Firefly MF1 Core
Design Manual

Part Number:	Firefly MF1 Core
Revision Number:	**3.4**
Issue Date:	**November 2003**

ZARLINK™
SEMICONDUCTOR

Manual Revision History

Version	Revision	Date	Update Summary
V1R1	001	September 1998	First draft, for internal review only.
V1R2	002	March 1999	Second draft edition.
V1R3	003	June 1999	First printed edition.
V2R1	001	October 1999	2 new appendices added: System Decision Guide and Examples of Typical System Configurations; tables in section 2 updated; section on IDDQ testing added to section 2; ARM trademarks updated for new print run.
V2R2	002	December 1999	Added in two locations, a note regarding the use of a Bus Hold cell.
V2R3	003	June 2001	Change of Corporate identity from Mitel Semiconductor to Zarlink Semiconductor.
V3R4	004	November 2003	Updated to new corporate template.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

Document Conventions

The following terms, which appear in the manual, are defined here:

- Embedded Microcontroller ASIC: An ASIC which contains an Embedded Microcontroller cell (e.g., Firefly MF1)
- on-core: Part of the Firefly MF1 Embedded Microcontroller core
- off-core: NOT part of the Firefly MF1 Embedded Microcontroller core.
- on-chip: Inside the Embedded Microcontroller ASIC but outside the Firefly MF1 core.
- off-chip: Not part of the overall Embedded Microcontroller ASIC
- external device: Device, such as a memory or customer logic, which is ’on-chip’ or ’off-chip’.
- port: Points on the Firefly core where connections are made from Firefly waveforms to off-core circuitry (which can be on-chip or off-chip (via a driver cell and a pin)).
- pin: A pin of the ASIC.
- signal: A signal sourced from a Firefly port and used to create another signal in the off-core system.
- system bus: A bus from certain Firefly ports for the connection of on-chip & off-chip logic
- external master: A Master device sited on the system bus.
- low *or* clear: Refers to a logical condition 0 of a signal or bit-field.
- high *or* set: Refers to a logical condition 1 of a signal or bit-field.

Numbers prefixed with ’0x’ are hexadecimal.

Numbers prefixed with ’0y’ are binary.

Register, bit-field and signal names are all in **bold** type.

External signal names begin with a capital letter and all internal signal names are in lower case. Both are prefixed with an "n" if active low.

Register field bit positions are represented within square brackets, thus: [n]

"Reserved": When associated with a register field, the location should not be written to or read from. When used in a bit-field among other referenced fields, the default value must be maintained during write operations.

1.0 Introduction

1.1 Firefly MF1 Embedded MicroController Core Overview

This book describes the Firefly MF1 Embedded MicroController Core, which is a Thumb® based derivative of the Butterfly standard product microcontroller. It is object-code compatible with earlier ARM® products and fully supported by industry standard cross development tools available on the PC, Sun and HP platforms, including the ARM Software Development Toolkit and software routines for on-chip functions.

The Firefly MF1 Core consists of an ARM7TDMI™ cpu, a purpose built system bus and a number of peripherals, which combine to create a powerful microcontroller. The Firefly MF1 Core can be embedded and treated just the same as any other library component in a standard base array. The cell includes additional circuitry that facilitates easy integration of additional customer logic, which may have been developed in discrete hardware, without any redesign. This satisfies the time to market needs of customers in the highly competitive embedded processing market.

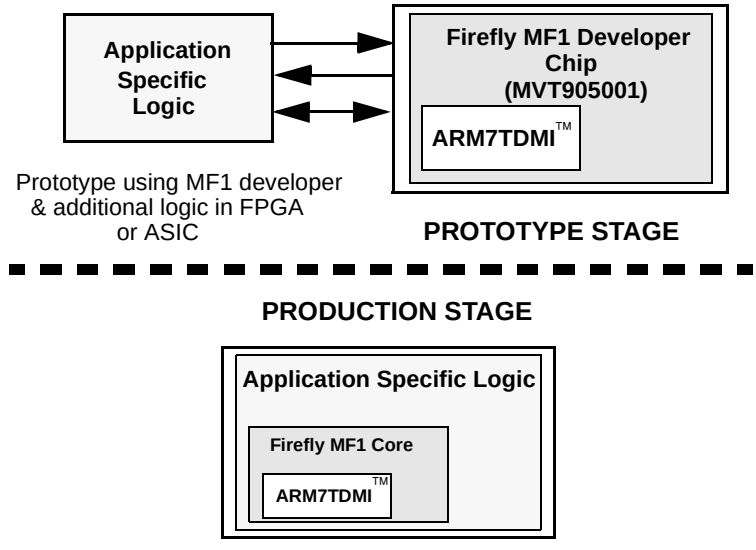


Figure 1 - Embedded Design Made Simple

This embeddable microcontroller is designed around a modular on-chip 32-bit bus architecture that has been developed to facilitate fully-testable, reliable and debuggable embedded processor products.

The Firefly MF1 Core is designed to operate at 44MHz (3 Volts), 27MHz (2 Volts) over a commercial temperature range (0-70°C) when fabricated in 0.35µm CMOS technology.

At 3 Volts/44MHz, up to 37 MIPS (Dhrystone 2.1) of processing power is available.

1.2 Features

The Firefly MF1 Core incorporates an ARM7TDMI™, the industry's most power-efficient embeddable 32-bit RISC core, to which Zarlink has added:

- A System Services Module (SSM) which controls bus functions and provides diagnostic facilities
- A programmable memory interface (MPC)
  - Support for 8,16 and 32-bit data transfers and memory widths
  - A flexible address interface – providing 4 memory areas, each of 4 MBytes
- A serial I/O port (UART)
- A flexible 32-channel interrupt controller with programmable priority (INTC)
- A Direct Memory Access (DMA) controller providing 2 fly-by channels or 1 memory to memory channel (DMAC)
- Four 32-bit timer/counters (TIC)
- An easy route to facilitate the connection of additional application specific logic through the Up Integration Module (UIM) (patent pending)
- Choice of Embedded Microcontroller debug options
- Built-in test modes to ease the production of manufacturing test patterns
- MicroController Expansion Interface to permit customization of the microcontroller core
- Firefly MF1 developer chip (MVT905001, implemented on 0.6 micron CMOS) (see 11.0 “Appendix C: The Firefly MF1 Developer Chip (MVT905001)” on page 136 for more information.)

1.3 Functional Block Description

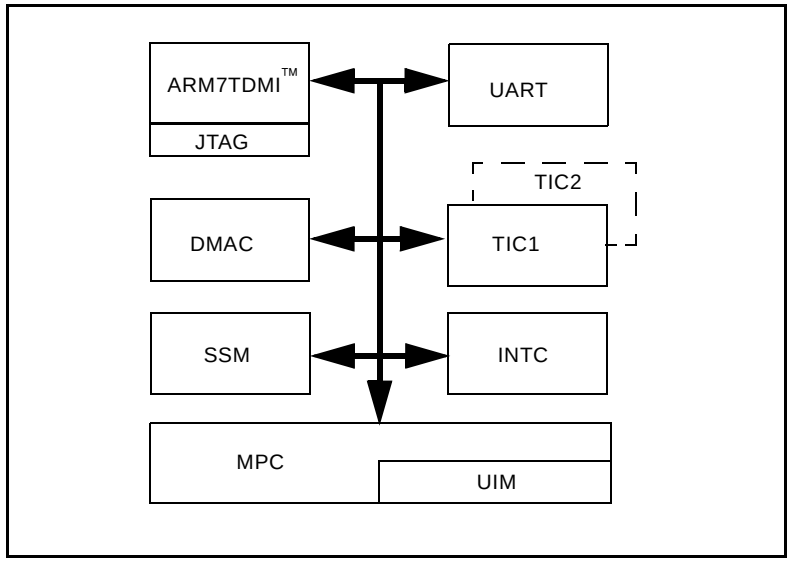


Figure 2 - MF1 Core Block Diagram

1.3.1 Module Bus

The Firefly MF1 Core has a modular bus architecture and specification, via which all on-chip modules communicate with each other. These modules can either be bus masters or slaves. A bus master can initiate a bus access, generate addresses and control read or write transfers. A bus slave responds to a bus master request when selected by the system address decoder, and if required, may assert a wait signal on the bus until the relevant data transfer has been completed. All internal data transfers on the module bus are single cycle.

The microcontroller has three modules that are capable of operating as bus masters. These are the ARM7TDMI™ Core, DMAC and SSM, described below.

1.3.2 ARM® Processor (ARM7TDMI™)

The ARM7TDMI™ is a 32-bit RISC processor core and is object-code compatible with all earlier ARM6 and ARM7 based products. The ARM7TDMI™ is a fully static design and as such consumes dynamic power only when clocked. The processor also has the Thumb® 16-bit instruction set extension.

1.3.3 System Services Module (SSM)

The System Services Module holds the **System Configuration Register** and controls the bus mode (e.g. **RESET, ERROR, RUN,** etc.). For more information about the System Services Module and the **System Configuration Register**, see Chapter 2.0 “System Details” and Chapter 8.0 “Software Debug Facilities”.

1.3.4 Memory/Peripheral Controller (MPC)

The MPC ensures the correct multiplexing of data is applied for bus transfers between the ARM7TDMI™ core and 8-, 16- or 32-bit on-chip (but external to the Firefly MF1 Core) or off-chip peripherals. Four different contiguous memory areas are available, each with an address range of 4 Mbytes and individually programmable wait-state and stop-state generation. A “SWAP” function allows memory area “1”, which is addressed at system reset, to be switched with memory area “4”. This allows, for example, booting from ROM and then switching memory area 1 to address SRAM, so that time-critical software and interrupt routines can operate from fast memory.

1.3.5 Universal Asynchronous Receiver Transmitter (UART)

The full duplex asynchronous channel provides an RS232 type interface, which supports an XON/XOFF software protocol. The Receive and Transmit channels are double buffered. The UART may be polled, or may use an interrupt scheme for module bus transfers. An internal baud rate generator can provide selectable data rates, derived from on-chip sources for an Rx/Tx pair.

Directly triggered DMA transfers with the UART are also possible without the need for CPU intervention.

1.3.6 Interrupt Controller (INTC)

The ARM7TDMI™ core accepts two types of interrupt: **Normal** (IRQ) and **Fast** (FIQ). All Interrupts can be switched between types, depending upon the relative priorities required.

The INTC is the central control logic that decodes the priority level and handles interrupt request signals from a total of 8 fixed, predefined sources within the Firefly MF1 Core and 24 user-definable, external sources. External interrupts can be set for edge or level sensitivity with a polarity option. To minimize interrupt latency, there is a hard-wired priority scheme for each channel for both FIQ and IRQ; alternatively this can be ignored and the priority assessment handled in software.

1.3.7 DMA Controller (DMAC)

Two DMA engines are available on the microcontroller. These may be configured as a pair to provide a memory-to-memory DMA capability between any two locations in the ARM7TDMI™ memory space. Alternatively, they may be used independently for fly-by transfers between off-core requestors and either on-core or off-core locations.

Single or multiple byte transfers (Demand or Burst Mode) are supported and may be word, half-word or byte wide.

1.3.8 Timer/Counters (TIC)

Two dual independent 32-bit timer/counters, with an 8-bit prescaler capability for each counter, are provided (Timers 1A, 1B, 2A and 2B). These are synchronous to the system clock and may be polled, or setup to generate interrupts on over-run, with auto-reload.

1.3.9 Up-Integration Module (UIM)

The Up-Integration Module provides a series of internal (on-chip) connection ports which mimic the MPC external interface. This allows the rapid and efficient integration of customer logic that has been developed by hardware breadboarding, using the external MPC interface of an existing device, such as the MVT905001 Firefly MF1 Developer chip. See section 2.8, “Designing With The Embedded Microcontroller Cell” for more information.

1.3.10 Embedded Microcontroller Debug Options

The Firefly MF1 Core incorporates three sophisticated methods of hardware and software debug. The designer should choose which methods are required. The options are:

- EmbeddedICE™,
- Angel™ Debug Monitor, and
- Logic Analyser coupled with an Inverse Assembler and Zarlink’s Diagnostic Broadcast feature.

1.3.11 Built-in Routes for Manufacturing Tests

The Firefly MF1 Core incorporates two built-in routes for manufacturing test. The first is for the cell itself, the second is a direct path into the up-integrated logic. In this way, the time to generate manufacturing test patterns is greatly reduced.

1.3.12 Microcontroller Expansion Interface

The expansion interface allows customization of the microcontroller core via the addition of new macrocells. For details of available macrocells and the use of this interface, designers should contact their nearest Zarlink Design Centre.

2.0 System Details

2.1 System Address Map

The system address map for the Firefly MF1 Core is shown below in Table 1, “Address Map”.

Address Range	Function
0x0000 0000 >> 003F FFFF	External-1:nScs0/nScs3#
0x0040 0000 >> 1FFF FFFF	External-1 reflected*
0x2000 0000 >> 203F FFFF	External-2:nScs1
0x2040 0000 >> 3FFF FFFF	External-2 reflected*
0x4000 0000 >> 403F FFFF	External-3:nScs2
0x4040 0000 >> 5FFF FFFF	External-3 reflected*
0x6000 0000 >> 603F FFFF	External-4:nScs3/nScs0#
0x6040 0000 >> 7FFF FFFF	External-4 reflected*
0x8000 0000 >> 9FFF FFFF	Reserved
0xA000 0000 >> BFFF FFFF	Reserved
0xC000 0000 >> DFFF FFFF	Reserved
0xE000 0000 >> E001 FFFF	Internal I/O
0xE002 0000 >> FFFF FFFF	Reserved
Note: ’Internal’ and ‘External’ above refer to the Firefly MF1 Core , not the total ASIC.	

Table 1 - Address Map

* Reflected address areas will produce images of data in the original memory area addresses e.g. External-1 will repeat at addresses 0x00400000, 0x00800000, 0x00C00000 etc.

See details of SWAP function in section 2.2, “SWAP Function”.

2.1.1 Internal Memory Area

The internal I/O memory area is subdivided into 32 sub-regions extending from 0xE000 0000 to 0xE001FFFF and is assigned to the on-core modules. Those modules capable of generating addresses are configured as ‘bus masters’ and the remaining modules as ‘bus slaves’.

The bus masters are:

- The ARM7TDMI™ Processor,
- The System Services Module (SSM), and
- The DMA Controller.

As only one bus master can be in control of the module bus at any one time, they are prioritized and given access to the bus in accordance with the arbitration system (see section 2.5, “System Bus Arbitration” for more information.)

2.2 SWAP Function

The **System Configuration Register** (see section Chapter 2.7.1.2 “Register Details”) includes a control bit that swaps the addressed memory areas for chip selects **nScs0** and **nScs3**. This allows ROM to be used for ARM® bootstrap code, and faster memory (e.g. SRAM) to be relocated into the low memory-address locations (including those where exception vector tables reside), once the application code is up and running.

Note: If **SWAP** is used, the set-up parameters for the relevant off-core memory may also need reprogramming, as described in section 3.6.2, “MPC Configuration Register Settings For The Example System Configuration”.

2.3 Address Map For Internal I/O

Table 2, “Peripheral Memory Map” shows how registers for the various modules of the Firefly MF1 Core are allocated to memory areas within the internal I/O memory space. Refer to the relevant chapter for further register descriptions.

