fff00094	R/W	RAR1	Region 1 architecture
fff00090	R/W	RTR1	Region 1 timing
fff00084	R/W	RAR0	Region 0 architecture
fff00080	R/W	RTR0	Region 0 timing
fff0002c	R/W	ETR	EMI test register
fff00028	RO	DTR1	Delay timing delay (high)
fff00024	RO	DTR0	Delay timing delay (low)
fff00020	R/W	DCR	Delay control register

10.14 Test mode

The external memory interface also supports a test mode, activated by taking ‘Ntest’ low. This is described in ‘The AMULET3H Test Interface Controller (TIC)’ on page 60.

10.15 Notes

- booting from off-chip ROM requires ROM to be in region 0.

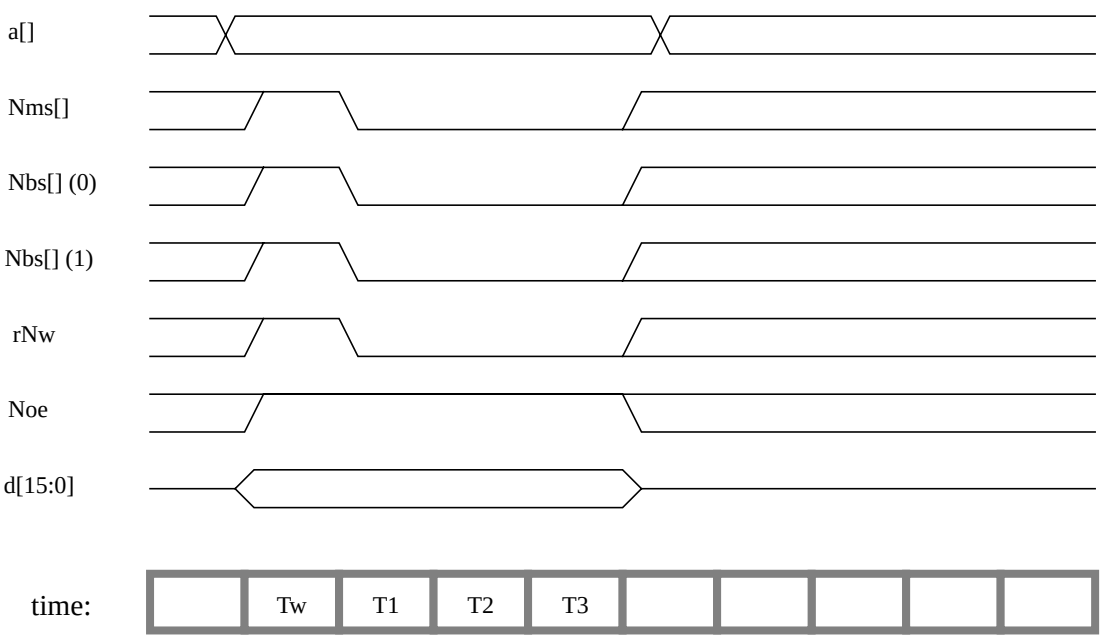


Figure 4: Write timing set at 3Dt

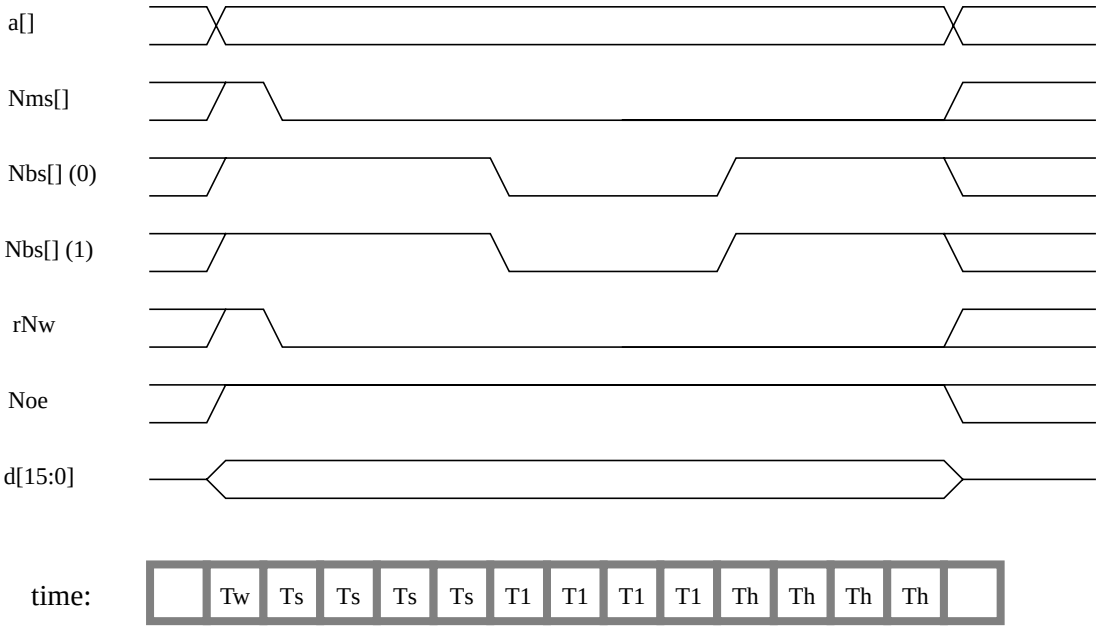


Figure 5: Write timing set at 1Dt, slow, with setup and hold

11. Memory Map

This section summarizes all the details of the AMULET3H memory map.

11.1 Overall memory map

The 32 bit (4GB) ARM address space is divided as follows:

From	To	Function	Location
00000000	000fffff	On-chip RAM (8KB)	Local bus
00100000	001fffff	On-chip ROM (16KB)	MARBLE
00200000	1fffffff	External region 0	MARBLE/EMI
20000000	3fffffff	External region 1	MARBLE/EMI
40000000	5fffffff	External region 2	MARBLE/EMI
60000000	7fffffff	External region 3	MARBLE/EMI
80000000	9fffffff	External region 4	MARBLE/EMI
a0000000	bfffffff	External region 5	MARBLE/EMI
c0000000	dfffffff	External region 6	MARBLE/EMI
e0000000	ff7fffff	External region 7	MARBLE/EMI
ff800000	ff9fffff	AEDL/ADC	MARBLE
ffa00000	ffbfffff	SOCB	MARBLE/MSB
ffc00000	ffcfffff	DMA control registers	MARBLE
ffd00000	ffdfffff	reserved	
ffe00000	ffefffff	CPU control registers	MARBLE
fff00000	ffffffff	EMI control registers	MARBLE

11.2 On-chip RAM

On the local bus and decoded in the range 00000000 to 000fffff. The 8KB RAM block repeats throughout the 1MB space.

11.3 On-chip ROM

On MARBLE and decoded in the range 00100000 to 001fffff. The 16KB ROM block repeats throughout the 1MB space.

11.4 Action at reset

Following reset, the first fetch by the processor (at 00000000) is not taken from RAM but from either on-chip ROM (at 00100000) or from external memory at 00200000 (controlled by bt[1:0]).