Note: All registers except the UART registers must be accessed as 32-bit operands or a run-time bus error will occur. Attempts to access an address that is not mapped to a register, or is designated as reserved, will also result in a bus error**.**

Address Range	Function	Address Range	Function
0xE000 0000- > 1FFF	Reserved	0xE000 D000- > DFFF	Reserved
0xE000 2000- > 2FFF	System Configuration (including SSM)	0xE000 E000- > EFFF	Timer 1 (TIC1)
0xE000 3000- > 3FFF	Reserved	0xE000 F000- > FFFF	Timer 2 (TIC2)
0xE000 4000- > 4FFF	Reserved	0xE001 0000- > 1FFF	Reserved
0xE000 5000- > 5FFF	Reserved	0xE001 2000- > 2FFF	Reserved
0xE000 6000- > 6FFF	Interrupt Controller (INTC)	0xE001 3000- > 7FFF	Reserved
0xE000 7000- >7FFF	Reserved	0xE001 8000- > 8FFF	UART
0xE000 8000- > 8FFF	Memory/Peripheral Controller (MPC)	0xE001 9000- > 9FFF	Reserved
0xE000 9000- > BFFF	Reserved	0xE001 A000- > AFFF	Reserved (Internal)
0xE000 C000- > CFFF	DMA controller (DMA)	0xE001 B000> FFFF FFFF	Reserved

Table 2 - Peripheral Memory Map

2.4 System Reset

The Firefly MF1 Core is reset via its reset port nreset**,** using the signal **nSreset,** which must be generated off-core by the user.

The resulting initialization sequence is as follows:

- The microcontroller enters RESET mode, resetting all internal modules, including the ARM7TDMI™ CPU.
- The ARM7TDMI processor is granted the bus, and proceeds to fetch its next instruction from address location 0x0000 0000.

- The MPC defaults to accessing byte-wide external area 1 as slow memory, with 15 wait states and 15 stop states. The default memory width is determined by signals on two ports (**reset_size<1:0>**) which can select byte, half word or word widths.

2.4.1 Important Notes

- Following controller initialization, the user must configure the MPC appropriately for the off-core devices connected to it. See section 3.4, “Programmer’s Model” for register details.
- Immediately after power-up, **nSreset** should be taken low and **Sclk** should be clocked for at least 2 ARM® processor cycles - to bring the core into a stable state.
- Note: It is recommended that designers place a Bus Hold cell (Type CLBHOLD for CLA200 series CMOS Gate Arrays, Type HOLDX1 for GSC200 CMOS Standard Cell series) on each of the indicated Firefly MF1 ports (which are available at the Firefly MF1 Microcontroller Expansion Interface) shown below.
  The purpose of the cells is to hold each microcontroller internal bus signal to a logical value during the time that the internal bus is held in the reset state (when no bus masters are allocated and the bus remains undriven). If the cells are omitted, then no logical effect will be seen and the microcontroller will continue to function correctly as normal; however, during periods of excessively long active resets (nSReset asserted), a small increase in power consumption may be observed.
  Firefly MF1 Ports where hold cells are recommended include:
  B_ADDR[31..0], B_OPC, B_REQ, B_SEQ, B_SIZE[1..0], B_SUPER, B_WRITE, B_WAIT, B_MODE.
  Other Microcontroller Expansion Interface ports:
  There are already hold cells on the signals B_DATA[31..0] and B_ERROR. B_HOLD is of VHDL Port Type ‘Buffer’ and as such does not facilitate the addition of a Bus Hold cell to itself. This is not a material omission.
  The signals are q_spare, ncs, intclken, a_gnt_spare do not require a Hold cell to be added.

2.5 System Bus Arbitration

The system contains three separate bus masters (modules capable of generating addresses): the ARM7TDMI™ Processor, the DMA controller, and the System Services Module (SSM). The arbitration system uses a “highest priority” scheme. Bus masters request use of the bus from the Arbiter. The highest priority bus master is granted the bus for as long as it wishes irrespective of the demands of lower priority bus masters. The priority levels implemented are as shown in Table 3, “Bus Master Priorities”. For a technical overview of the protocols associated with bus arbitration and bus transactions, see Chapter 10.0 “Appendix B: Modular Bus Operation”.

Priority Level	Function
0- Highest	SSM
1	DMA
2- Lowest	ARM7TDMI™ [1]

Table 3 - Bus Master Priorities

1. ARM7TDMI™ can be promoted above DMA during interrupt services; see below.

The SSM block is primarily used for manufacturing test and system debug and hence requires the highest priority level, although it is not active as a bus master during normal system operation. The ARM7TDMI processor is bandwidth intensive; if not given the lowest priority, then any lower priority master would rarely get access to the bus. A further feature allows the ARM® to have priority on receipt of a FIQ interrupt. In this instance the ARM may be optionally promoted in priority to be above the DMA for the duration of any FIQ. Bit 1 in the **System Configuration Register** enables this function (see Chapter Table 5 - “System Configuration Register (SCR) Details”).

2.6 Off-core Bus Masters

The Firefly MF1 Core provides a method for off-core devices to take control of the off-core (system) bus (i.e., the bus to which the MPC ports are connected). It also allows an off-core master to take control of the on-core internal bus.

Off-core bus mastership is obtained using the External Master/Manufacturing Test ports of the Firefly MF1 Core , namely: **x_req**, **x_gnt**, **x_ctrl**, **x_burst** and **x_write**. There are two different modes of off-core bus mastership:

- where both source and destination devices for the transaction are off-core, and
- where the destination device for the data transaction is within the Firefly MF1 core.

2.6.1 Off-core-only Bus Mastership

When an on-core device (e.g., the ARM CPU or the DMA Controller) has bus mastership, off-core transactions normally take place over the address and data buses, with the MPC outputs actively driven to facilitate those transactions.

However, when an off-core bus master wishes to perform entirely off-core data transactions, it is essential that the MPC address and data bus outputs and the associated memory and MPC control pin outputs, become and remain tristated throughout the pure off-core transactions.

2.6.1.1 On-core Bus Suspension

To prevent any operations occurring on the on-core bus, the control lines are held in a state which prevents any valid transaction. This means that no transactions can be initiated by any on-core bus master while an off-core bus master has obtained an **x_gnt** assertion and has bus mastership. Parallel processing using both the Firefly MF1 Core logic and the off-core logic can NOT therefore be carried out while off-core logic has control of the system bus.

2.6.1.2 Off-core Mastership Protocol

To obtain bus mastership (see <italic>Figure 3 -, "External Mastership Sequence" overleaf), the off-core bus master must:

- Ensure that the signals **x_ctrl**, **x_write** and **x_burst** are low, and assert the signal **x_req**;
- Wait until signal **x_gnt** has been asserted;
- Wait for the next Firefly clock cycle (indicated by **SCIk** going low), at which point the Firefly MF1 output drivers on applicable signals begin to turn off;
- Wait until the address, data and control signals from the Firefly MF1 core (and, if necessary, any other drivers on the off-core bus) have turned off.

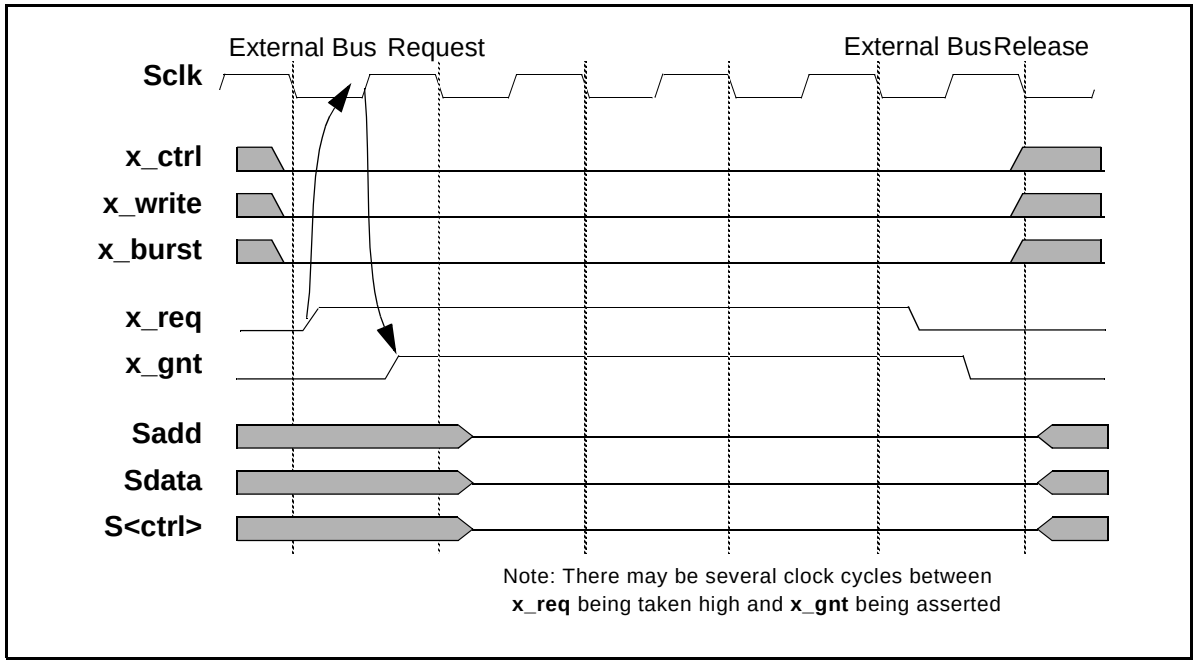


Figure 3 - External Mastership Sequence

The off-core bus master now has both bus mastership and safe conditions under which to begin its transactions with other off-core devices across the system bus.

Note: If more than one off-core bus master device is required, then it will be necessary for the designer to provide an arbitration facility between these masters, whose outputs to the Firefly MF1 core then obey the protocol described in this section.

2.6.1.3 FIQ Promotion Warning

Note that when using the external bus mastership protocol mechanism, FIQ promotion should NOT be enabled in the **System Configuration Register**. If it is, then a FIQ exception will cause **x_gnt** to be deasserted, as the ARM7TDMI core will be promoted above the off-core master for the duration of the interrupt. If promotion is disabled, then FIQs arising while an off-core master has control will not be serviced, at least until the off-core master releases the bus, because the off-core master normally has higher priority than any other master.

2.6.1.4 Timing For Bus Mastership By Off-core Masters

The timing parameters for off-core bus mastership are defined in Chapter 9.0 “Appendix A: Firefly MF1 Core Datasheet”, in the timing diagram and table under section 9.18, “External Master” on page 131. To request bus mastership, **x_req** must be setup before the rising edge of **Sclk**. **x_gnt** is rising edge triggered.

The tristate disable control is applied on the following falling edge after **x_gnt**: hence, outputs are switched on or off during the cycle following an **x_gnt** change - NOT when **x_gnt** is first asserted.

Figure 3, “External Mastership Sequence” on page 22 shows the external master asserting **x_req**, being granted the bus, and then releasing it several cycles later.

2.6.2 Off-core Bus Mastership Transactions With An On-core Device

For an off-core bus master to perform a transaction with an on-core slave or master device then the off-core bus master must use the 'Manufacturing Test Protocol' to facilitate the transactions. This is NOT the same as a standard bus transaction and is outside the scope of this Manual. It is described in Appendix 2 of the guide **System Design using the Zarlink Firefly MF1 Cell**. Please contact your nearest Zarlink Design Centre for a copy of this document.

2.7 System Configuration

After reset, the system registers are initialized with the values shown in the tables below.

The user should then programme values for desired operation of the device.

2.7.1 System Services Module

2.7.1.1 Register Map

Note: The “Side Effect on Read” column is used to identify registers which cause one or more side-effects when read.

Address (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Side Effect on Read	Description
+000	Diagnostic Configuration	32	00000000	RW		Detailed in Chapter 8.0 “Software Debug Facilities”
+004	System Configuration	32	00000000	RW		Only least significant 6 bits used

Table 4 - System Services Module Registers

2.7.1.2 Register Details

The **System Configuration Register** controls the operation of various system functions, as described in Table 5, “System Configuration Register (SCR) Details”. The DR column is blank unless a read is destructive.

Bit	Function	Description	Reset value	Type	DR[1]
31:6	Reserved	Must be 0	0	RW	
5:3	DMA Trigger Source	Selects the DMA request input source [0y000 - 0y111] See <italic>Table 6, “DMA Channel 1 Trigger Selection,” on <italic>page 24 for full details.	0y000	RW	
2	DMA Trigger Mode	Controls DMA request input source - set selection [0] Standard [1] Butterfly Compatible	0	RW	

Table 5 - System Configuration Register (SCR) Details

Bit	Function	Description	Reset value	Type	DR[1]
1	Bus Arbitration Mode	[1] Promote ARM FIQ See 2.5, “System Bus Arbitration” for more information. [0] No Promotion	0	RW	
0	Memory Area Control Status	[1] Swap: Swaps external nScs0 and nScs3 address ranges (see section 2.1, “System Address Map” on page 18 for more information) [0] No Swap	0	RW	

Table 5 - System Configuration Register (SCR) Details (continued)

1. An R in this column indicates that a read is destructive.

2.7.1.3 DMA Triggers

The following table describes the encoding system for allocating DMA channel 1 DMA-requests to sources that have a DMA capability. DMA channel 2 DMA-request is ONLY connected to the dreq2 input port.

SCR[5:3]	SCR[2] = 0 (Butterfly-compatible mode)	SCR[2] = 1 (default mode)
0 0 0	Off-core dreq 1	Off-core dreq 1
0 0 1	dreq1_src(3)	Off-core dreq 2
0 1 0	dreq1_src(2)	TIC 1A interrupt
0 1 1	UART 1 receive	UART receive
1 0 0	ext_int(1)	ext_int(1)
1 0 1	dreq1_src(1)	ext_int(2)
1 1 0	dreq1_src(0)	TIC 2A interrupt
1 1 1	UART 1 transmit	UART transmit

Table 6 - DMA Channel 1 Trigger Selection

2.7.2 Interrupt Sources

The following is the channel allocation number for each peripheral. More information on interrupt control may be found in Chapter 5.0, “Interrupt Controller (INTC)” on page 76.

Channel	Interrupt Source	Function
0 (highest priority)	ext_int(3)	User defined
1	ext_int(4)	User defined
2	ext_int(5)	User defined

Table 7 - Interrupt Sources

Channel	Interrupt Source	Function
3	DMAC	Channel status change
4	ext_int(6)	User defined
5	ext_int(7)	User defined
6	TIC1	Time out A
7	TIC1	Time out B
8	ext_int(1)	User defined
9	ext_int(8)	User defined
10	ext_int(9)	User defined
11	ext_int(10)	User defined
12	ext_int(11)	User defined
13	ext_int(12)	User defined
14	UART	Transmit or receive error; modem status change
15	UART	Receive Buffer Full
16	UART	Transmit Buffer Empty
17	ext_int(13)	User defined
18	TIC2	Time out A
19	TIC2	Time out B
20	ext_int(2)	User defined
21	ext_int(14)	User defined
22	ext_int(15)	User defined
23	ext_int(16)	User defined
24	ext_int(17)	User defined
25	ext_int(18)	User defined
26	ext_int(19)	User defined
27	ext_int(20)	User defined
28	ext_int(21)	User defined
29	ext_int(22)	User defined
30	ext_int(23)	User defined
31(lowest priority)	ext_int(24)	User defined

Table 7 - Interrupt Sources (continued)

The four timer interrupts (channels 6, 7, 18, and 19) are edge-triggered. All other interrupt sources that are generated on-core are level sensitive, active ***HIGH***.

Directly driven external interrupts are user defined, and can be set up for either edge or level sensitivity. For these, the edge/level and polarity registers need to be set up to match the required operation. For example, the values that need to be written for ext_int(1) and ext_int(2) are:

edge/level	0x000C00C0	external interrupts 1 & 2 are level sensitive
	0x001C01C0	external interrupts are edge sensitive
polarity	0xFFFFFFFF	external interrupts are active high or rising edge triggered
	0xFFEFFEFF	external interrupts 1 & 2 are active low or falling edge triggered

Table 8 - Interrupt Register Values

2.8 Designing With The Embedded Microcontroller Cell

This section considers two different design and prototyping methods and shows how the Firefly MF1 Core can be used in real applications. The methods are as follows:

- The designer develops a solution in external hardware (e.g. FPGA) and connects it to the memory I/O on the Firefly MF1 developer chip using a development board. Once debugged, the external logic is up-integrated, using the Zarlink Up Integration Module (UIM), to produce the ASIC solution. The UIM is effectively a sophisticated ‘signal router’; it controls the flow of data, address and control signals between the MPC, SSM, the ‘internal’ UIM port and any ‘external’ logic or memory. As well as being able to ‘Up integrate’ external logic and/or memory, the UIM also permits the use of external microcontroller bus masters and allows these to be up-integrated if required.
- The designer develops and proves a solution in a simulation environment only, and proceeds directly to the final ASIC.

2.8.1 Method 1: Additional Logic Is Developed Using External Hardware

This development consists of two phases. The first is the prototype phase in which the additional logic is developed externally, probably using FPGA’s on a daughter board attached to the expansion port of a development board, such as the Zarlink Semiconductor MAP-2TE. The second phase involves up-integration whereby the additional logic (and possibly the memory too) is re-targeted and a final ASIC also is created.

The Firefly MF1 Developer Chip (MVT905001) used on the MAP-2TE can be used for prototyping on a custom designed target. The additional logic could then be developed as peripherals connected to the MPC in gate array or FPGA.

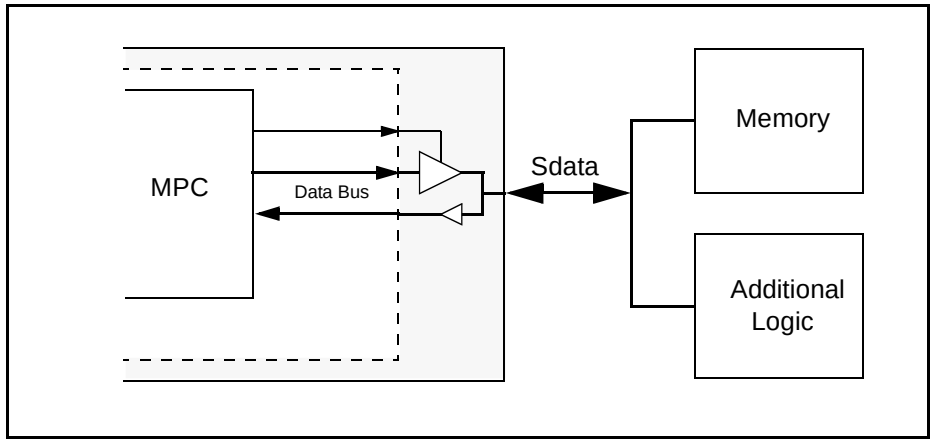


Figure 4 - Development Using External Logic

When the additional logic has been satisfactorily designed and verified, it can be up-integrated, back onto the chip, to create the application specific solution. The logic netlist is simply connected to the internal UIM data ports and the netlist for the completed ASIC can be produced. The UIM ports are designed to mimic the behaviour of the MPC external I/O's, the only difference being the bidirectional Sdata bus of the MVT905001, which is split into **uim_mdata_in** and **uim_mdata_out**. As bidirectional functionality is usually obtained within an I/O cell, and these cells are removed when up-integration occurs, it is unlikely that any logic changes are required. Similarly, as the data paths still involve the MPC, no software changes should be required.

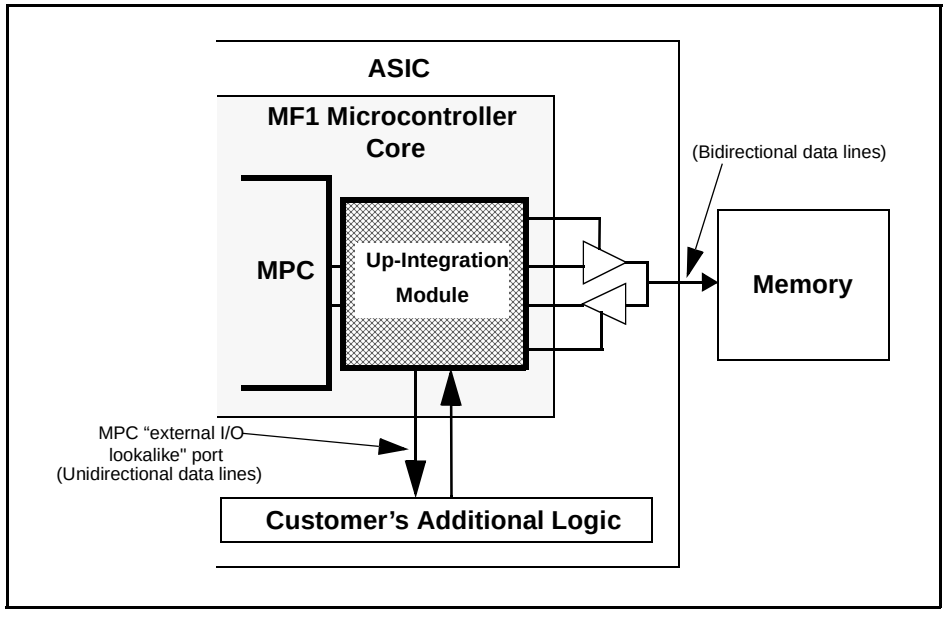


Figure 5 - Integration Using The UIM

2.8.2 Method 2: Additional Logic Developed In Simulation Environment Only

If the designer is using simulation-only to verify their logic, then they can connect it directly to the Embedded Microcontroller. This method of development is the same as the previous one except that the first, prototyping, stage is not used.

The interface point for up-integration will not actually be at the MPC pins but will be to a port on a module called the Up-Integration Module (UIM) (see "Appendix A: Firefly MF1 Core Datasheet"). The UIM port mimics the normal

memory I/O (note that bidirectional buses are split), and will be controlled from the existing MPC in the same way as any other memory area. The designer is able to connect directly to it and fully simulate the system.

When the development phase is over the complete netlist is then turned into the final ASIC solution.

2.8.3 UIM Interconnection Details

Table 9, “UIM Interface Ports” below shows the ports used to interface with the UIM. It also shows the equivalent I/O connection points on the MPC external interface. See “Appendix A: Firefly MF1 Core Datasheet” for full details of port descriptions and UIM timing information.

UIM PORT	TYPE	MPC I/O Equivalent	Description
uim_mdata_in<31:0>	I	**Sdata**<31:0>	UIM Data inputs
uim_mdata_out<31:0>	O	**Sdata**<31:0>	UIM Data outputs
uim_mdata_tri	O		UIM Data I/O tristate control if needed
uim_madd_out<21:0>	O	**Sadd**<21:0>	UIM Address outputs
uim_madd_in<21:0>	I	**Sadd**<21:0>	UIM Address inputs - used if external ‘master’ is connected to UIM port
uim_nmcs<3:0>	O	**nScs**<3:0>	UIM Chip selects for internal logic/memory
uim_nmwe<3:0>	O	**nSwe**<3:0>	UIM Write enables for internal logic/memory
uim_nmoe	O	nSoe	UIM Output enable for internal logic/memory
uim_mints<3:0>	I		Selects which chip selects lines are internal
uim_mdints<1:0>	I		Selects whether DMA flyby aware components are connected to UIM port
uim_test_mode	I		Selects test route for logic connected to UIM port

Table 9 - UIM Interface Ports

2.8.4 Internal And External Memory Area Selection

Table 10, “Selection Of Internal And External Memory Areas” below details the **uim_mints**<3:0> signals which are used to indicate which chip select areas (**nScs**<3:0>) are to be treated as internal to the ASIC. For example, if external logic selected by **nScs2** is to become up-integrated, then **uim_mint**(2) should be taken high. By hardwiring these ports internally the selection becomes permanent; however, if the **uim_mints** ports are bonded out to pins, then it is possible to control the selection externally. This is particularly useful if the up integrated logic contains a ROM, as it becomes possible to run the ASIC from either internal or external ROM.

UIM PORT	TYPE	Description
uim_mints<3:0>	I	logic high - chip selects lines are internal logic low- chip selects lines are external

Table 10 - Selection Of Internal And External Memory Areas

2.8.5 Internal And External DMA Flyby Component Selection

Table 11, “Internal And External DMA Flyby Component Selection” below details the **uim_mdints**<1:0> signals which are used to indicate whether a DMA flyby component has been connected to the UIM or MPC port. For example, if a flyby-aware component selected by dack1 is to be up-integrated, then **uim_mdint**(1) should be taken high. By hardwiring these ports internally, the selection becomes permanent. However, if the **uim_mdints** ports are bonded out to pins, then it is possible to control the selection externally.

Note: the DMA permits the polarity of **dack** to be programmed. When connected to the UIM port, **dack** MUST be programmed as active high.

UIM PORT	TYPE	Description
uim_mdints<1:0>	I	01 = **dack1** relates to UIM logic slave 10 = **dack2** relates to UIM logic slave (**dack** must be programmed high for UIM)

Table 11 - Internal And External DMA Flyby Component Selection

2.8.6 Test Mode Selection

Table 12, “Operational Mode Selection” below details the **uim_test_mode** and test signals which are used to indicate which test mode has been selected. For details on the methods of testing, see section 2.10, “Manufacturing Test” on page 32.

UIM PORT	TYPE	Description
uim_test_mode	I	These pins are encoded and determine the current operational state - see section 2.10, “Manufacturing Test” on page 32
Test	I	

Table 12 - Operational Mode Selection

2.9 Embedded Microcontroller Debug Options

During system development, there is a conflict between the requirement to maintain maximum processor throughput, and the desire to interact with the system to achieve maximum observability of the processor activity. Traditionally, this has been resolved partly by the use of expensive bond-out chips and In Circuit Emulation (ICE) support, to monitor and debug application code running on the processor. With the advent of low-cost deeply embedded systems, further on-chip integration has increased, resulting in constraints on system observability through a limited number of external package pins. The required capital investment to develop high speed analysis and debugging systems based on traditional in-circuit emulation techniques has risen dramatically, while the target market requires cheaper and more flexible tools. Furthermore, system integration is increasingly performed at higher levels of the design, through analysis and debugging of high level language source code and real time kernel calls.

The Firefly MF1 Core incorporates three sophisticated methods of hardware and software debug. The designer should choose which methods are required and then make sure that access can be made to the relevant ports on the cell. The options are:

- EmbeddedICE™.
- Angel™ Debug Monitor.
- Logic Analyser coupled with an Inverse Assembler and Zarlink’s Diagnostic Broadcast feature.

2.9.1 EmbeddedICE™

EmbeddedICE™ is an extension to the architecture of the ARM® family of RISC processors and provides the ability to debug cores that have been deeply embedded into systems. It consists of three parts (see 8.3.1.2, “EmbeddedICE™ Module” on page 111 for more information on EmbeddedICE):

1. a set of debug extensions to the ARM core;
2. the EmbeddedICE macrocell, to provide access from the outside world to the extensions; and
3. the EmbeddedICE interface, to provide communication between the EmbeddedICE macrocell and the host computer.

2.9.1.1 Debug Extensions To The ARM® Core

The presence of debug extensions to the ARM core are indicated by the “D” suffix on the core name (ARM7T**D**MI™). The extensions consist of a number of scan chains around the core and some additional pins which are used to control the behaviour of the core for debug purposes. The three most significant pins are:

- BREAKPT, which allows external hardware to halt execution of the processor for debug purposes;
- DBGRQ, which is a level-sensitive input that causes the core to enter debug state immediately the current instruction has completed execution; and
- DBGACK, which is an output from the ARM that goes HIGH when the core is in debug state.

Please refer to the “Debug Interface” section of the ARM7TDMI data sheet for further details.

2.9.1.2 EmbeddedICE™ Macrocell

The EmbeddedICE™ macrocell is the integrated on-chip logic which provides debug support for ARM cores. Its presence is indicated by the “I” suffix on the core name (ARM7TDMI**™**). The EmbeddedICE macrocell is programmed in a serial fashion through the TAP controller on the ARM7TDMI via the JTAG interface. This programming is achieved by using the EmbeddedICE (or alternatively, Mutli-ICE) interface unit.

Please refer to the “ICEBreaker” section of the ARM7TDMI data sheet for further details.

2.9.1.3 EmbeddedICE™ Interface

The EmbeddedICE interface is a JTAG protocol conversion unit. This translates the debug protocol messages sent out by the debugger into JTAG signals that can be sent to the EmbeddedICE macrocell (and vice versa).

Please refer to ARM applications note “Apps31vB” for further details.

Ports that must be connected for EmbeddedICE:

PORT	Type	Description
ntrst	I	Test Reset
tdi	I	Test Data In
tms	I	Test Mode Select
tck	I	Test Clock
tdo	O	Test Data Out

Table 13 - Ports Requiring Connection For EmbeddedICE™

2.9.2 Angel™ Debug Monitor

Angel™ (the Angel Debug Monitor) is a program which allows rapid development and debugging of applications running on ARM-based hardware. Angel can debug applications running in both ARM® and Thumb® state on target hardware.

A typical Angel system has two main components that communicate via a serial link.

- Debugger (runs on the host computer). This component gives instructions to Angel and displays the results obtained from it.
- Angel Debug Monitor: This component runs alongside the application being debugged on the target platform.

Angel can be used for:

- evaluating existing application software on real hardware;
- developing software applications on development hardware; and
- bringing into operation new hardware that includes an ARM processor.

These activities require some understanding of how the Angel™ components work together. For further information refer to the ARM® “SDT User Guide” (ARM DUI 0040C) and the ARM “SDT Reference Guide” (ARM DUI 0041B).

If using the UART within the Firefly MF1 Core , ports that must be connected for Angel are shown in Table 14 below.

PORT	Type	Description
utxd	O	UART Transmit Data
urxd	I	UART Receive Data

Table 14 - Ports Requiring Connection For Running Angel™

2.9.2.1 How A Debug Monitor Differs From EmbeddedICE™

A debug monitor, such as Angel™, is an application that runs on the target hardware in conjunction with the user's application. When the target hardware powers up, Angel installs itself by initializing the vector table so that it takes control of the target hardware when an exception occurs. Communication coming in from the host causes an interrupt, halting the user application and calling the appropriate code within Angel. After Angel has dealt with the interrupt control is passed back to the user application. This can complicate matters if the application also requires access to interrupts. Angel requires ROM to store the debug monitor code, RAM to store its data, and control over the exception vectors to allow it to gain control of the ARM7TDMI™ while the user's application is running.

EmbeddedICE™ on the other hand requires no such resource. Rather than existing as an application on the target hardware, it works by using a combination of the additional debug hardware on the ARM7TDMI core and the interface box which handles communication between the core's debug hardware and the host. EmbeddedICE has been designed to allow debugging via the JTAG port to be as non-intrusive as possible:

- No special hardware is required in the target system to support debugging (the EmbeddedICE macrocell and the JTAG TAP controller are all that is required).
- No memory in the target system need be set aside for debugging, and no special software need be incorporated to allow debugging.

2.9.3 Logic Analyser

A logic analyzer is a useful tool for examining hardware and software interactions. Coupled with an Inverse Assembler for the ARM7TDMI, it becomes a powerful debugging tool. It can be triggered by addresses, data, control signals or any combination of these. It can also be set to do conditional triggering. An example might be: trigger on the third occurrence of address x AND if the data is y AND if it is during a byte access. This type of triggering is done without altering the application code in any way.

Transfers between blocks internal to the Firefly MF1 Core will not be visible outside the device. To overcome this, the SSM module allows data on the internal bus to be broadcast externally, over the address and data busses. It also provides diagnostic flags, to indicate the operation in progress.