11.5 AEDL/ADC registers (base at ff800000)

Address	Read/Write	Function
ff800000	Rd	Read ADC, ignore AEDL
ff800040	Rd	Read ADC, sync to AEDL_rdy[0]
ff800080	Rd	Read ADC, sync to AEDL_rdy[1]
ff8000c0	Rd	Read ADC, sync to AEDL_rdy[2]
ff800000	Wr	Write ATCR

11.6 SOCB registers (base at ffa00000)

The SOCB (synchronous on-chip bus) is accessed over MARBLE via the MSB (MARBLE/SOCB bridge). The address space of the SOCB maps directly onto the AMULET3 address space based at ffa00000 and repeats within the 1MB area.

11.7 DMA controller registers (base at ffc00000)

Address	Read/Write	Bits	Function
ffc00000	R/W	16	Channel 0 source
ffc00004	R/W	16	Channel 0 destination
ffc00008	R/W	16	Channel 0 count
ffc0000c	R/W	17	Channel 0 control
ffc00010	R/W	16	Channel 1 source
...	...	...	...
ffc001f0	R/W	16	Channel 31 source
ffc001f4	R/W	16	Channel 31 destination
ffc001f8	-	-	[Channel 31 count N/A]
ffc001fc	R/W	17	Channel 31 control
ffc00200	R/W	17	GenCtrl
ffc00204	R/W	32	Channel status
ffc00208	R/W	32	IRQ mask
ffc0020c	R/W	32	FIQ mask
ffc00210	R	32	IRQ request
ffc00214	R	32	FIQ request
ffc00218	R/W	32	Set request (for test)

Channels 0 to 15 are short channels with 16-bit source, destination and count registers. Channels 16 to 21 are long channels with 32-bit source, destination and count registers. Channels 22 to 31 are chain-only channels with 16-bit source and destination registers and no count register.

11.8 CPU debug and control registers (base at ffe00000)

These registers allow access to the debug, test and control facilities of the AMULET3 core.

Address	R/W	Function	Location
ffe00000	R/W	Breakpoint control register (BPC)	MARBLE
ffe00004	R/W	Watchpoint control register (WPC)	MARBLE
ffe00008	R/W	CPU control register (CCR)	MARBLE
ffe0000c	R/W	BTB control register (BCR)	MARBLE
ffe00010	R/W	BTB test data register (BTD)	MARBLE
ffe00014	R/W	BTB test lcc register (BTL)	MARBLE
ffe00018	R/W	BTB test control register (BTC)	MARBLE
ffe0001c	R/W	BTB test address register (BTA)	MARBLE

11.9 EMI control registers (base at fff00000)

These registers control the operation of the EMI. All registers are 8 bits wide.

Address	Read/Write	Register	Function
fff00020	R/W	DCR	Delay control register
fff00024	RO	DTR0	Delay timing count (low)
fff00028	RO	DTR1	Delay timing count (high)
fff0002c	R/W	ETR	EMI test register
fff00080	R/W	RTR0	Region 0 timing
fff00084	R/W	RAR0	Region 0 architecture
fff00090	R/W	RTR1	Region 1 timing
fff00094	R/W	RAR1	Region 1 architecture
fff000a0	R/W	RTR2	Region 2 timing
fff000a4	R/W	RAR2	Region 2 architecture
fff000b0	R/W	RTR3	Region 3 timing
fff000b4	R/W	RAR3	Region 3 architecture
fff000c0	R/W	RTR4	Region 4 timing
fff000c4	R/W	RAR4	Region 4 architecture
fff000d0	R/W	RTR5	Region 5 timing
fff000d4	R/W	RAR5	Region 5 architecture
fff000e0	R/W	RTR6	Region 6 timing
fff000e4	R/W	RAR6	Region 6 architecture
fff000f0	R/W	RTR7	Region 7 timing
fff000f4	R/W	RAR7	Region 7 architecture

12. The MARBLE bus

The Manchester AsynchRonous Bus for Low Energy is a multi-master on-chip bus for connecting macrocells. Its functionality is broadly similar to previous on-chip macrocell buses such as the OMI PI-Bus and ARM's AMBA. The fundamental difference is that MARBLE is asynchronous, operating without a clock.

MARBLE provides support for bridging between multi-master buses (albeit with slightly restrictive bus-error support), but the AMULET3H chip has no such requirement, and hence the version of MARBLE used in this chip has no defer mechanism for causing transfer retries (i.e. hardware polling). Similarly, the design of MARBLE provides for using tristate-signalling as an alternative to the centrally-gated signalling used in the AMULET3H. The documentation presented here is for the AMULET3H implementation of (reduced functionality) MARBLE. The complete MARBLE specification can be found elsewhere.

12.1 Features

- Support for high-speed transfers of bytes, half-words and words:
  - cycle rate over 50M transfers/sec;
  - sustained throughput of over 200Mbytes/s per initiator;
  - up to 320MBytes/s in total for the whole bus.
- Zero quiescent power due to asynchronous operation.
- Multi-master capability with central arbitration.
- Bandwidth can be apportioned on a per-initiator basis
- Support for atomic transactions
- Test support (see 'The AMULET3H Test Interface Controller (TIC)' on page 60)
- 8/16/32-bit data accesses.
- Processor independent
- Abort response support for bus errors and systems without full MMU support
- Supports sequential address/same page (burst) memory system optimisations

12.2 Architecture

MARBLE uses two channels, both operating witha 4-phase bidirectional push/pull protocol. The channels are:

- the address/command channel; and
- the error-response and read/write data channel.

Each of these channels has separate arbitration and carries a tag (2-bits in AMULET3H) indicating which initiator owns the transfer so that the data cycles can be routed to the correct initiator. This is necessary because (unlike most synchronous buses) every MARBLE transfer is a performed as a split transfer.

Minimal bus occupancy

Every port where a packet is taken from the bus contains a latch to provide local storage for the packet. This is especially important at the target-address port where it allows the target to use the address/command information throughout its device-access whilst not occupying the bus. This allows the bus to be completing its return-to-zero phase and possibly be used for another transfer (see split transfers below).

Split transfers

Cycles on the address/command channel are always started by the initiator and are used to send an

address and other command information (such as the transfer size and direction) to the target. The full version of MARBLE allows for transfers to be postponed through a defer mechanism on the address/command channel (required for bridging between multi-master buses), but the implementation used in the AMULET3H chip does not exploit this feature so every transfer has exactly one address/command channel cycle associated with it.

Cycles on the error-response and read/write data channel are started by the target when it is ready to transmit an error response and perform the appropriate data activity, either accepting write data or supplying read data. The read and write data transfers can be merged onto the same wires in this channel since data will only ever be transferred in one direction during any one cycle. Each transfer has exactly one cycle on this channel. The separate arbitration network for this channel is a consequence of the split-transfer architecture allowing the targets to control when cycles start.

During the period between the completion of the address/command cycle of a transfer and the start of the corresponding error-response/data cycle of that transfer, the bus is free to be used by transfers between other devices thus giving what is known as a split transfer. (A fully featured MARBLE bus also allows other transfers between the same devices, but these are disallowed here - see below). The split-transfer approach is beneficial in allowing, for example, the CPU to be reading data from the external memory interface of the chip (which may take 10s of nanoseconds) without stalling the bus, allowing the DMA controller to continue transfers between on-chip memory and peripherals.