To use Broadcast Diagnostics, the analyzer needs to connect to the ports:

PORT	Type	Description
bdiag<3:0>	O	Broadcast Diagnostic signals
saddr_out<15:0>	O	Address Bus
sdata_out<15:0>	O	Data Bus
nscs<3:0>	O	Chip select outputs

Table 15 - Ports Required To Interpret Broadcast Diagnostic Cycles (Only)

2.10 Manufacturing Test

Firefly has three basic modes of operation which are selected via the test and uim_test_mode ports:

- **Normal running mode**: the mode in which the device will operate when it is incorporated into the final system and is running the system application software.
- **Firefly test mode**: the mode that is used for manufacturing test of the Firefly cell. Basically it isolates Firefly from any logic connected to the UIM port and provides a pathway for test data to be applied directly to the Firefly cell. Note that this mode is further split into two separate modes (System and Macrocell test modes.)
- **UIM Logic test mode**: the mode that is used for manufacturing test of any logic connected to the UIM port. Basically, it disconnects the Firefly cell and provides a route for customers to write and read from the external memory ports (e.g., x_data_in, x_noe_in, etc.) directly to/from the UIM ports (e.g., uim_mdata_out, uim_nmoe_out). Hence any logic connected to it can be tested in isolation.

The operational modes are selected by using 2 ports as follows:

test	uim_test_mode	Description
0	0	Normal operating mode
0	1	Firefly System test mode
1	0	Firefly Macrocell test mode
1	1	UIM Logic test mode

Table 16 - Details Of Selecting Operating Modes

It is important to note that the two Firefly test modes are compulsory and visibility of certain ports must be achieved when operating in this mode. As well as applying vectors to the MF1 cell during manufacturing test, it is also necessary to perform IDDQ testing. The Firefly manufacturing test vectors include IDDQ strobe points at which the manufacturing tester will stop and measure the quiescent current in the VDD supply rail. Therefore, it is important that all devices using the MF1 cell are capable of having IDDQ measurements made.

A summary of the port visibility requirements follows; full details of the test requirements can be found in the document “System Design Using the Zarlink Firefly MF1 Cell” (contact your nearest Zarlink design centre for a copy of this guide).

2.10.1 Testing The Microcontroller Core, Minus The UIM And Any Integrated Logic

In order to achieve an acceptable fault coverage figure, it is necessary to fully exercise the Firefly cell. This is done using either the Firefly Macrocell test mode or the Firefly System test mode.

2.10.1.1 Firefly Macrocell Test Mode

The Macrocell test mode uses the Test interface ability of the internal bus that is used to interconnect the macrocells within Firefly and as such requires access to certain ports, as shown in Table 17, “Ports Required For Firefly Macrocell Test Mode” below.

Signal [width]	Type	Description [test = 1 and uim_test_mode = 0]
x_data_out[15:0], x_data_in[15:0]	IO	Used to create a 32 bit bidirectional bus
x_add_out[15:0], x_add_in[15:0]		
x_tri_data_op, x_tri_add_op, x_tri_data_ip	O	Direct access to these signals is not required, however they are used to control the I/O’s for the 32 bit bidirectional bus
x_write, x_burst, x_ctrl, x_req	I	Test bus control signals
x_gnt	O	Test bus control signal
Note: It is important that the TIC macrocell is enabled during this mode hence ten1a & 1b and ten2a & 2b should be pulled high. The tests assume an 8 bit ROM on power up so reset_size[1:0] should be pulled low. Unused upper 16 bit x_data_in [31:16] should be pulled low.		

Table 17 - Ports Required For Firefly Macrocell Test Mode

2.10.1.2 Firefly System Test Mode

The System test mode is used to verify the interaction of the various macrocells that make up the MF1 cell and as such, requires access to certain ports as shown in Table 18, “Ports Required For Firefly Macrocell Test Mode” below:

Signal [width]	Type	Description [test = 0 and uim_test_mode = 1]
x_data_out[15:0], x_data_in[15:0]	IO	Used to create a 16bit bidirectional memory data bus
x_tri_data_op, x_tri_data_ip	O	Direct access not required - needed to control I/O’s for memory data bus
x_add_out[15:0], x_add_in[15:0]	IO	Used to create a 16bit bidirectional memory address bus
x_tri_add_op	O	Direct access not required - needed to control I/O’s for address data bus
x_ncs_out[3:0], x_noe_out, x_nwe_out[1:0]	O	Memory control signals
x_tri_ctrl_op	O	Direct access not required - needed to control I/O’s for Memory control signals
x_write, x_burst, x_ctrl, x_req	I	External master bus control signals
x_gnt	O	External master bus grant signal
dreq1&2	I	DMA request signals
dack1&2	O	DMA acknowledge signals
ten1a & 1b, ten2a & 2b	I	TIC enable signals - Note these can be combined as single signal for each timer module e.g. ten1a&1b combined as ten1.
tpwm1 & 2	O	TIC PWM signals
urxd, ucts, ueclk	I	UART data, control and clock signals
utxd, urts	O	UART data and control signals
iextint1&2	I	Firefly system interrupt signals
b_clk, nreset	I	Firefly system control signals
Note: In this mode it is important that the spare interrupts - iextint[3:22] are all pulled low also the tests assume an 8 bit ROM on power up so reset_size[1:0] should be pulled low. For the Uart tests to function correctly signal UDCD should be pulled high whilst signals URI and UDSR should be pulled low.		

Table 18 - Ports Required For Firefly Macrocell Test Mode

2.10.2 Testing The Integrated Logic Connected To The UIM

In order to aid the testing of the integrated logic it is possible to use the UIM to configure the data paths, such that UIM ports are connected directly to package pins. In this mode, control signals **nScs**, **nSwe** and **nSoe** will become inputs feeding directly to the corresponding UIM ports. The **Sdata_in**<31:0> and **Sdata_out**<31:0> ports will be fed directly to **uim_mdata_out**<31:0> and (from) **uim_mdata_in**<31:0> with **Sdata_tri** controlled by **nSoe**. **Sadd**<22:0> will become fixed inputs driving directly on to **uim_madd_out**<22:0>. The logic connected to the UIM can then be driven directly from external (package) pins.

OUTPUT Signal [width]	DRIVEN BY Signal [width] [test = 1 and uim_test_mode = 1]
x_data_out[31:0]	uim_mdata_in[31:0]
x_tri_data_op	x_noe_in
x_tri_data_ip	logic zero i.e. active
x_tri_addr_op	logic one i.e. tristated
x_tri_ctrl_op	logic one i.e. tristated
uim_maddr_out[21:0]	x_addr_in[21:0]
uim_mdata_out[31:0]	x_data_in[31:0]
uim_mncs_out[3:0]	x_ncs_in[3:0]
uim_mnwe_out[3:0]	x_nwe_in[3:0]
uim_mnoe_out	x_noe_in
uim_mnub_out	x_nub_in
uim_mdata_tri	NOT(x_noe_in)

Table 19 - How Signals Are Driven In UIM Logic Test Mode

As well as applying vectors to the MF1 cell during manufacturing test, it is also necessary to perform IDDQ testing. The Firefly manufacturing test vectors include 5 IDDQ strobe points at which the manufacturing tester will stop and measure the quiescent current in the VDD supply rail. It is therefore important that all devices using the MF1 cell are capable of having IDDQ measurements made.

Briefly this means that they should contain no floating buses and all I/O pullup/pulldown resistors should be capable of being disconnected. For full details of IDDQ requirements reference should be made to section 13 of the CLA200 Design Manual (Publication number DM4805, DM4806)

For full details of the IDDQ strobe points which should be used, refer to the document:

Fault Simulation results for Zarlink Firefly1 (MF1) cell, RDS/ESG/182-30.1

2.11 Advantages Of Using The UIM In The Firefly MF1 Core

- Using this method eliminates the need for any software changes when the logic becomes internal (i.e., any Flyby DMA is still treated as external, as it is the UIM which controls the tristate enable on the data output drivers.) Similarly, the MPC can still act as though the designers logic is located in an external memory/peripheral area; the UIM simply disables the external I/O pins. Note that if broadcast diagnostics are enabled and transactions are made to/from memory and/or peripherals connected to the UIM port, the UIM will enable the external I/O pins and hence, make the transactions visible for external debug purposes.
- Customer logic is integrated into the ASIC by using an interface which looks exactly the same as the external MPC (with the exception of the bidirectional **Sdata** pins). In this way, once any external I/O cells have been removed, it should not be necessary to make any alterations to custom logic when the final ASIC, including the Firefly MF1 Core is produced.
- As the MPC is still used, full byte packing and unpacking facilities are available to access internal customer logic without the overhead of a second MPC.
- If the select lines are bonded out to package pins, the option is available to use either the internal logic OR to switch back to using external logic, as in the hardware development phase. This helps with final debug, particularly if the up-integration includes boot ROM. It also makes the task of producing future variants easier.
- When the broadcast diagnostic feature is turned on, the external data bus drivers are enabled, making data flow visible off-chip, even though the transfers are essentially internal. This permits the use of code disassembly, using a logic analyser, if any embedded memories have been up-integrated.

3.0 Memory/Peripheral Controller (MPC)

3.1 Overview

The MPC acts as the main gateway between on-core and off-core bus systems. The off-core bus (or 'system bus' must be capable of interfacing to a multitude of standard parts which could be used in conjunction with an ARM® microcontroller within embedded applications.

Features of the MPC include:

- performs accesses to memories (ROM, SRAM), and peripherals (ADC's, DAC's, UART's etc.);
- generates all control signals to access external components (chip-selects, write-enables, output-enable, etc.);
- provides dynamic bus sizing, so that accesses of the correct width are directed to the off-core component (8-, 16- and 32-bit data widths supported);
- correctly handles16-bit SRAM's with Upper and Lower Byte selection;
- provides programmable wait state generation when accessing specific off-core devices;
- provides programmable Start- and Stop-wait state generation, allowing for the slow turn-on or turn-off times of some devices (e.g. ROM);
- allows independent access, Start- and Stop-wait state generation for reads and writes;
- can perform an external access with zero wait states at the maximum system clock speed;
- supports "fly-by" DMA operation so that on-core to off-core, off-core to on-core, and off-core to off-core modes are supported;
- provides a SWAP function to allow fast SRAM devices connected to **nScs3**[1] to be switched to external area 1, replacing slow external ROM used at start-up (see section 2.2, "SWAP Function" on page 19); this minimises access latency for the exception vectors (for example, when executing an Interrupt service routine); and
- offers software compatibility with the MPC on the Zarlink Butterfly microcontroller.

3.2 Architecture

The diagram in Figure 6, "MPC Functional Blocks And System Interconnections" below shows the various functional blocks within the MPC. The MPC ports are not directly visible to the user: instead they are routed through the Up-Integration Module (UIM). This is to allow for the swap between internal and external memory areas and permits the flow of data between the two. The functions of the various external pins are described more fully in <Italic (Body)>Table 30, "MPC Off-core Ports" on <italic>page 56.

1. mapped to external area 4 at start up

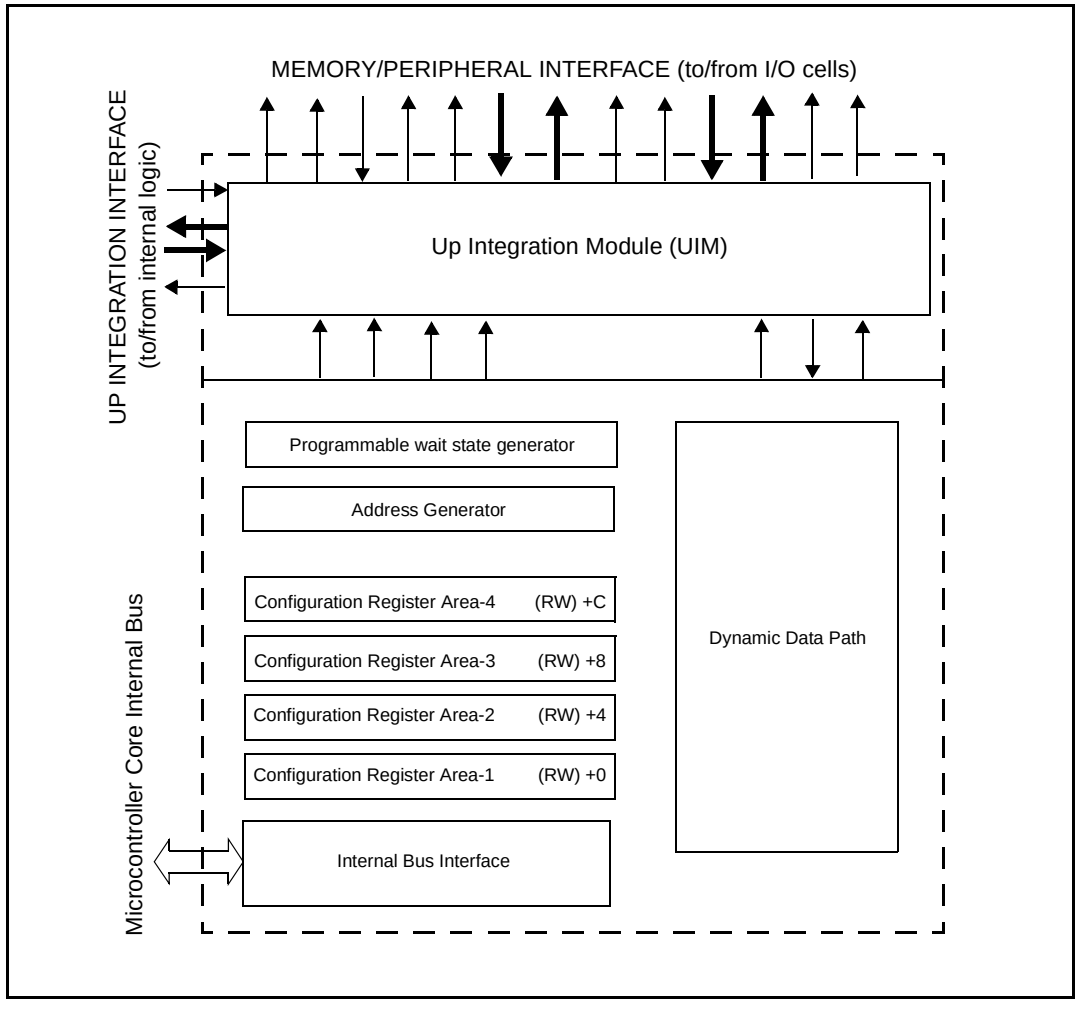


Figure 6 - MPC Functional Blocks And System Interconnections

3.3 Operational Description

In order for the microcontroller to do useful work, it must interface successfully to external memory and peripheral subsystems. The Memory and Peripheral Controller (MPC) is designed to provide a “glueless” interface to common memory sub-systems, such as ROM, EPROM & SRAM. It is also designed to interface to many types of external peripheral interface functions, ranging from analogue interfaces such as A/D and D/A convertors, to networking interfaces such as Serial Communications Controllers and Ethernet ports.

The MPC provides a standard functional set of external interface signals, for which it derives the appropriate timing relationships in order to exchange data between the microcontroller and external devices.

Throughout this section, please refer to the **MPC Configuration Register** format described in Table 25, “MPC Register Map” on page 51. Note that the following sections detail the full MPC configuration (of which Butterfly compatibility is a subset).

3.3.1 Memory Areas

The MPC is able to access up to four memory areas with varying characteristics, allowing a memory subsystem to be constructed using most appropriate types of memory. For example, one memory area could be set-up to contain slow ROM, another for fast SRAM, and a third for an external peripheral.

The characteristics of each memory area are set-up in a **Configuration Register** (see Figure 6, "MPC Functional Blocks And System Interconnections" on page 38.) The parameters that can be varied include:

- automatic wait state generation, or wait indication supplied by memory subsystem;
- the number of Access-wait states for read and/or write;
- the number of Start- and Stop- states (to allow slow memory/peripherals time to turn off) for read and/or write;
- memory size (e.g., 8-, 16- or 32-bits wide);
- read only or read/write;
- memory or peripheral;
- the capability to accept writes to less than the whole memory width; and
- the capability to support 16-bit memory with “built-in” byte selection.

These parameters are described more fully in the following sections.

Note that the Firefly MF1 Core allows external masters to take control of the off-core memory-mapped areas. This facility is described in section 2.6, “Off-core Bus Masters” on page 21.

3.3.2 Signal Relationships

Figure 7, "MPC External Interface Signals" below shows the interrelationship of external signals of the MPC.