Single-outstanding-issue ordering constraint

A system that makes full use of the benefits of a split transfer architecture would allow an arbitrary number of address/command packets to be issued from the same initiator without any corresponding response/data cycles taking place, thus allowing pipelining of the memory system. This introduces the problems of out-of-order reponse/data cycles occuring, i.e. for two address packets issued in the order A then B, the corresponding response/data cycles may occur in the order A then B, or B then A. Clearly this requires a re-ordering capability if the correct data is going to be delivered in the correct order to the correct device

The AMULET3H MARBLE implementation does not allow this situation to occur by restricting the number of outstanding cycles for any initiator to one, i.e after an initiator has performed an address/command cycle, it cannot perform another one until after it has started to perform the corresponding response/data cycle.

Overlapping (Pipelining) of Transfers

To improve performance MARBLE allows the overlapping of some of its protocol stages. This behaviour is typical of modern high throughput buses, but the asynchronous nature of MARBLE allows greater flexibility and reduced latency compared to the rigid pipelines in clocked buses.

Cycles on the address/command channel may occur at the same time as cycles on the response/data channel. Further, on each of these channels, once a cycle has started, the arbitration for the next cycle on that channel may begin, thus the arbitration for a cycle may be overlapped with the execution of the previous cycle. (The implementations of the MARBLE interface controllers used in the AMULET3H do not allow a device to arbitrate for its next access to a channel whilst still using that channel).

12.3 Signal List

The MARBLE bus signals, including the arbitration and address-select signals used in the AMULET3H chip, are listed in the table below.

12.4 Implementation Specifics

The implementation of MARBLE used in the AMULET3H chip lacks three features of the full-blown MARBLE: the defer mechanism (MDEF signal), support for multiple outstanding commands from the same initiator, and the MP[1:0] privilege code for use by a bus protection unit. Otherwise the functionality is as per the MARBLE specification.

This implementation uses centralised address decoding and distributed data-tag decoding. Centralised OR-gates are used to gather local channel requests and acknowledges from all of the initiators/targets to form the MAR, MAA, MDR and MDA signals of the bus. As per the specification all bus signals are driven at all times with hold-cells used to ensure that tristate lines do not oscillate when not driven by an initiator or target.

Name	Function	Description
MNRES	Active Low Reset	Global bus reset signal
MAarbreqn	Address Arbitration Request n	Signal from initiator to bus arbiter indicating that initiator n requires access to the address channel of the bus
MAarbgntn	Address Arbitration Grant n	Signal from bus arbiter to initiator n indicating that it has been granted access to the address channel for the next cycle
MASm	Address Select m	Signal from the address decoder to target m indicating that it is to respond as the target of the current address cycle
MAR	Address Request	The address channel request line driven by the initiator
MAA	Address Acknowl-edge	The address channel acknowledge line driven by the target
MAO	Address Operation	The address operation indicating if the transfer is a read or a write
MAT	Address Tag	The address tag indicating which initiator the transfer originated from
MA[31:0]	Address	The 32-bit address, driven by the initiator
MSIZE[1:0]	Transfer Size	The size of the data packet to be transferred, which may be byte, half-word or word
MS[2:0]	Sequential	Sequentiality indicator
ML	Lock	Lock signal indicating that the current and next transfers from this ini-tiator must be performed as an atomic sequence.
MDarbreqn	Data Arbitration Request n	Signal from target to the bus arbiter indicating that target n requires access to the address channel of the bus
MDarbgntn	Data Arbitration Grant n	Signal from bus arbiter to target n indicating that it has been granted access to the data channel for the next cycle
MDR	Data Request	The data bundle request line driven by the sender
MDA	Data Acknowledge	The data bundle acknowledge line driven by the receiver
MDO	Data Operation	The data operation indicating if the data cycle is a read or a write
MDT	Data Tag	The data tag indicating which initiator the data cycle is destined for
MD[31:0]	Data Bus	The bi-directional data bus, driven by the sender
MABT	Abort	Abort indicator driven by the target

12.5 Initiator Bridge

The initiator bridge allows connection of an initiator device to the MARBLE bus. Two bundles (both four-phase broad protocol unidirectional bundled-data) are used for the connection of the initiator bridge to its associated device. These are:

- The address/command bundle - AonR{Aon[31:0],O,S[2:0],L,size[1:0],Don[31:0]}AonA
- The response/read-data bundle - RoffR{ABT,Doff[31:0]}RoffA

Details of the signals forming each of these bundles are given below. For every transfer there will be one transaction on each bundle, but only one of the data-fields, as indicated by the operation bit (O) of the command will contain valid information. An additional input to the initiator bridge is the Tag (T) that is an identifier unique to each initiator, which should be hard-wired when a system is constructed. Many of the signals detailed below closely resemble their MARBLE counterparts.

- The Address/Command-On (Aon) Bundle

The Aon bundle, the signals for which are shown below, is used to supply address/command packets from the initiator device to the initiator bridge. The channel uses a 4-phase broad protocol and the encoding of the bundled signals is the same as their counterparts on the MARBLE bus.

Signal	Description	Driver
AonR	Aon bundle request line	Device
AonA	Aon bundle acknowledge line	Bridge
Aon[31:0]	The 32-bit address from the bus	Device
O	The operation, 0=write, 1=read	Device
S[2:0]	Sequential	Device
L	Lock, 0=last_cycle, 1=more_to_follow	Device
size[1:0]	The 2-bit size field	Device
Don[31:0]	The 32-bit write-data (only valid for a write transfer)	Device

The lock signal is used to indicate when a transfer is a (non-final) part of an atomic activity. This is important since the bus must not allow the atomic sequence to be interrupted. The size field is used to indicate how many bytes should be transferred in the data cycle, the encoding is shown in the table below.

SIZE		Transfer Size
[1]	[0]	
0	0	BYTE - 8 bit transfer
0	1	HALFWORD - 16bit transfer
1	0	WORD - 32 bit transfer
1	1	Reserved

The sequential field allows hints to given about the relationship between one address/command packet and the next packet from the same initiator. Such information is useful with certain types of memory, e.g. DRAM which can perform sequential accesses much faster than random accesses.

Bit S[0] indicates if the current address is sequential to the previous address. Bits S[2:1] indicate the relationship between the current address and the next address:

S		Meaning
[2]	[1]	
0	0	No predictive relationship
0	1	Next address may be sequential and in the same 2^n page
1	0	Next address will be sequential and in the same 2^n page
1	1	not used

The resulting codes that can be transferred, and typical uses are shown in the table below.

S			Meaning	Typical Use
[2]	[1]	[0]		
0	0	0	THIS address has no special relationship to any other address.	Non-sequential memory access Branch Target Instruction Fetch
0	0	1	THIS address is sequential to the previous address	Instruction fetch
0	1	0	The NEXT address MAY be within the same 2^n page	Cache Line Fetch (wrap-around) DMA Transfer[a]
1	0	0	The NEXT address will be sequential to the current address	DMA transfer, ARM LSM instructions, Cache Line Fetch

a. The AMULET3H DMA controller does not use sequential bus modes of any sort as it does not use burst transfer modes

- The Response/Read-data Off (Roff) Bundle

The Roff bundle, the signals for which are shown below, is used to provide an abort response to the initiator device. There will be one cycle on this channel for each cycle on the Aon channel. The channel uses a 4-phase broad protocol and the encoding of the ABT signals is the same as that of the MABT signal of MARBLE.

Signal	Description	Driver
RoffR	Roff bundle request line	Bridge
RoffA	Roff bundle acknowledge line	Device
ABT	The abort response	Bridge
Doff[31:0]	The 32-bit read-data	Bridge

Channel Timing Relationship

A transfer begins when the device supplies an address/command packet on the Aon bundle. Following AonR+, a response will be provided, RoffR+ indicating validity of the abort bit (Roff) and any read-data.