NOTE: This diagram assumes that the MPC has been connected to standard I/O cells in order to create the signals shown. See *Figure 17, "Example System Configuration (Little Endian)" on page 58* for an example.

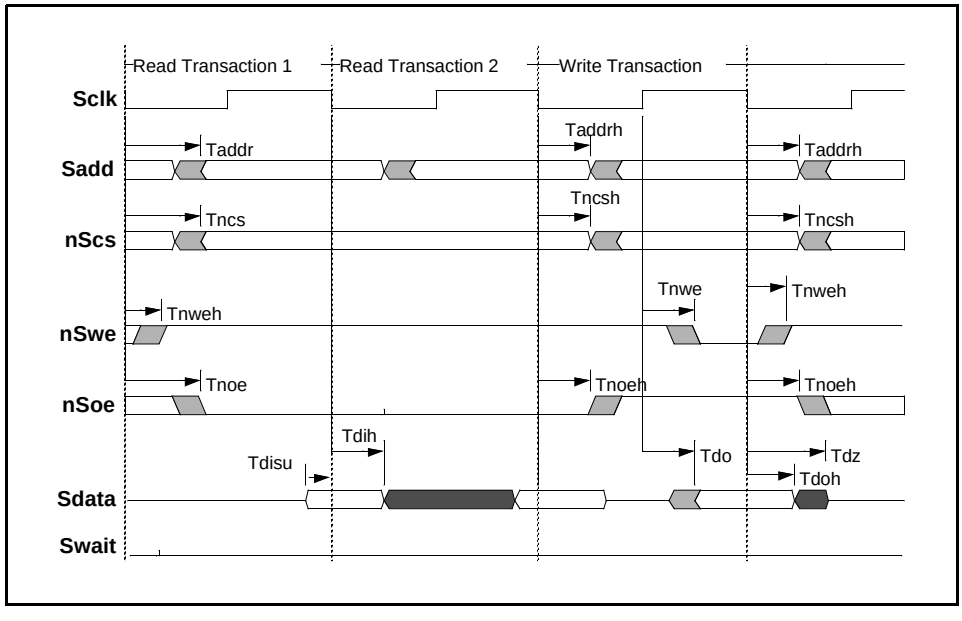


Figure 7 - MPC External Interface Signals

During a read transaction the address (**Sadd**) and chip selects (**nScs**[3:0]) are generated shortly after the falling edge of the system clock (**Sclk**). The write enables (**nSwe**[3:0]) are ***cleared*** to a high state and the output enable (**nSoe**) is asserted. The external memory must then put the data on the bus (**Sdata**) within a prescribed set-up time before the next falling edge of the system clock.

The address and chip selects behave similarly in a write transaction. However, the output enable is turned off at the start of the transaction. The write enables and the data to be written to the memory appear shortly after the first rising edge of the clock. After the end of the cycle the write enables are ***cleared*** first to avoid any error in the transaction. The data, address and chip selects are then allowed to change for the next cycle.

During sequential read accesses to the same memory area, the signals **nScs**[0-3] and **nSoe** will remain permanently enabled. The new access is indicated solely by a change of address. During sequential write access to the same memory area, the signals nScs[0:3] will remain permanently enabled.

3.3.3 Wait State Insertion

While maximum system operating bandwidth is achieved when each memory or peripheral reference is completed in one clock cycle, it is often the case that wait states must be inserted to match the microcontroller speed to the timing parameters of the memory or peripheral being referenced. This may be the access time of the device, or some other timing parameter, such as data bus turn off time.

3.3.3.1 Automatic Wait State Generation

In most cases, the relationship between the clock period and the memory timing parameter is well understood. The MPC has a programmable wait state generator that can be set-up to introduce a number of wait states into each memory access. Three parameters are defined for reads and three for writes:

1. the number of Start-wait states required for the first accesses to a memory or peripheral device, or **start write waits (CRn[31:28])** and **start read waits(CRn[19:16])**.
   Note: **CRn** refers to the MPC **Configuration Register** n, where n is between 1 and 4.
2. the number of wait states required for the first and subsequent accesses to a memory or peripheral device, or **access write waits (CRn[27:24])** and **access read waits (CRn[15:12])**.
3. the number of Stop-wait states for which the data bus must be idle after the last access to a memory or peripheral device, or **stop write waits (CRn[23:20])** and **stop read waits (CRn[11:8])**.

Start-wait cycles are used to interface to devices with long address setup time requirements. The four Start-wait bits (separate bits for read and write) define the number of wait cycles which are inserted before the MPC asserts the read or write strobe for the current access. Start-wait cycles apply to both single cycle accesses and accesses which form part of a byte packing (write) or word building (read) sequence. These register bits must be loaded with zero if the Wait Control bit is Low.

Access-wait cycles are used to interface to devices with large access time / output delay requirements. The four Access-wait bits (separate bits for read and write) define the number of wait cycles which are inserted whilst the MPC asserts the read or write strobe for the current access. Access-wait cycles apply to both single cycle accesses and accesses which form part of a byte packing (write) or word building (read) sequence. These register bits must be loaded with zero if the Wait Control bit is Low.

The four Stop-wait bits are used by the MPC to hold off the next Firefly MF1 Core internal bus access. For write cycles the Stop-wait cycles are used to allow for external devices with large address and/or data hold time requirements. For read cycles, the Stop-wait cycles are used to allow slow peripherals to get off the external bus before the start of the next external access. If the operation in the clock cycle following the read transaction makes no reference to external memory, then the operation can be performed in parallel with the required stop-state. Therefore, in normal operation the device is only stalled during Stop-wait states if the next access is also to external memory. This reduces the number of wait states actually incurred.

Note that Stop-wait cycles are inserted during byte packing (write) sequences but are not inserted during word building (read) sequences (except for the last access) since during these sequences the external device is not required to vacate the external bus. These register bits must be loaded with zero if the Wait Control bit is low.

The application programmer must calculate the number of Start-, Access- or Stop-wait states required for the particular memory or peripheral connected to that memory area. An example of the effect of Wait- and Stop-states is shown in Figure 8, "Effect Of Start And Stop-states On A Read Access" below. This figure shows a single wait state, combined with a single Stop-wait state. The effect of this is that the memory access in total takes three clock cycles.

During diagnostic broadcast, it is sometimes necessary to incur stop-states on the complete device, instead of the external bus exclusively, as the broadcast information corresponding to the internal bus cycle would be masked by the stop-state on the external bus. In this case, the internal cycle would take place during the cycle following the stop-state, and the diagnostic information will be broadcast on the external bus, ensuring no loss of information. Refer to section 8.2, “Diagnostic Broadcast” on page 105.

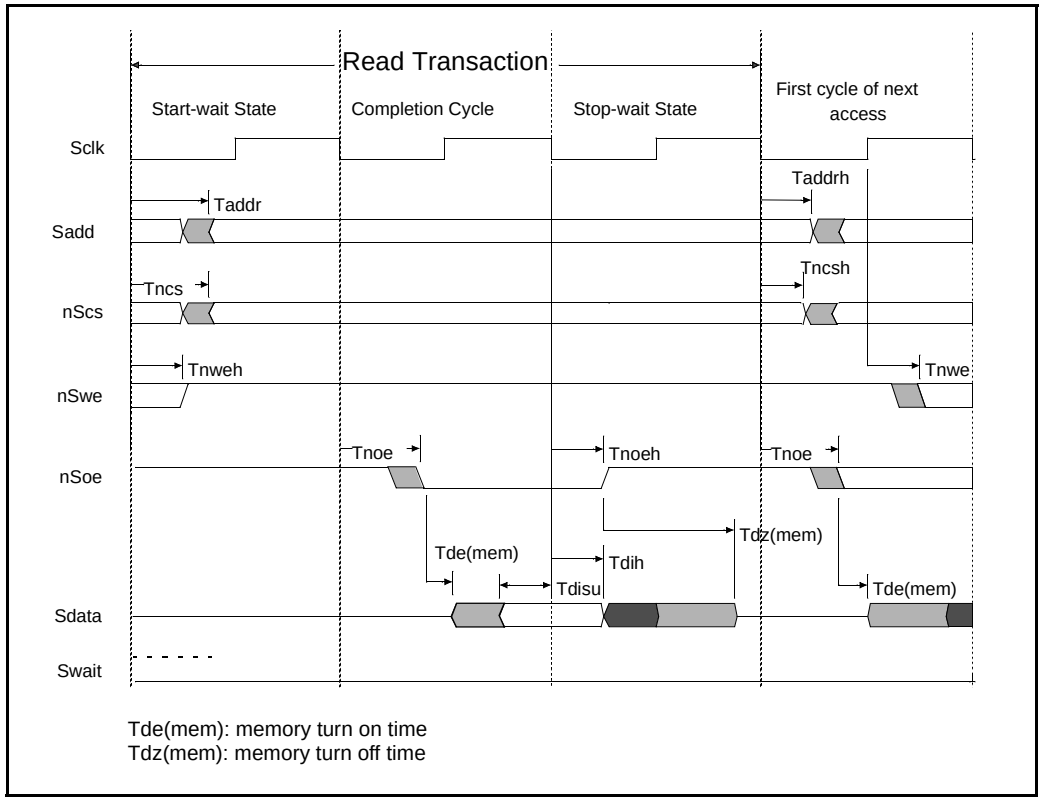


Figure 8 - Effect Of Start And Stop-states On A Read Access

NOTE: This diagram assumes that the MPC has been connected to standard I/O cells in order to create the signals shown. See Figure 17, "Example System Configuration (Little Endian)" on page 58 for an example.

During read transactions **Sadd** and **nScs** are maintained at the same level throughout any Start-, Access- or Stop-wait states. During any Access-wait states the output enable (**nSoe**) signal is maintained active (i.e., low); however, during any Start- or Stop-wait states, the **nSoe** signal is ***inactive*** (i.e., high.) Write transactions behave in a similar manner with respect to wait states, except that **nSwe** becomes active instead of **nSoe.**

3.3.3.2 External Wait-State Generation

In some cases, it cannot be determined at initialization how many clock cycles are required to access a particular memory-mapped device. This is particularly true of bridging devices (which give access to another bus) where access latency is dependent on the current conditions on the remote bus. In this case, wait states must be applied externally, by the assertion of the **Swait** signal.

If the external wait state insertion signal is to be used, the **Configuration Register (CR)** corresponding to that memory-mapped device must be programmed for external wait states; **Wait Control (CRn[5])** must be ***clear***. An initial default wait state is inserted into the transaction when external wait states are indicated. **Swait** is used to indicate how many additional wait states should be inserted.

Swait must initially be setup to the falling edge of **Sclk**, at the end of the first clock cycle (i.e., at the end of the initial default wait state cycle.) On subsequent clock cycles, **Swait** will be sampled on the falling edge of the clock. While it is sampled as high, the system will keep inserting wait states into the transaction. When the transaction is to complete, **Swait** must be ***cleared***. This will be sampled on the next falling edge, and will cause the subsequent cycle to become the completion cycle.

Figure 9, "MPC Externally Generated Wait State Insertion" on page 42 shows the timing required for **Swait** during a *read* transaction. The control signals (**Sadd**, **nScs**, **nSoe** and **nSwe**) are maintained at the same level throughout the transaction until the end of the completion cycle. Write transactions behave in a similar manner, except that **nSwe** becomes active after the first rising edge of **Sclk** in the transaction, as described in section 3.3.2, “Signal Relationships” on page 39. It then remains active until the end of the completion cycle.

Note: **Swait** is sampled for each transaction; therefore if, for example, a 32-bit read is requested from an area that is configured for 8 bits, then an **Swait** signal should be supplied for each of the four bytes read.

NOTE: This diagram assumes that the MPC has been connected to standard I/O cells in order to create the signals shown. See Figure 17 -, “Example System Configuration (Little Endian)” on page 58 for an example.

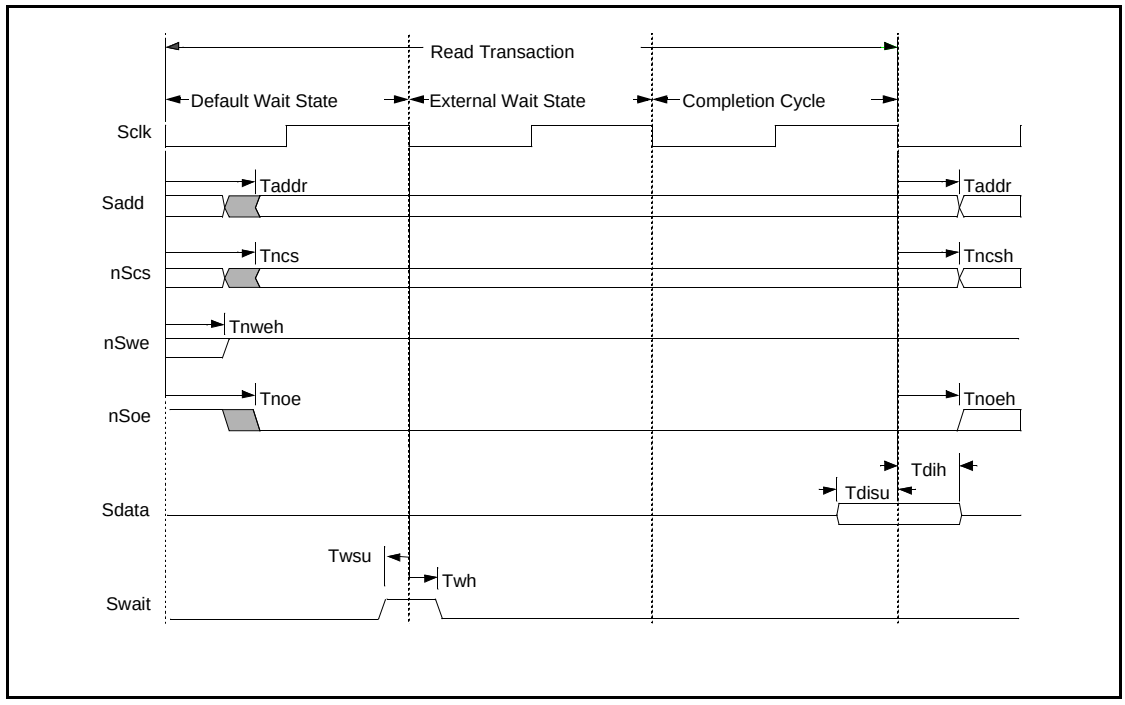


Figure 9 - MPC Externally Generated Wait State Insertion

3.3.4 Instruction Fetches From Memory

Since the MF1 core contains no internal memory, all processor instructions must be fetched from external memory. The ARM7TDMI™ processor has a 32-bit ARM® instruction set and a 16-bit Thumb® instruction set; therefore, 32-bit or 16-bit instructions must be fetched from the external memory sub-system.

However, it is not a requirement that the external instruction memory be configured to the same width as the instruction set being employed, and it may be desirable from cost or size considerations to configure the memory as 32-bit, 16-bit or 8-bit memory. In these cases, the MPC must reconstruct the ARM or Thumb® instructions from the data fetched from the memory subsystem. This reconstruction is performed by the dynamic data path sub-block within the MPC (see Figure 6, "MPC Functional Blocks And System Interconnections" on page 38).

The MPC does this by performing multiple read accesses to the memory system, reconstructing the instruction in an internal buffer. When the instruction is complete, the MPC signals that the instruction fetch is over and passes the complete instruction back to the processor. Hence, fetching an instruction from 16-bit memory will require two memory accesses, and from 8-bit memory will require 4 memory accesses for ARM instructions. Thumb instructions require only one and two memory accesses respectively.

Note that each read access to the memory sub-system may take one or more clock cycles, depending on the characteristics of the memory in use, and the way it has been configured in the MPC **Configuration Register**. The diagrams in Figure 10, "Instruction Fetch From 32-bit Memory (One Wait State)" below, Figure 11, "Instruction Fetch From 16-bit Memory (One Wait State)" on page 44 and Figure 12, "Instruction Fetch From 8-bit Memory (One Wait State)" on page 44 show instruction fetches from a memory system requiring one wait state for each memory access. In this example, if **Mode 0** is selected then **Access wait (CRn[31:28])** should be set to **[0001]**.