Channel Timing Relationship Summary: In summary, the dependencies imposed by the initiator bridge on this interface (in addition to the usual channel signalling protocol) are:

- RoffR+ follows AonR+
- AonA+ of transfer n will be at about the same time as, or later than RoffR+ of transfer n-1.

12.6 Target Bridge

The target bridge allows connection of a target device to the bus using two four-phase broad protocol unidirectional channels:

- The address/command-off bundle - AoffR{Aoff[31:0],O,T,S[2:0],L,size[1:0],Doff[31:0]}AoffA
- The response/read-data-on bundle - RonR{ABT,Don[31:0]}RonA

For every transfer there will be one transaction on each of these bundles. However, a transfer may be deferred using the Anack signal (instead of AoffA). The additional Ardy signal provides an indication to the target device that the command is available except for any write data (which is not significant for a read transfer).

- The Address/command-Off (Aoff) Bundle

  The Aoff bundle, the signals for which are shown below, is used to feed address/command packets from the bridge to the target device. The channel uses a 4-phase broad protocol and the encoding of the bundled signals is the same as their counterparts on the MARBLE bus and in the initiator interface Aon bundle.

Signal	Description	Driver
AoffR	Aoff bundle request line	Bridge
AoffA	Aoff bundle acknowledge line	Device
Aoff[31:0]	The 32-bit address from the bus	Bridge
T[1:0]	The initiator tag (2-bits in AMULET3H)	Bridge
O	The operation, 0=write, 1=read	Bridge
S[2:0]	Sequential	Bridge
L	Lock	Bridge
size[1:0]	The 2-bit size field	Bridge
New	Indicates when the initiator is different to the previous initiator that addressed this device	Bridge
Doff[31:0]	The 32-bit write-data	Bridge

- The Response/read-data On (Ron) Bundle

  The Ron bundle, the signals for which are shown below, is used by the target device to provide an abort response. There will be one cycle on this channel for each cycle on the Aoff channel. The channel uses a 4-phase broad protocol and the encoding of the ABT signals is the same as that of the MABT signal of MARBLE.

Signal	Description	Driver
RonR	Ron bundle request line	Device
RonA	Ron bundle acknowledge line	Bridge
ABT	The abort response	Device
Don[31:0]	The 32-bit read-data	Device

Channel Timing Relationship

A transfer begins with an address/command packet being delivered on the Aoff bundle. After some processing delay, the target device will then supply a respone packet for return across MARBLE.

Channel Timing Relationship Summary: In summary, the dependencies imposed by the target bridge on this interface (in addition to the usual channel signalling protocol) are:

- RonR+ follows AoffR+
- RonA- follows AoffA+

13. The MARBLE/SOCB Bridge (MSB)

The MARBLE to Synchronous On-Chip Bus Bridge (MSB) is a MARBLE target device which handles the bus handshake and control signal retiming on behalf of the Synchronous On-Chip Bus (SOCB). The bridge provides latching of all address, data and control signals. The MSB controls transfers on the SOCB data bus, providing the appropriate read/write signals. Table 1 lists the signals in the SOCB/MSB interface.

13.1 Features

- supports a conventional, clocked, simple peripheral bus
- busy signalling to stall a peripheral access

13.2 Interface signals

Table 1: MSB-SOCB interface signals

Signal	Type	Function
Nwait	I (active low)	informs bridge that the shared RAM is busy
NIOCS	O (active low)	SOCB peripheral access signal
NSBrd	O (active low)	read from SOCB
NSBwr	O (active low)	write to SOCB
SA[19:0]	O	SOCB address lines
SD[31:0]	IO	SOCB data lines
SIZE[1:0]	O	SOCB size of data transfer
CLK	I	SOCB clock

13.3 The SOCB

The SOCB allows access to peripheral devices in the synchronous subsystem. Transfers across the SOCB are controlled using the SOCB clock. The SOCB supports 8-, 16- and 32-bit data transfers. The address decoding is performed by a decoder unit on the SOCB. Peripherals on the SOCB can only drive the bus in response to a transfer request from an initiator on MARBLE.

13.4 Operation of Nwait

The signal Nwait allows devices on the SOCB to extend their access cycles beyond the default minimum timing. If Nwait is high then the transfer may complete in the current cycle while if Nwait is low then further bus cycles are required. Nwait should be valid at the rising edge of the SOCB clock.

Figure 6 illustrates the generation of a single wait state. To produce this single wait state Nwait can lowered at any time in the period starting before the request from MARBLE (point (a) Figure 6) to just before the SOCB clock rises (point (b) Figure 6). Nwait must be low for the positive edge of the SOCB clock to convert a transfer cycle to a wait cycle. To generate a further wait cycle Nwait is held low for the positive edge of the SOCB clock in the following cycle. To commence a transfer cycle Nwait must rise before the positive SOCB clock edge of the transfer cycle (point (c) Figure 6).

Independently of when Nwait arrives [point (a) to point (b)] the action of the MSB will be the same. During write cycles the MSB ensures that data, NSBwr, NIOCS and address lines remain valid

throughout any wait cycles and the transfer cycle. During read cycles the peripheral must drive the data bus such that it is valid by the end of the last SOCB clock HIGH phase of the transfer.

13.5 Synchronisation and bus timing

Initiators on MARBLE indicate to the MSB that they wish to access peripherals on the SOCB by making a request. To allow access to the SOCB the MSB must first synchronise with the SOCB clock. As SOCB transfers commence on the positive edge of the SOCB CLK, and the MSB synchronises with the negative edge of SOCB CLK, the process of synchronisation takes a worst case time of 1.5 SOCB clock cycles and a best case time of 0.5 SOCB clock cycles. Figure 6 shows the arrival of two requests from MARBLE [labelled (1) and (2)]leading to the worst and best case synchronisation times, respectively. The average case synchronisation time is 1 cycle. Buffering is provided within the MSB to allow pipelining of the synchronisation process and SOCB access.

The synchronisation process completes in time to allow the SOCB transfer to commence on the first positive edge of the SOCB clock after the synchronisation edge. At this time the MSB will drive the address information and data (if the MARBLE initiator is writing to the peripheral) onto the SOCB. An additional signal NIOCS is also supplied to inform the SOCB address decoder that the target lies within its address range. Half-cycle setup time is then provided (to allow the SOCB to decode the address information) before read or write strobes are applied.

If the MARBLE initiator wishes to read from an SOCB peripheral, then the information must be stable at the rising edge of NSBrd. In order to avoid drive clash, the ‘turn-off’ time (labelled ‘t’ in Figure 6) must be less than the hold time.

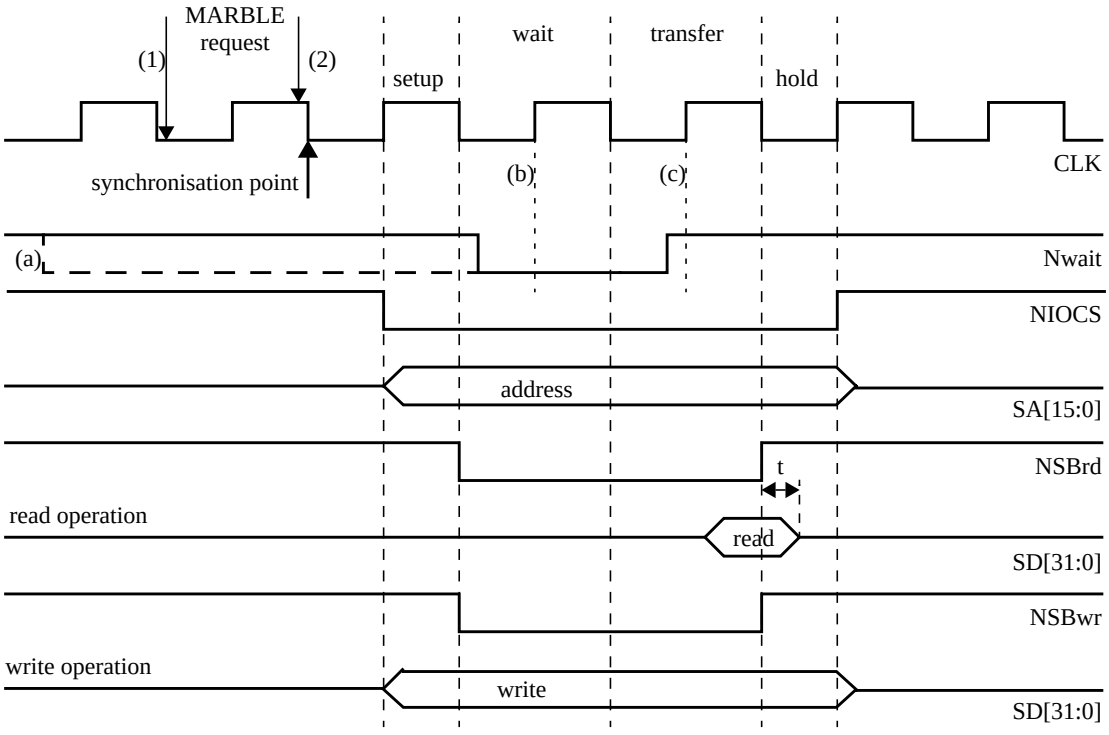


Figure 6: Operation of the MSB/SOCB interface

14. The AMULET3H Test Interface Controller (TIC)

The AMULET3H test interface controller (TIC) provides an external interface to a MARBLE initiator. This allows external access to all MARBLE targets for test or debug purposes. The TIC interface is designed to suit conventional VLSI production test equipment and it therefore uses a clocked protocol.

14.1 Features

- direct access to individual on-chip macrocells via external memory interface and MARBLE.
- auto-incrementing address generation to increase the efficiency of testing of memory blocks.
- support for modular, macrocell-based VLSI production test strategy.

14.2 Test interface signals

The test port uses the following signals:

Signal	In/Out	Description
TCLK	I	test clock
TOPC[2:0]	I	test opcode
TABT	O	MARBLE abort
TBUS[31:0]	I/O	test data bus

In order to use the TIC the chip must be put into test mode, which is achieved by taking the ‘Ntest’ external pin low. (The DRACO chip also requires other ‘mode’ pins to be set correctly.)

14.3 External pin mapping

The external memory interface provides the test mode interface to the internal bus. The ‘Ntest’ signal enables test mode when it is low. When configured for test mode the address and data buses provide the test mode interface as follows:

Pins	Type	Normal function	Test mode function
d[15:0]	IOZ	data bus	TBUS[15:0]
a[15:0]	IOZ	address bus	TBUS[31:16]
a[16]	I	address bus	TCLK
a[19:17]	I	address bus	TOPC[2:0]
rNw	O	read/not write	TABT
Ntest	I	(High)	(Low)

In the above table the ‘Type’ column describes the use of the pin in test mode; this is different in some cases from the ‘Type’ in normal operating mode.

In test mode the external memory interface becomes a MARBLE initiator, enabling external test hardware to access all MARBLE target registers (including SOCB peripherals via the MARBLE-SOCB bridge).

14.4 Test interface operation

The test interface is used as follows:

First, a 3-bit opcode is placed on the TOPC[2:0] pins while the clock pin, TCLK, is held low. Then TCLK is set first high (phase 1) and then low again (phase 2). The state of the TOPC[2:0] pins is latched when TCLK rises. There are six valid operations (as defined by TOPC[2:0]) which may be performed and two encodings which are currently reserved. The 32-bit TBUS is used to read or write data to/from the TIC. The TABT pin reflects the state of the MARBLE abort bit following a MARBLE read or write cycle. The value on TABT persists until another MARBLE cycle is performed.

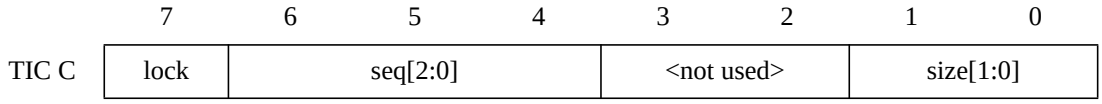
14.5 TIC registers

Within the TIC there are two registers which may be written via the interface:

- The A register provides the address (32 bits) which is used by the TIC MARBLE initiator. The bottom 16 bits of A increment automatically after certain operations.
- The C register provides 8 bits of control information to the initiator to determine what type of cycle will be performed.

The C register maps directly to MARBLE control signals and enables the TIC to generate any form of MARBLE transfer.

The mapping of bits in the C register to MARBLE control bits is as follows: .



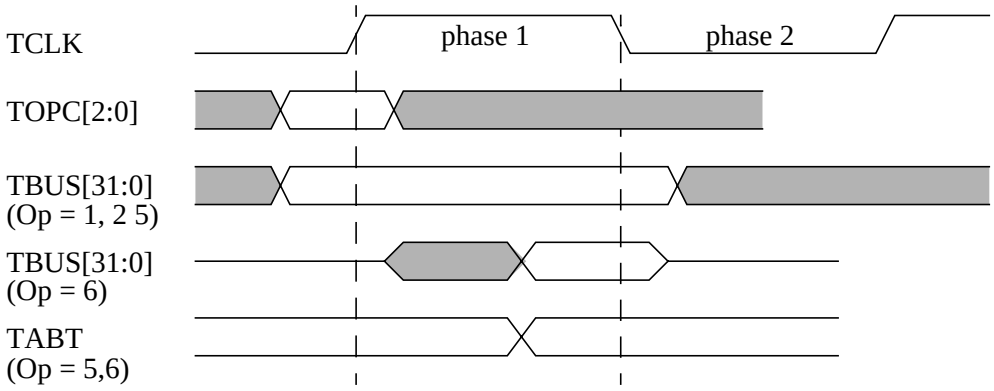
14.6 TIC operations

The operations which the TIC can perform are listed below:

TOPC[2:0]	Op	Function	TBUS[31:0]
000	NOP	No operation	(input - unused)
001	WRA	Write A register	input A value
010	WRC	Write C register	input C value
011	-	(reserved)	(undefined)
100	INC	Increment A register	(input - unused)
101	MWR	MARBLE write (& inc A)	input - MARBLE write data
110	MRD	MARBLE read (& inc A)	output - MARBLE read data
111	-	(reserved)	(undefined)

Those operations which write data must ensure that the data is set up on TBUS[31:0] before TCLK rises and held until after TCLK falls. In a read operation the TBUS[31:0] signals are driven as outputs while TCLK is high (and TOPC[2:0] = 6) and revert to inputs when TCLK is low.