3.3.4.1 ARM® Instruction Fetches From 32-bit Memory

In this case, each instruction is fetched in its entirety. Only one memory access is required per instruction. **Data Size (CR[1:0])** must be programmed to **[10]** in the appropriate **Configuration Register**.

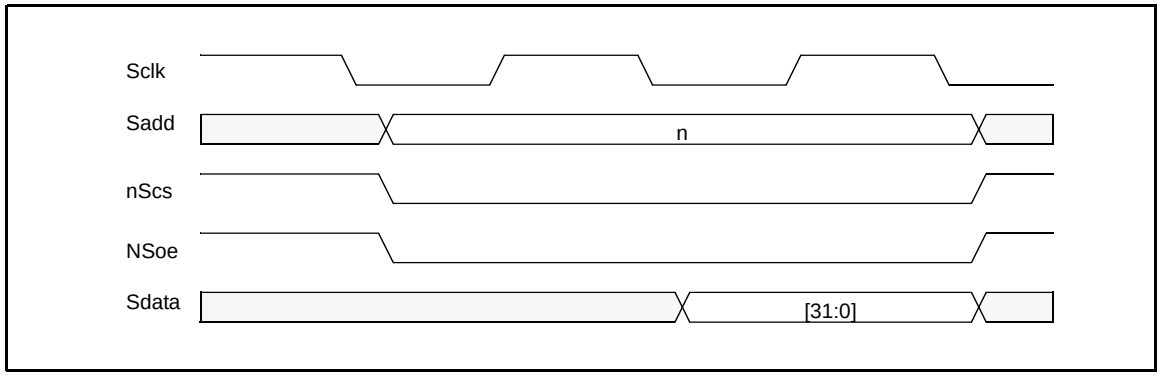


Figure 10 - Instruction Fetch From 32-bit Memory (One Wait State)

3.3.4.2 ARM® Instruction Fetches From 16-bit Memory

In this case, each instruction is fetched in two halves. Each half is fetched in consecutive memory accesses, lowest 16-bits first. **Data Size (CR[1:0])** must be programmed to **[01]** in the appropriate **Configuration Register.**

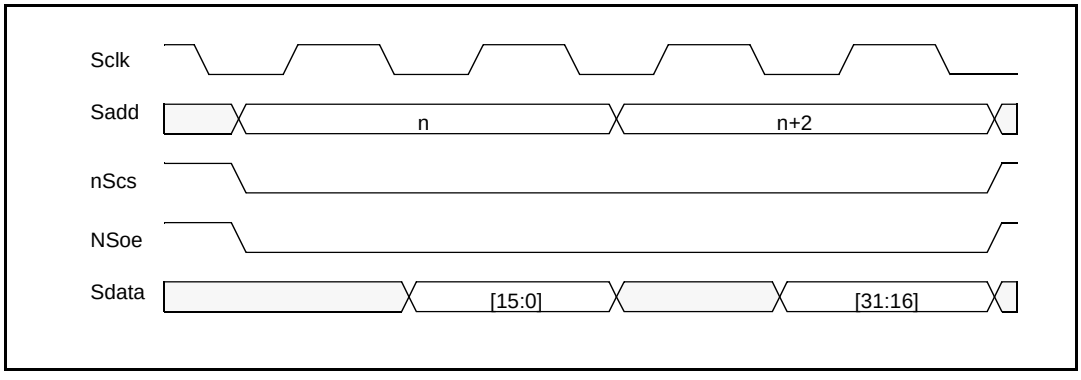


Figure 11 - Instruction Fetch From 16-bit Memory (One Wait State)

3.3.4.3 ARM® Instruction Fetches From 8-bit Memory

Here each instruction is fetched one byte at a time. Each byte is fetched in consecutive memory accesses, lowest order byte first. **Data Size (CR[1:0])** must be programmed to **[00]** in the appropriate **Configuration Register.**

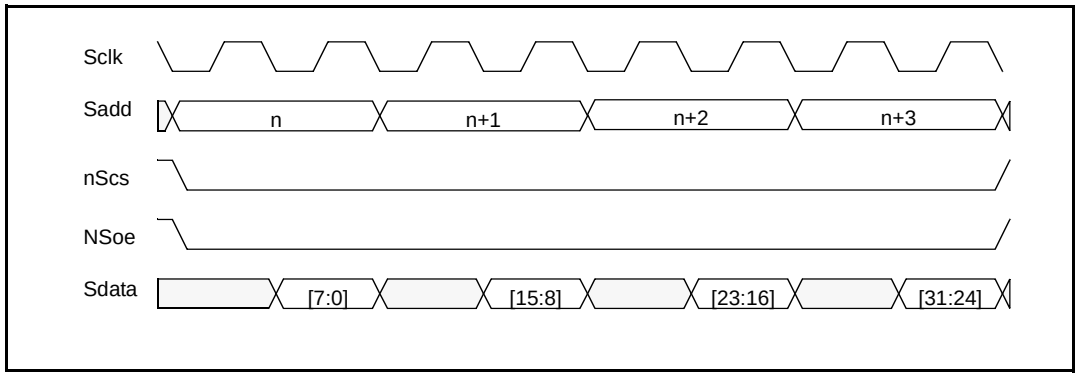


Figure 12 - Instruction Fetch From 8-bit Memory (One Wait State)

Since instructions are only fetched from memory, **Access Type (CRn[2])** should be set. If the associated memory area only contains instructions and constants, and the variable data is maintained in a separate memory area, then **Read Only Status (CRn[4])** may be set. Illegal writes to this memory area will then cause a data abort exception, as described in the **ARM7TDMI™ Technical Reference Manual (Rev 4),** available from ARM Ltd at http://www.arm.com.

3.3.5 Data Transfers To And From Memory

While instructions are defined to be 32-bit operands for ARM® instructions, and 16-bit operands for Thumb® instructions, data operands may be either 8-bit, 16-bit or 32-bit in size. Similarly, the external memory or peripheral device may be either 8-bit, 16-bit or 32-bit in size. Some adaptation of the read or write data must therefore be done to translate the operand size into the memory or peripheral size. The mechanism employed depends on whether a read or a write operation is being performed.

3.3.5.1 Data Read From Memory

If the external memory size is greater or equal to the operand size, then the operand may be read from memory in a single memory access. However, if the external memory size is smaller than the operand size, then operand packing is required.

Operand packing describes the process in which the MPC performs multiple bus accesses to an external memory device to build up the required quantity of data for the requested transaction (using **Data Size (CRn[1:0]**). This is done in a similar manner to the way that 32-bit instructions are built up, as described in section 3.3.4, “Instruction Fetches From Memory” on page 43.

For example the MPC would carry out four 8-bit accesses to an 8-bit external memory in order to read a 32-bit data operand. In a similar manner the MPC would perform 2 accesses for 16-bit external memory.

The multiple read accesses required by operand packing rely on the external device residing in contiguous memory space. The device must be capable of handling the address bus changing (for loading the relevant bytes or half words), while **nSoe** and its chip select line remains active.

When the operand size requested is smaller than the memory data width, the MPC reads the entire memory word, ignoring the bits that are not required. The way in which read accesses are broken down is shown in Table 20, “Data Read From Memory” .

Operand Size	**Memory Size**		
	8-bit	**16-bit**	**32-bit**
8-bit	Single 8-bit read	Single 16-bit read, (top 8 bits ignored)	Single 32-bit read (top 24 bits ignored)
16-bit	Two consecutive 8-bit reads, packed to form16-bit value.	Single 16-bit read	Single 32-bit read (top 16 bits ignored)
32-bit	Four consecutive 8-bit reads, packed to form 32-bit value.	Two consecutive 16-bit reads, packed to form 32-bit value	Single 32-bit read

Table 20 - Data Read From Memory

3.3.5.2 Data Write To Memory

Data writes are handled in a similar manner to data reads. Hence, a 32-bit write to an 8-bit memory will be broken down into four consecutive 8-bit writes. However, when the operand size is smaller than the memory width, it may not be possible to perform the requested transaction. This is because the memory may be incapable of accepting a write to less than the full memory width. In this case, the **Sub Memory Write** bit **(CRn[3])** must be ***cleared***. Sub memory width writes to this memory area will then cause a data abort exception.

If the memory is capable of accepting sub memory width writes, the **Sub Memory Write** bit **(CRn[3])** should be ***set***. The MPC uses separate write enable signals (**nSwe**[3:0]) to address each byte of the memory word. Doing so allows the MPC to perform 8- or 16-bit accesses to 32-bit wide external memories, and 8-bit accesses to 16-bit wide external memories, as illustrated below in Figure 13, "Writing to Individual Bytes of Memory".

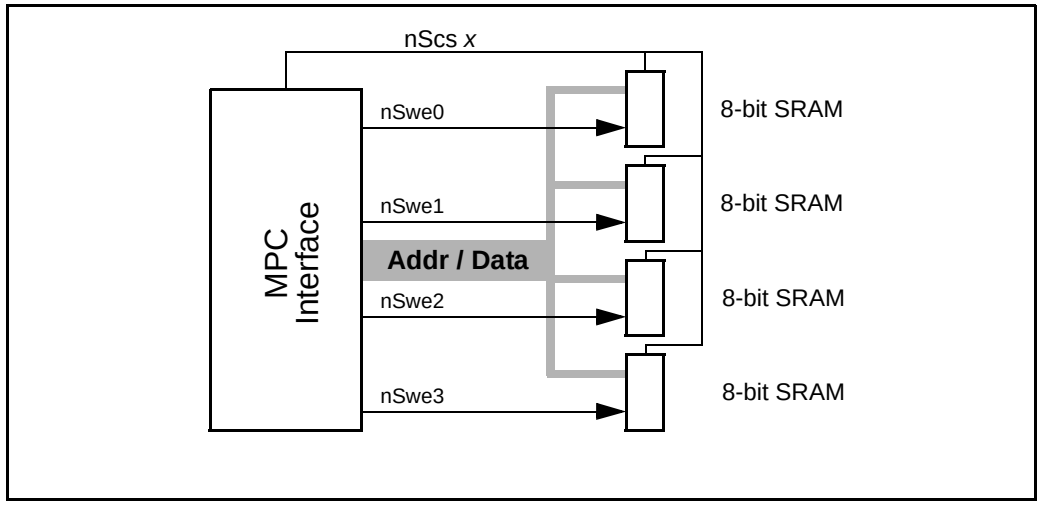


Figure 13 - Writing to Individual Bytes of Memory

The way the MPC writes to various widths of memory is summarized in Table 21, “Data Write To Memory” below.

Operand Size	Memory Size		
	8-bit	16-bit	32-bit
8-bit	Single 8-bit write	Single 8-bit write (if the memory can accept sub-width writes)	
16-bit	Two consecutive 8-bit writes	Single 16-bit write	Single 16-bit write (if the memory can accept sub-width writes)
32-bit	Four consecutive 8-bit writes	Two consecutive 16-bit writes	Single 32-bit write

Table 21 - Data Write To Memory

3.3.6 Endian Configuration

The MPC treats words in memory as being stored in Little Endian format

In the Little Endian scheme, the lowest numbered byte in a word is considered to be the least significant byte of the word and the highest numbered byte is the most significant. Byte 0 of the memory system should be connected to data lines 7 through 0 in this scheme.

Little Endian

Higher Address	31 24	23 16	15 8	7 0	Word Address
	11	10	9	8	8
	7	6	5	4	4
	3	2	1	0	0
Lower Address					

Table 22 - Little Endian Addresses Of Bytes Within Words

3.3.7 Access To Non-aligned Memory Addresses

Non-aligned accesses are defined as accesses where the address is not an integer multiple of the data size in words. For example, if x is a multiple of four, a 32-bit access to address x is aligned, whereas a 32-bit access to address x+1 is non-aligned.

The MPC does not handle non-aligned memory accesses. If an access is made to a non-aligned address, the MPC will propagate the *aligned* address to the external bus. Any byte alignment is then performed by the processor.

Using the previous example, if a 32-bit access is requested from address x+1, the MPC will return the aligned data starting from address x. The same response will be obtained by a 32-bit access to address x+2 and x+3. When the address is x+4, this is aligned again, and the MPC returns the data starting from address x+4.

Table 23, “Non-aligned Address Accesses” below summarises how the MPC propagates aligned addresses.

Address Offset	Data BytesReturned		
	8-bit Access	16-bit Access	32-bit Access
0	byte 0	bytes 0, 1	bytes 0, 1, 2, 3
+1	byte1		
+2	byte 2	bytes 2, 3	
+3	byte 3		
+4	byte 4	bytes 4, 5	bytes 4, 5, 6, 7
+5	byte 5		
+6	byte 6	bytes 6, 7	
+7	byte 7		

Table 23 - Non-aligned Address Accesses

3.3.8 Access To 16-bit Devices With Byte Selection Pins

Certain 16-bit SRAMS have upper (BHE) and lower (BLE) byte enable pins that allow byte accesses to be performed even though they are essentially 16-bit devices. Usually, to make a 16-bit access to such a device, both enables need to be taken active, whereas byte accesses require only one enable to be active. Such memories typically have *single* write enable (WE), chip select (CS) and output enable (OE) pins.

In order to accommodate such devices, the **Access Sub Memory Type** bits **(CRn[3:2])** and **Data Size** bits **(CRn[1:0])** should be configured to select a 16-bit memory device with 8-bit writes (byte select device.)

The MPC will now select the upper byte by activating **x_nub_out** and the lower byte by NLB, which is actually internally multiplexed with **x_addr_out**(0).

Connection of the device should therefore be: device WE to **x_nwe_out**(1), device upper byte select (BHE) to **x_nub_out**, device lower byte select (BLE) to **x_addr_out**(0). CS and OE should be connected as normal.

Byte accesses and 16 bit accesses are illustrated in <Italic (Body)>Table 14, “16-bit Accesses To 16-bit Byte Selectable RAM” on <italic>page 49, <Italic (Body)>Table 15, “Byte Read Accesses To 16-bit Byte Selectable RAM” on <italic>page 49, and <Italic (Body)>Table 16, “Byte Write Accesses To 16-bit Byte Selectable RAM” on <italic>page 50.