14.7 TIC timing diagram



After the MRD, MRW and INC operations the lower 16 bits of the A register are incremented by the amount specified in the size field of the C register. The upper 16 bits are unchanged by the increment. All operations take place in phase 1 of the test cycle and the A register is incremented in phase 2.

The A and C registers are undefined after chip reset and must both be initialised before TIC MARBLE reads and writes are performed.

14.8 Notes

- if the TIC is enabled (‘Ntest’ is low) while the processor, DMA controller or TIC is accessing the EMI via MARBLE, the normal bus arbitration mechanisms will apply and the processor or DMA controller will have normal access to the EMI. The only difference, as far as the processor or DMA access is concerned, is that those external signals reassigned to TIC functions will not respond in any way.

15. Test facilities and procedures

Asynchronous components present a new challenge for VLSI production testing. The AMULET3H asynchronous subsystem has been designed with production test in mind, and this section discusses the issues involved in obtaining good test coverage of the asynchronous components.

The following approaches are used to give efficient test coverage of the AMULET3H asynchronous subsystem:

- The external memory interface can be reconfigured to act as a test interface controller (TIC). (See 'The AMULET3H Test Interface Controller (TIC)' on page 60) This enables standard external test equipment to 'see' on-chip macrocells via the MARBLE bus and to access their registers for test purposes.
- The TIC can be used to load test programs into the on-chip RAM. These programs can then be run by the AMULET3H core (at full speed) to test functions not covered by the test program in the on-chip ROM.
- On-chip registers have read/write functionality even when they could be write-only for operational purposes. This makes the registers themselves easy to test.
- Difficult-to-test structures, such as the CAM in the branch target buffer, have special test access ports accessible via MARBLE. Thus the BTB can be tested by an external tester, or even by an on-chip test program provided that the BTB function is disabled while the test is running.

15.1 Testing the AMULET3 core

The AMULET3 core is the most complex part of the system to test. The approach will be similar to that used on clocked ARMs, which is to generate self-testing code in a structured way from assembly source code.

The following test control signals are available via the CCR (See 'CPU control register - CCR - at ffe00008' on page 14):

- TstInt - CCR[7] - interrupt test control - when 1 connects the external boot control pins to the core interrupt inputs. bt[0] is connected to $\overline{irq}$ and bt[1] to $\overline{fiq}$. Other interrupt sources are disconnected.
- TstNrst - CCR[5] - when 0, and the system is in test mode, forces a hard reset into the CPU core logic. This may be used by an external tester to prevent the CPU from running while, for example, the tester loads a test program into RAM. Once the program is loaded the tester may release the processor from reset by writing a 1 into this bit. There is no guarantee that the CPU interface state will be stable if reset is applied once the processor has started running, so this feature should only be used to release the processor from reset in test mode and not to return it into reset.

The key AMULET3 areas to test are:

- The register file. Firstly, check all registers are unique by writing different values into each register (covering all modes) and then reading these values back. Then check all the register bits by writing 0x55555555 and 0xAAAAAAAA into each register and reading the values back. Finally, each of the 3 register read ports must be checked by using instructions which access each register via each port.
- The PC. Checking all the PC bits requires some cunning code, as testing the MSBs as 1s can only be done by branching up to high memory (where there is no memory!). This may require cooperation between the AMULET3 core and an external tester. Again, a 1 and a 0 must be read from every bit of the PC via each of the 3 register bank read ports.
- The CPSR and SPSRs. The state bits can be tested using MSR/MRS instructions. The CPSR must then have flag setting from the ALU and shifter tested, and a conditional instruction sequence written to test every conditional execution combination. (That is, BEQ and BNE each

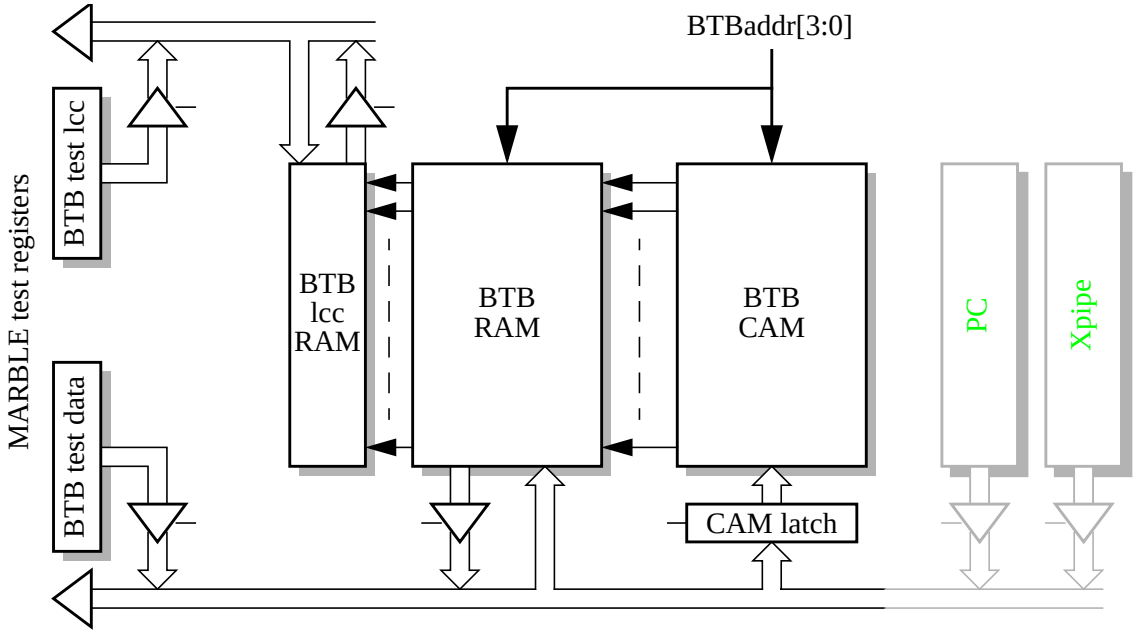
with Z set and Z clear, etc.)

- The instruction decoder. All relevant ARM instruction formats must be executed, looking into corners (small & large immediate values, use of PC, and so on).
- The Thumb decoder. Likewise, a thorough exploration of the Thumb instruction space must be undertaken.
- Interrupts. Some tests must handle interrupts and check for correct SPSR values, mode switching, and so on.
- Aborts. Prefetch and data aborts must likewise be explored.
- The multiplier. Several multiplications must be run, exploring corners.
- The shifter. All shift amounts must be tested, including left and right shifts. Carry out cases must be explored for register shift amounts greater than 32.
- The ALU. Check all single-bit carry-propagate conditions, plus a number of other additions and subtractions (including 32-bit carry propagate cases). Check all ALU operations.
- The branch adder. This is hard to test for the same reasons as the PC itself.
- The reorder buffer. It's not clear how best to test the reorder buffer, but it is likely to have taken a fair beating in the above, and test escapes are considered unlikely.

15.2 Testing the BTB

The CAM-RAM structure of the BTB is particularly difficult to test in normal operation, so additional test features are provided to facilitate test. The test structure is based around two bidirectional test buses as illustrated below.

BTB test organization



- the 8-bit test lcc bus is used only to read and write data from and to the link and condition code (lcc) RAM in the BTB;
- the 32-bit test data bus is used to read and write the main BTB target address RAM, the BTB CAM input latch, and other 32-bit components.

BTB test registers

Address	R/W	Function
ffe00010	R/W	BTB test data register - BTD
ffe00014	R/W	BTB test lcc register - BTL
ffe00018	R/W	BTB test control register - BTC
ffe0001c	R/W	BTB test address register - BTA

The BTB CAM test input latch uses bits [31:2] of the BTB test data register as inputs, latched by pulsing BTBclt high, to send as inputs to the BTB CAM. The value is stored inverted.

The BTB test data register is used to hold values to be written to the BTB RAM or to read values from the BTB RAM. Bits [31:1] are valid.

The BTB test lcc register is used to read and write the BTB RAM link and condition code bits. These operate in parallel with and as extensions of the BTB test data register. Bits [4:0] are valid. In addition, in read mode bit [5] returns the BTB CAM hit signal and bits [7:6] return the bottom two bits of the address from the BTB address generator (pseudo-random or sequential counter).

BTB test control register - BTC - at ffe00018

	7	6	5	4	3	2	1	0
BTC	BTBtst	BTBinv	BTBtrw	BTBclt	BTBtrq	BTBsrc[2:0]		
reset to:	0	0	0	0	0	000		

- BTBtst - BTB test - when 1 enables various BTB test functions as described below. BTBen (in the CCR) is forced to 0 whenever BTBtst is enabled.
- BTBinv - BTB invalidate - when 1 clears all the BTB locations. BTBinv is only effective when BTBen is low.
- BTBtrw - BTB test read/write - when 1, configures the BTB for reading in test mode; when 0, for writing.
- BTBclt - BTB CAM latch enable - when 1, copies the value in the BTB test data register into the CAM test input latch. The BTB test data register must be selected as the BTB test data bus source (BTBsrc[2:0] = 0).
- BTBtrq - BTB test request - a low-to-high transition on this signal will cause a BTB read or write cycle to take place.
- BTBsrc[2:0] - BTB test data bus source - selects which source drives the BTB test data bus.

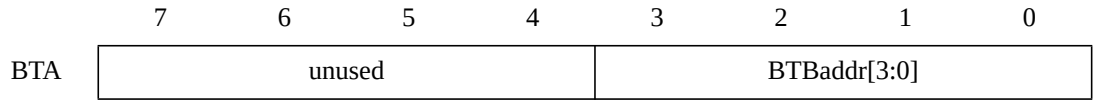
The following four BTBsrc[2:0] encodings are defined (other encodings should not be used):

BTBsrc[2:0]			BTB test data source	BTB test lcc source
0	0	0	test data register	test lcc register
0	0	1	BTB RAM	BTB lcc RAM
0	1	0	PC	-
0	1	1	Xpipe	-

The last two codes enable the PC and Xpipe outputs to be read directly for test purposes, improving

access to functions which are otherwise hard test.

BTB test address register - BTA - at ffe0001c



- BTBaddr[3:0] - BTB address - used to address one of the 16 CAM-RAM pairs for writing into the BTB; replaces the pseudo-random number generator as the address source when BTBtst is one.

BTB test register operation

To set up the BTB for test access:

- disable the BTB for normal operation (BTBen = 0 in the BCR)
- set the BTB into test mode (BTBtst=1 in the BTC register)

In addition, interrupts are ignored while the BTB is in test mode, and indirect PC loads must be avoided (as they will cause deadlock).

To write a value into a BTB entry:

- set the the entry address in BTBaddr[3:0] in the BTA register
- set BTBsrc[2:0]=0, BTBtrq=0, BTBtrw=0 in the BTC register
- write the required CAM value into the BTB test data register
- write the BTB CAM latch by pulsing BTBclt high and then low
- write the required RAM value into the BTB test data register
- write the required lcc value into the BTB test lcc register
- pulse BTBtrq high and then low to perform the write

To read a value from the BTB RAM and lcc RAM, the CAM must first have been set up to match a unique value for RAM addressing purposes (since RAM read happens only as a result of a CAM match). Then:

- set the the entry address in BTBaddr[3:0] in the BTA register
- set BTBsrc[2:0]=0, BTBtrq=0, BTBtrw=1 in the BTC register
- write the required CAM value into the BTB test data register
- write the BTB CAM latch by pulsing BTBclt high and then low
- set BTBsrc[2:0]=1 to enable the RAM output onto the BTB test data bus
- take BTBtrq high to perform the read
- read the BTB RAM output from the BTB test data register; bit[0] returns the CAM hit signal
- read the BTB lcc RAM output from the BTB lest lcc register
- take BTBtrq low to precharge the BTB for the next access

Testing the CAM itself can be performed using an efficient parallel algorithm described in ‘BTB test operation’ on page 66, after which the CAM can be configured to address each RAM location uniquely and the RAM tested using the CAM as a simple addressing mechanism.

BTB test operation

The BTB CAM may be tested as follows:

1. The BTB should be disabled (BTBen=0), set into test mode (BTBtst=1) and flushed (BTBinv

pulsed high then low) in preparation for testing.
2. Now write 0xaaaaaaa8 into each of the 16 BTB CAM locations (the value written to the RAM is irrelevant for this test).
3. Set the BTB to test read mode (BTBtrw=1) and pulse BTBtrq.
4. Read the BTB test lcc register: bit[5] should be 1, indicating at least one hit.
5. Write 0xaaaaaaac into the BTB CAM data register.
6. Read the BTB RAM test lcc register: bit[5] should be 0, indicating that every CAM location missed.
7. Repeat with 0xaaaaaaa0, 0xaaaaaab8, ..., inverting one bit of the original value each time.
8. Repeat the above from stage 2 using inverted values (0x55555554).

The BTB RAM may be tested as follows:

1. The BTB should be disabled (BTBen=0), set into test mode (BTBtst=1) and flushed (BTBinv pulsed high then low) in preparation for testing.
2. Write a unique value into a CAM location, together with test values into the corresponding address and condition code RAM locations.
3. Repeat for the other 15 RAM-CAM slots, using a different CAM value for each slot (such as the slot address, 0-15).
4. Set the BTB into test read mode.
5. Write one of the unique CAM values into the BTB CAM test latch.
6. Read the BTB RAM output - the written data should be recovered (and bit[5] of BTL should be 1, indicating a CAM ‘hit’).
7. Repeat the read for the other 15 slots.
8. Write different (inverted?) data into each of the RAM slots, and read out again.

The BTB address generator may be tested as follows:

1. The BTB should be disabled (BTBen=0), set into test mode (BTBtst=1) and flushed (BTBinv pulsed high then low) in preparation for testing. Invalidation also resets the address generator.
2. Check the address value by reading BTL[7:6].
3. Perform a write to the BTB to cause the address generator to update.
4. Repeat steps 2-3 16 times.
5. Perform this test in both pseudo-random and sequential counting modes.

Note that in test mode the BTB address generator is ignored as the BTB address is supplied directly from the BTA register. However, it still generates addresses which are updated on every BTB write operation.

There should be at least some testing of the BTB in functional mode, though observability is an issue here as the only software-detectable effect is an increase in performance (although we could use the watchpoint hardware to check that an instruction in the branch shadow hasn’t been fetched).

15.3 Testing the on-chip debug logic

The debug scan chains (data and mask) can be tested by shifting patterns through and observing them emerge through the serial data out bits (see section 5.2 on page 15).

The comparison logic will require each scan chain bit to be tested at 0 and 1, controlling the matched value to give hit and miss for each case. Using the mask to test bits individually would also test the mask logic thoroughly.