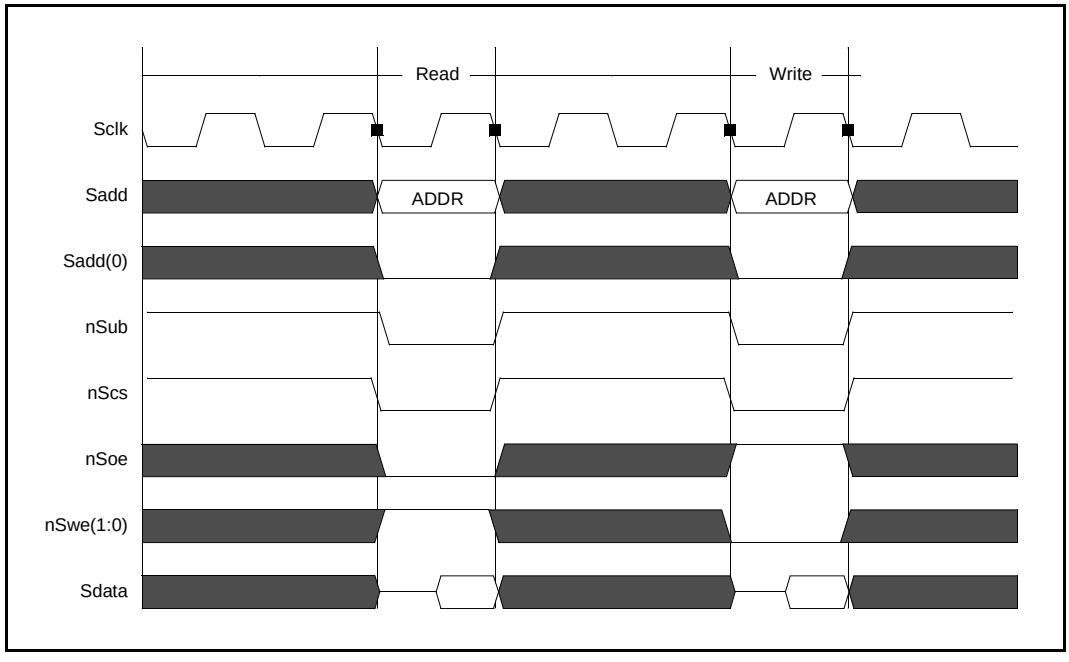


Figure 14 - 16-bit Accesses To 16-bit Byte Selectable RAM

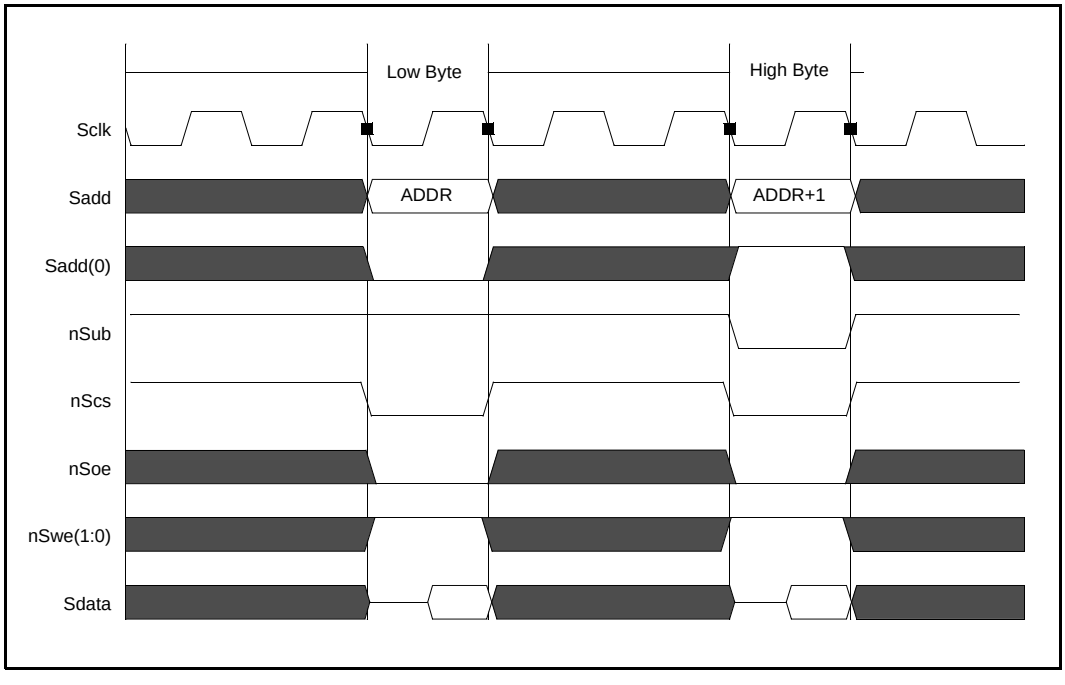


Figure 15 - Byte Read Accesses To 16-bit Byte Selectable RAM

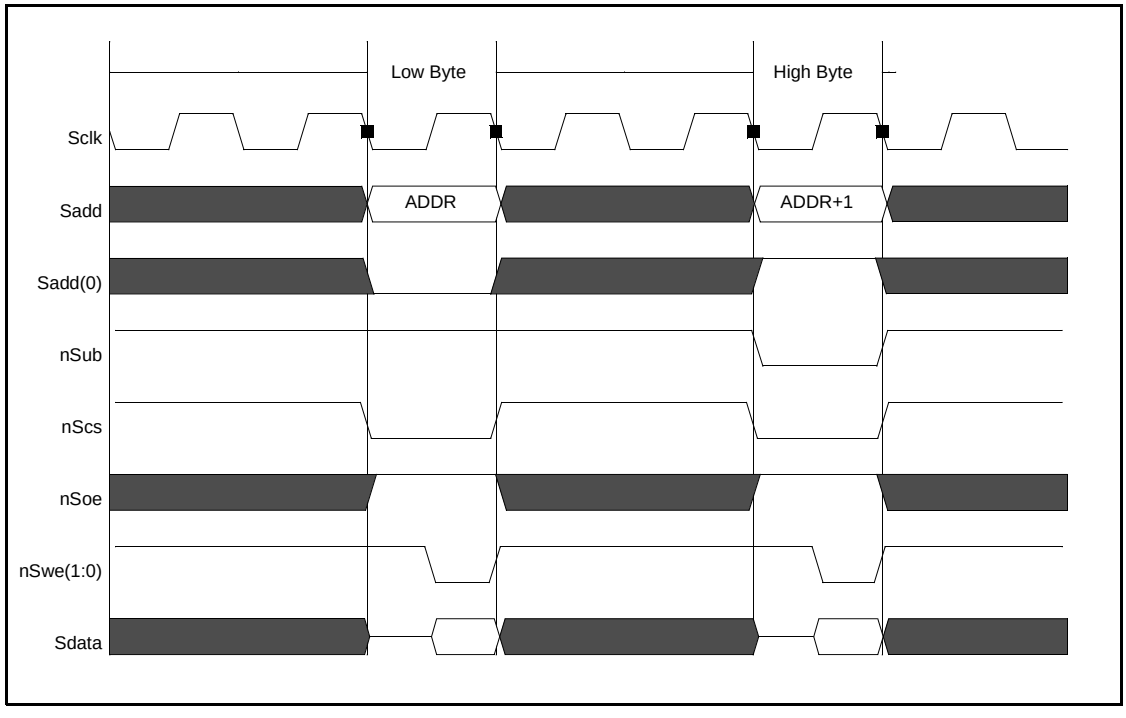


Figure 16 - Byte Write Accesses To 16-bit Byte Selectable RAM

3.4 Programmer's Model

The external memory areas for which the chip select signals **nScs**[3:0] become active are listed in Table 24, "MPC nScs External Decode Map" below:

Address (Hexadecimal)	Active Chip Select	External Memory Area
0000 0000 --> 1FFF FFFF	**nScs**[0]/**nScs**[3]*	Area 1
2000 0000 --> 3FFF FFFF	**nScs**[1]	Area 2
4000 0000 --> 5FFF FFFF	**nScs**[2]	Area 3
6000 0000 --> 7FFF FFFF	**nScs**[3]/**nScs**[0]*	Area 4
* When **SWAP** = 1 in **System Configuration Register** (See section 2.7, "System Configuration" on page 23)		

Table 24 - MPC nScs External Decode Map

3.4.1 Register Map

Table 25, "MPC Register Map" displays the register views for this module. Addresses are specified as an offset from the system-defined base address. All the registers are accessed as 32-bit registers, with unused bits defined as zero for read operations. Attempts to access any register with a half-word or byte transfer result in a bus error, as do any accesses to reserved registers or writes to read-only registers. The "Side Effect" column is used to note cases where a Read or Write modifies the contents of a register.

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
AREA 1						
+000	Configuration (Area 1)	32	FF00003#	RW		# = 4,5, or 6 depending on boot code memory requirements
AREA 2						
+004	Configuration (Area 2)	32	FF00003#	RW		# = 4,5, or 6 depending on boot code memory requirements
AREA 3						
+008	Configuration (Area 3)	32	FF000034	RW		
AREA 4						
+00C	Configuration (Area 4)	32	FF000034	RW		

Table 25 - MPC Register Map

1. An R in this column indicates that a read is destructive, and a W, that a write is destructive.

3.4.2 Register Details

Each of the four **Configuration** registers can be configured into 2 distinct modes, whose functions are determined by the contents of the **Configuration** bit-field. Table 26, “Mode 0: Butterfly Compatible Mode” below gives details of the bit-field significances when in the Butterfly compatible mode. Note that, in Table 26 below and <Italic (Body)>Table 28, “Mode 1 (Standard Mode)” on <italic>page 53, the DR (Destructive Read) column denotes whether or not the value in a register is reset by a **Read**.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:28	Access-waits	Number of wait states required for the each read access to a memory or peripheral device. [0y0000 – 0y1111]	0y1111	RW	
27:24	Stop-waits	Number of Stop-wait states for which the data bus must be idle after the last read bus access to a peripheral or device. [0y0000 – 0y1111]	0y1111	RW	
23: 8	Reserved		0		

Table 26 - Mode 0: Butterfly Compatible Mode

Bit	Function	Description	Reset Value	Type	Volatile[1]
7: 6	Configuration Mode	0y00 and 0y01 are the only legal values for this bit-field. Must be set to 0y00 to be in Mode 0; [0y00] Zero [0y01] One.	0	RW	
5	Wait Control	Indicates wait state control source of the current access. [0] External: controlled by Swait signal [1] Internal: controlled by internal wait state generator	1	RW	
4	Read Only Status	When set, indicates that only read accesses can be performed on the external address space. [0] Clear [1] Set	1	RW	
3	Sub Memory Write Status	Enable or disable based on whether the external memory is capable of receiving sub-memory width writes. [0] Disable [1] Enable	0	RW	
2	Access Type	Indicates type of access accepted by the external address space. [0] Peripheral: operand size must equal the memory size. [1] Memory: transaction may be constructed from multiple memory accesses i.e. operand size does not have to equal memory size.	1	RW	
1:0	Data Size	Indicates size of external data bus. Valid options are 8-, 16- and 32-bits. [0y00] Byte [0y01] Half Word [0y10] Word [0y11] Conditional Word: If Sub Memory Write Status is enabled, setting this value affects access type.	0y00	RW	

Table 26 - Mode 0: Butterfly Compatible Mode (continued)

1. An R in this column indicates that a read is destructive, and a W, that a write is destructive.

Table 27, “Mode 0 - Configuration Register Bit Actions” below gives a summary of the action for each of the combinations of *Data Size, Access type, and Sub Memory Write Status* in Mode 0.

Sub Memory Write Status	Access Type	Data Size bit 1	Data Size bit 0	Comments
X	0	0	0	8-bit peripheral
X	0	0	1	16-bit peripheral
X	0	1	0	32-bit peripheral
X	0	1	1	Illegal (reserved)
X	1	0	0	8-bit memory
0	1	0	1	16-bit memory with no 8-bit writes
1	1	0	1	16-bit memory with 8-bit writes
0	1	1	X	32-bit memory with no 8- or 16-bit writes
1	1	1	0	32-bit memory with 8- and 16-bit writes
1	1	1	1	32-bit memory with 16-bit writes

Table 27 - Mode 0 - Configuration Register Bit Actions

Table 28, “Mode 1 (Standard Mode)” below gives details of the bit-field significances when in the Standard mode.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:28	Start Write Waits	Numbers of Start-wait states required for each write access to a memory or peripheral device. [0y0000 – 0y1111]	0y1111	RW	
27:24	Access Write Waits	Number of wait states required for the first and subsequent write access to a memory or peripheral device. [0y0000 – 0y1111]	0y1111	RW	
23:20	Stop Write Waits	Number of Stop-wait states for which the data bus must be idle after the last write bus access to a peripheral or device. [0y0000 – 0y1111]	0y1111	RW	
19:16	Start Read Waits	Number of Start-wait states required for the each read access to a memory or peripheral device. [0y0000 – 0y1111]	0y1111	RW	

Table 28 - Mode 1 (Standard Mode)

Bit	Function	Description	Reset Value	Type	Volatile[1]
15:12	Access Read Waits	Number of wait states required for the each read access to a memory or peripheral device. [0y0000 – 0y1111]	0y1111	RW	
11: 8	Stop Read Waits	Number of Stop-wait states for which the data bus must be idle after the last read bus access to a peripheral or device. Must be zero if Wait Control bit is clear. [0y0000 – 0y1111]	0y1111	RW	
7: 6	Configuration	0y00 and 0y01 are the only legal values for this bit-field. Must be set to 0y01 to be in Mode 1; [0y00] Zero [0y01] One.	0	RW	
5	Wait Control	Indicates wait state control source of the current access. [0] External: controlled by Swait signal [1] Internal: controlled by internal wait state generator	1	RW	
4	Read Only Status	When set, indicates that only read accesses can be performed on the external address space. [0] Clear [1] Set	1	RW	
3:2	Access Sub Memory Type	Indicates access area device type and whether the external memory is capable of receiving sub-memory width writes. This field however cannot be viewed in isolation as there are limitations on supported combinations of Memory Type and Data size as outlined in <Italic (Body)>Table 29, “Mode 1 - Configuration Register Bit Actions” on <italic>page 55. [0y00] Peripheral: operand size must equal the memory size. [0y01] Memory: operand size must equal the memory size [0y10] Byte Select: 16-bit byte selectable memory [0y11] Sub Memory : transaction may be constructed from multiple memory accesses i.e. operand size does not have to equal memory size	1	RW	

Table 28 - Mode 1 (Standard Mode) (continued)

Bit	Function	Description	Reset Value	Type	Volatile[1]
1:0	Data Size	Indicates size of external data bus. Valid options are 8-, 16- and 32-bits. [0y00] Byte [0y01] Half Word [0y10] Word [0y11] Conditional Word.	0y00	RW	

Table 28 - Mode 1 (Standard Mode) (continued)

1. An R in this column indicates that a read is destructive, and a W, that a write is destructive.

The following table (Table 29) gives a summary of the action for each of the combinations of *Data Size and Access Sub Memory Type* in Mode 1.

Access Sub Memory Type [1:0]	Data Size [1:0]	Comments
00	00	8-bit peripheral device
01	00	8-bit memory device
10	00	Reserved for Future Use
11	00	Reserved for Future Use
00	01	16-bit peripheral device
01	01	16-bit memory device with no 8-bit writes
10	01	16-bit memory device with 8-bit writes (byte select device)
11	01	16-bit memory device with 8-bit writes (multiple write enables)
00	10	32-bit peripheral device
01	10	32-bit memory device with no 8- or 16-bit writes
10	10	Reserved for Future Use
11	10	32-bit memory device with 8- or 16-bit writes (multiple write enables)
00	11	Reserved for Future Use
01	11	Reserved for Future Use
10	11	Reserved for Future Use
11	11	32-bit memory device with 16-bit but no 8-bit writes (multiple write enables)

Table 29 - Mode 1 - Configuration Register Bit Actions

3.4.2.1 External Wait State Generation

If external wait state control is required, the MPC inserts a default single wait state into the system. Doing so allows the external memory system sufficient time to generate the **Swait** signal, which must be returned before the start of the next clock cycle. Therefore, it is not possible to perform single cycle memory access when the ***Wait Control*** bit is set to ***External***.

Note that all Wait-state bit-fields should be loaded with zero when the ***Wait Control*** bit is set to ***External***.

All Wait States are in '**n**' multiples of system clock (**Sclk**), where '**n**' is the value programmed into the appropriate bit-field.

3.5 External Interfaces

Table 30 below lists the off-core ports used by the MPC.