Checking all breakpoint bits will require care, for the same reason that checking the PC is tricky.

15.4 Testing the 8 Kbyte dual-port RAM

The RAM will be tested using a marching test pattern following the ‘Static RAM Testing Application Note’ from VLSI Technology, Inc., dated May 12 1997. The test will run from RAM, so the 8 Kbyte RAM block will be tested in two overlapping address ranges. The first range will be tested, then the program will self-relocate to allow the second range (which includes the area of memory occupied by the test program during the first phase) to be tested.

The above tests will cover the data port to the RAM. The instruction port should also be tested effectively by running this and other test programs from the RAM.

15.5 Testing the 16 Kbyte ROM

If the ROM contains appropriate code, the processor can be used to compute a CRC code for the ROM which will check that every location reads correctly. Alternatively, an external tester could check the ROM contents via the TIC, or load a program into RAM to do this.

A write to the ROM should abort - this should be checked.

15.6 Testing the DMA controller

The simplest way to test the functionality of the DMA controller is to program a number of memory to memory transfers across MARBLE. In this way the state transitions and data register functionality for a single channel can be tested. The majority of the user visible registers are also made readable and writable for the purposes of testing and no copies are kept (except by the transfer engine during a transfer) of controller state. A number of read only registers have individual test regimes:

CHANSTATUS: individual bits of the CHANSTATUS word can be set by completing transfers for each channel. It is recommended that a full test scheme perform at least one transfer for each channel which sets CHANSTATUS by being the terminal transfer of a run. CHANSTATUS bits can be reset by accessing the CTRL register for the matching channel from MARBLE.

IRQREQ/FIQREQ: these do not exist as registers within the controller and are computed from CHANSTATUS and the IRQMASK/FIQMASK registers on request.

GENCTRL register TFRCHAN field: this field contain transient state which is kept by the register bank regarding a transfer currently in progress (or the last completed transfer channel where the controller is idle). This state is used by the register bank to perform the actions required when register values are updated at the end of a transfer. To test this field in both the idle and the transfer-in-progress states, a transfer run should be programmed using a DMA controller channel in memory to memory mode without incrementing the source address (recommend: CTRL.USEDRQ=0, CTRL.SIZE=2, CTRL.SRCINC=0, CTRL.DSTINC=1, COUNT=-3), transferring from the GENCTRL register into SRC, DST, and COUNT registers of one of the DMA controller’s long registers. The GENCTRL.TFRCHAN values left in those registers (which can be read by the CPU in the conventional way) should match the channel used to perform the transfer.

Making use of the DMA controller’s own data registers as targets for these transfers may reduce the number of memory accesses required to test these data registers but more importantly allows the address space for the transfers to be kept within the range of all 32 channels by setting the BASE address to the base address of the DMA controller (ffc0xxxx).

The **SETREQ** register in the DMA controller (See ‘SETREQ’ on page 31) is provided to enable DMA requests to be synthesized by test software.

15.7 Testing the external memory interface (EMI)

The EMI control register are all readable and writeable apart from the delay timing registers, DTR0 and DTR1. The reference delay chain will have to be checked at every tap, and the prescaler checked at every setting. The necessary test features have been incorporated in the ETR.

EMI test register - ETR - at fff0002c

The timing reference test register (ETR) has the following format:

	7	6	5	4	3	2	1	0
ETR	Head	Tail	ETtreq	-	-	ETtack	EMItst	CtrTst
reset to:	0	0	1			0	0	0

- CtrTst (D0) splits the 16-bit calibration counter into two halves, each clocked from the same source. This significantly reduces the number of test cycles required to test the counters. Following a counter reset, DTR0 and DTR1 should count up together when this bit is set.
- EMItst (D1) causes the reference timing delay loop to be broken, so the cycle will only complete when an external Tack is supplied (on bt[0] or ETtack). This synchronises off-chip accesses to the tester.
- ETtack (D2) may be used to supply the external memory timing acknowledge when EMItst (D1) is high. Note that this is a two-phase signal, so its state must be observed and inverted to generate an acknowledge.
- ETtreq (D5) is an inverted copy of the two-phase timing request, treq. It is 1 at reset.
- Tail (D6) is used to check that the timing reference chain bar-code generator will clear to zero. Setting L, M and R to zero should give a one output on Tail, and any delay less than maximum should give a zero on Tail.
- Head (D7) is the opposite end of the timing reference chain bar-code generator, and should be one for all delays above the minimum. Setting L,M and R to one should give the minimum delay and should result in a zero output from Head.

D3-D0 are read/write; D7-D4 are read only.

Checking that L, M and R = 0 gives Head, Tail = 1 and L, M and R = 1 gives Head, Tail = 0 is sufficient to detect any gate stuck-at fault in the bar-code generator.

15.8 Testing the MARBLE bus

The MARBLE bus will receive extensive exercising in the course of testing the components connected to it. This is probably sufficient.

15.9 Testing the MARBLE/SOCB bridge

As the synchronous peripherals are tested via the SOCB, it is likely that all of the functionality of the MARBLE/SOCB bridge will be tested in the course of the synchronous peripheral subsystem tests.

15.10 Testing the AEDL

The AEDL logic is fairly simple and testing should be straightforward. The AEDL_rdy[2:0] and other interface signals are controllable from the asynchronous subsystem via the AEDL test control register (ATCR) so that the AEDL logic can be tested extensively from the asynchronous subsystem.

The external pin input bt[0] can also be used to generate AEDL_rdy inputs to give the tester direct control of the timing.

AEDL test control register - ATCR - at ff800000

	7	6	5	4	3	2	1	0
ATCR	Xtrdy	Tdata	Trdy[2:0]			Tsync	Reset	Test

The AEDL has a test mode controlled by writing to any address in the AEDL address range. Before the AEDL is used in either normal or test mode a write to this control register must be performed to initialise the controlling logic (which is not configured by chip reset).

- Test is low for normal operation, high for test mode.
- Reset is high to reset the AEDL logic, low for normal operation.
- Tsync performs the role in test mode that AEDL_sync performs in normal operation.
- Trdy[2:0] perform the role in test mode that AEDL_rdy[2:0] perform in normal operation.
- Tdata supplies the data value that is read on D[31:12, 8:0] in test mode, allowing 1s and 0s to be passed through the interface for test purposes.
- In test mode AEDL_ack[2:0] are read on D[11:9].
- Xtrdy, when high, connects bt[0] to all three of AEDL_rdy[2:0] (without disconnecting the other source: Trdy[2:0] when Test is high; AEDL_rdy when Test is low). This enables an external tester to generate the ‘rdy’ event. When Xtrdy is low bt[0] is disconnected from the AEDL logic.

This register is included to enable the AEDL logic to be tested to a considerable extent from within the asynchronous subsystem.

15.11 Testing the synchronous peripheral subsystem

The TIC gives an external tester full access to the SOCB via MARBLE. Therefore the asynchronous subsystem is ‘transparent’ as far as testing the synchronous subsystem is concerned. Test programs can be run from on-chip memory (RAM or ROM) where this is a useful way to test components on the SOCB.

The following test control signal is available via the CCR (See ‘CPU control register - CCR - at ffe00008’ on page 14):

- TstClk - CCR[6] - clock test control - when 1 connects the external boot control pins bt[1] and bt[0] to the SOCB bridge clock input (*CLK*) and *Nwait* signals respecively. The normal clock source is disconnected.