Port [Width]	Direction	Sense	Description
x_addr_out[21:0]	Output	-	Address Bus - 22 bits
x_addr_in[21:0]	Input	-	Address Bus - 22 bits
x_ncs_out[3:0]	Output	Active low	Memory area chip select
x_ncs_in[3:0]	Input	Active low	Memory area chip select
x_nwe_out[3:0]	Output	Active low	Byte Write Enable
x_nwe_in[3:0]	Input	Active low	Byte Write Enable
x_nub_out	Output	Active low	Upper Byte Select
x_nub_in	Input	Active low	Upper Byte Select
x_noe_out	Output	Active low	Output Enable
x_noe_in	Input	Active low	Output Enable
x_wait	Input	Active high	Wait State Request
reset_size[1:0]	Input	00 - 8-bit 01 - 16-bit 10 - 32-bit 11 - illegal	External Memory configuration - default size
x_data_out[31:0]	Output	-	Data Bus - 32 bits
x_data_in[31:0]	Input	-	Data Bus - 32 bits
x_tri_addr_op	Output	1 = Tri State	**x_add_out** tristate control
x_tri_ctrl_op	Output	1 = Tri State	**x_<control signals>** tristate control
x_tri_data_op	Output	1 = Tri State	**x_data_out** tristate control
x_tri_data_ip	Output	1 = Tri State	**x_data_in** tristate control

Table 30 - MPC Off-core Ports

3.6 Application Information: Designing A Memory System

3.6.1 Example System Configuration

An example memory system configuration is shown in Figure 17, "Example System Configuration (Little Endian)" overleaf. The figure shows a system that contains 8-bit wide EPROM in memory area 1, where the boot code for the microcontroller will reside. Note that the **reset_size**[1:0] bits have both been tied to ground, to indicate that area 1 will contain 8-bit wide memory. Area 2 contains 32-bit wide fast SRAM, made up of four 8-bit wide devices allowing writes to individual bytes. Area 3 contains a 16-bit peripheral and in this example, memory area 4 is unused.

External I/O cells have been added to create bidirectional external buses. Note that even though they are not used by the memory configuration, the input paths on the control and address signals are required for manufacturing test.

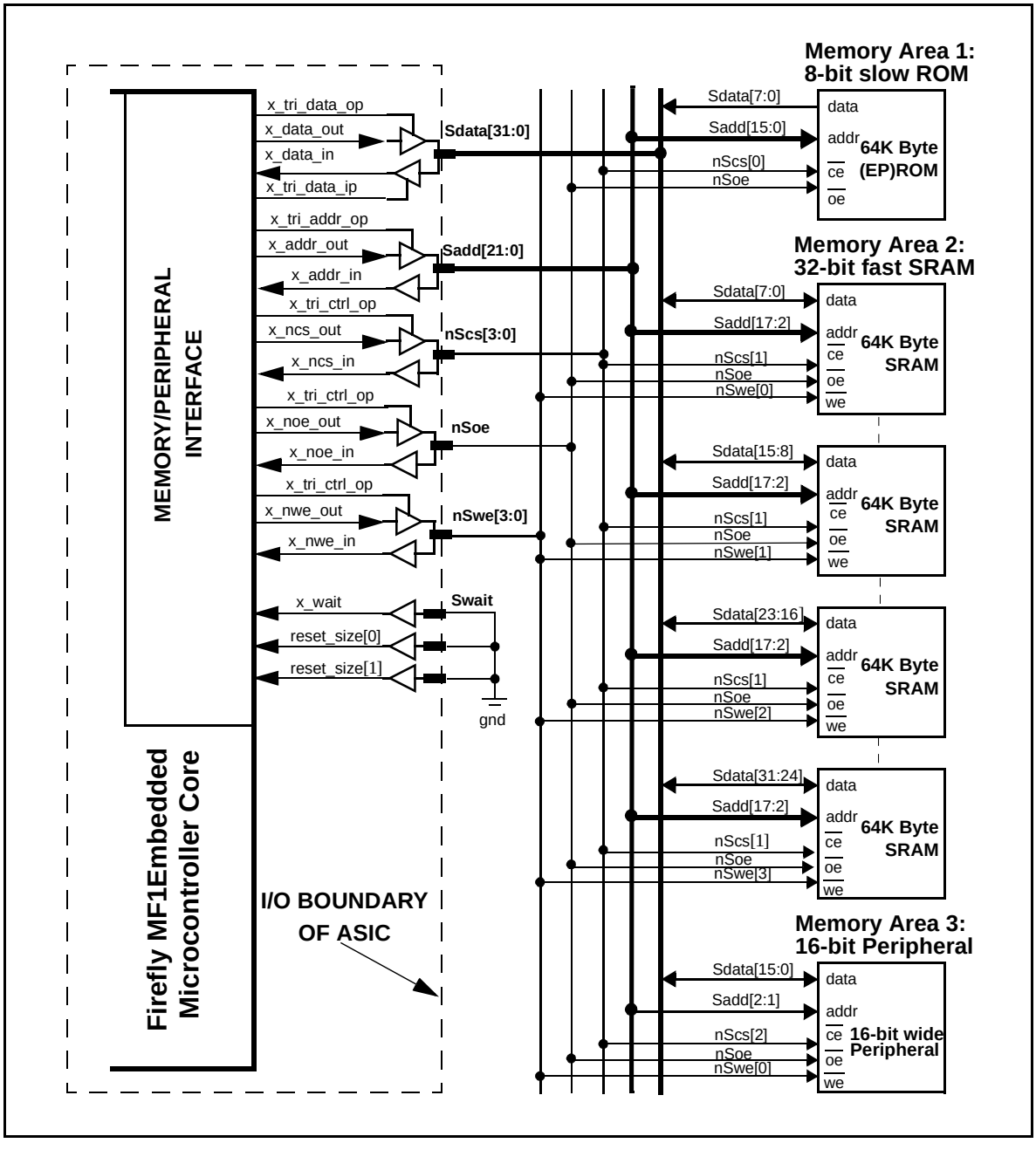


Figure 17 - Example System Configuration (Little Endian)

3.6.2 MPC Configuration Register Settings For The Example System Configuration

A typical EPROM to use in the example might be a 27C512, with 100ns access time. At an operating speed of 25MHz (40ns period), it would require 2 wait states to access this device. Similarly, the EPROM has a slow turn off time (typically 40ns); therefore, one Stop-state is required to prevent contention on the external data bus.

The **Wait Control** bit must be ***set*** to enable automatic wait state generation. The memory is read only, so the **read only** bit can also be ***set***. The **sub memory write** bit is redundant, since it is a ROM, and the **memory or peripheral** bit should be ***set*** to indicate memory. Finally the data width is 8 bits, so the **data size** bits should both be ***cleared***.

In memory area 2, the SRAM to be used is fast enough to be used at zero wait state, zero stop-state. Therefore both of these fields should be set to zero. Again the automatic wait state generator is being used, so **Wait Control** is ***set***. The memory is capable of accepting writes; therefore, the **read only** bit should be ***cleared***. The SRAM is capable of accepting both 8- and 16-bit writes, so the **sub memory write** bit is ***set***, along with the **memory or peripheral** bit. Finally the size is 32 bits, so **data size** is set to 1,0.

Memory area 3 contains a 16-bit peripheral. In this example the peripheral has a 200 ns access time, therefore 5 wait states will be required. Since the turn off time of its data drivers is 50 ns, two stop-states are required to prevent bus contention.

The automatic wait state generator is being used, so **Wait Control** is ***set***. The peripheral is capable of accepting writes, therefore the **read only** bit should be ***cleared***. Since peripherals cannot accept wrong-sized accesses, the **sub memory write** bit should also be ***cleared***, along with the **memory or peripheral** bit. Finally the size is 16 bits, so **data size** is set to 0,1.

The MPC **Configuration Register** settings are shown in Table 31, “Example MPC Configuration Register Settings” below.

Access-Waits	Stop-Waits	Reserved (16bits)	Config. Mode	Wait Control	Read Only	Sub Mem Write	Access Type	Data Size
Memory Area 1 Configuration: 8 -bit Slow EPROM								
0010	0001	0..0	00	1	1	0	1	00
Memory Area 2 Configuration: 32-bit Fast SRAM								
0000	0000	0..0	00	1	0	1	1	10
Memory Area 3 Configuration: 16-bit Peripheral								
0101	0010	0..0	00	1	0	0	0	10

Table 31 - Example MPC Configuration Register Settings

3.6.3 Calculating required memory timing parameters

The timings provided with the MPC describe the characteristics of the cell, but do not describe how to calculate what speed of RAM can be used in a system. For example, it is not obvious what speed of SRAM is required to run single-cycle accesses. This section explains how to calculate the common timing parameters for a memory sub-system.

The diagram in Figure 18, "Interfacing the MPC to SRAM" below shows a write transaction followed by a read transaction. The normal MPC timing parameters are shown in grey. The timing parameters required by a standard SRAM are shown in black. A description of the timing parameters for the MPC appears in <Italic (Body)>Table 33, “MPC Timing Parameters” on <italic>page 61, while <Italic (Body)>Table 32, “Typical SRAM Parameters (with formulae)” on <italic>page 60 contains a description of some typical SRAM timing parameters/constraints

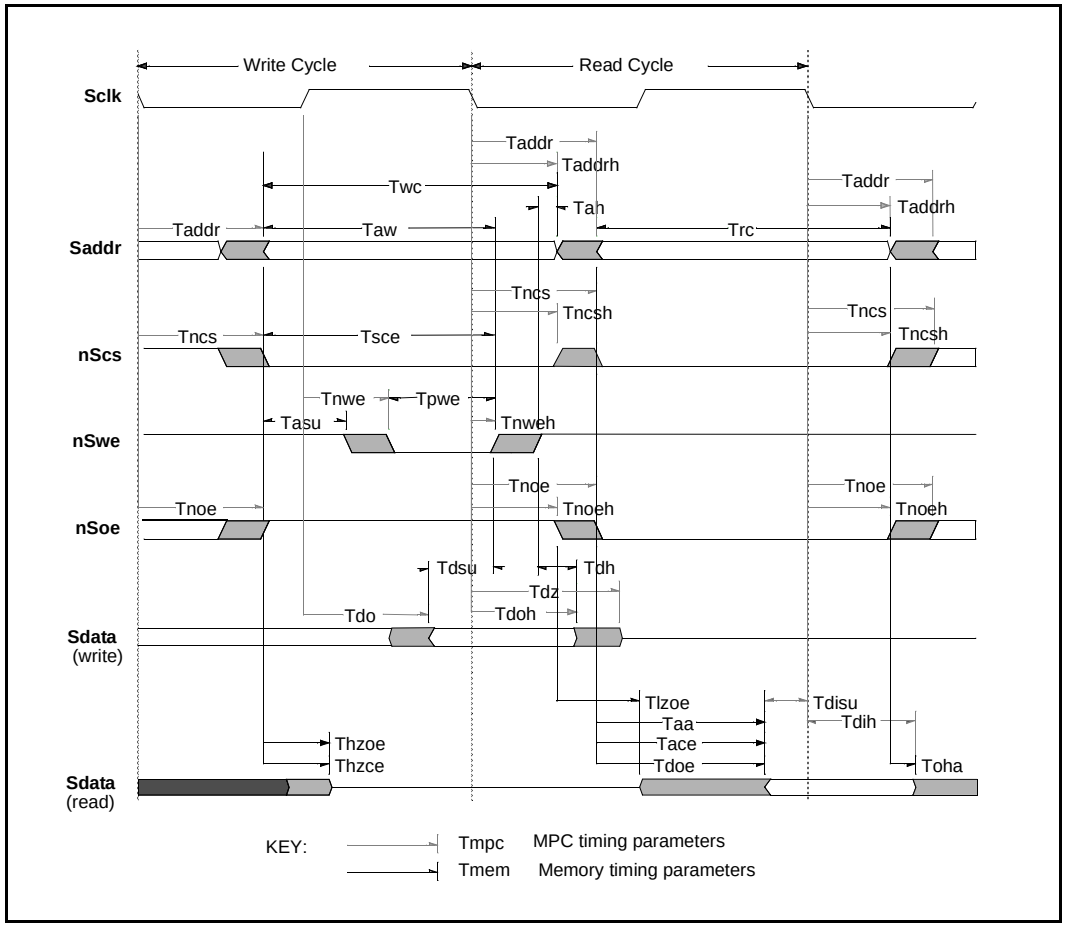


Figure 18 - Interfacing the MPC to SRAM

NOTE: This diagram assumes that the MPC has been connected to standard I/O cells in order to create the signals shown. See Figure 17 -, “Example System Configuration (Little Endian)” on page 58 for an example.

In order to calculate what speed of memory is required, there are connecting formulae that can be used to check that the timing requirements are all met. These appear in Table 32, “Typical SRAM Parameters (with formulae)” below. For example, to calculate the address access time needed on a read cycle, the formula is **Taa ð** n x (**Tclk**) **-** (**Taddr** + **Tdisu**)**.** This says that the memory access time **Taa** must be less than or equal to the number of clock periods, less the MPC address output delay, less the required input setup time on the MPC.

Name	Type	Description and Formula
Twc	constraint	Write cycle time FORMULA: **Twc <=** n x (**Tclk**) **-** (**Taddr - Taddrh**)
Tsce	constraint	CE low to write end FORMULA: **Tsce <=** n x (**Tclk**) **-** (**Tncs - Tnweh**)
Taw	constraint	Address set-up to write end FORMULA: **Taw <=** n x (**Tclk**) **-** (**Taddr - Tnweh**)
Tah	constraint	Address hold from write enable FORMULA: **Tah <= Taddrh - Tnweh**

Table 32 - Typical SRAM Parameters (with formulae)

Name	Type	Description and Formula
Tasu	constraint	Address setup to write start FORMULA: **Tasu <= Tclk/2 -** (**Taddr - Tnwe**)
Tpwe	constraint	Write enable pulse width FORMULA: **Tpwe <=** n x (**Tclk**) **-** (**Tclk/2 + Tnwe -Tnweh**)
Tdsu	constraint	Data input setup to write enable FORMULA: **Tdsu <=** n x (**Tclk**) **-** (**Tclk/2 + Tdo - Tnweh**)
Tdh	constraint	Data input hold from write enable FORMULA: **Tdh <= Tdoh - Tnweh**
Trc	constraint	Read cycle time FORMULA: **Trc <=** n x (**Tclk**) **-** (**Taddr - Taddrh**)
Taa	delay	Address access time to data valid FORMULA: **Taa <=** n x (**Tclk**) **-** (**Taddr +Tdisu**)
Toha	delay	Data hold after address change FORMULA: **Toha >= Tdih - Taddrh**
Tace	delay	CE low to data valid FORMULA: **Tace <=** n x (**Tclk**) **-** (**Tncs + Tdisu**)
Tdoe	delay	OE low to data valid FORMULA: **Tdoe <=** n x (**Tclk**) **-** (**Tnoe + Tdisu**)
Tlzoe	delay	Output enable to data drive FORMULA: **Tlzoe >= Tdz - Tnoe**
Thzoe	delay	Data output turn off from output enable FORMULA: **Thzoe <= Tclk/2 -** (**Tnoeh - Tdo**)
Thzce	delay	Data output turn off from chip enable FORMULA: **Thzce <= Tclk/2 -** (**Tncsh - Tdo**)

Table 32 - Typical SRAM Parameters (with formulae) (continued)

Throughout the equations, the parameter **Tclk** represents the clock period to be used in the application, and the parameter **n** represents the number of clock cycles to be taken for each memory access in the memory area under consideration; this is one more than the number of wait states, as programmed into the **Configuration Register** (i.e., a one wait state access takes two clock cycles).

The figures given in the cell data sheet are the timings to/from the periphery of the cell and DO NOT include any delays through I/O's. Table 33, "MPC Timing Parameters" below suggests how they should be adjusted to include I/O propagation timings.

Name	Type	Calculated Using Datasheet & I/O Cell Delay Value	Description
Tclk	-	-	System clock period in application (**Sclk**)
Taddr	delay	Tx_addr + op_delay	**Sclk** falling to address (**Sadd**) valid
Taddrh	delay	Tx_addrh + op_delay	address hold from **Sclk** falling

Table 33 - MPC Timing Parameters

Name	Type	Calculated Using Datasheet & I/O Cell Delay Value	Description
Tncs	delay	Tx_ncs + op_delay	**Sclk** falling to chip select (**nScs)** valid
Tncsh	delay	Tx_ncsh + op_delay	chip select hold from **Sclk** falling
Tnoe	delay	Tx_noe + op_delay	**Sclk** falling to output enable (**nSoe**) valid
Tnoeh	delay	Tx_noeh + op_delay	output enable hold from **Sclk** falling
Tnwe	delay	Tx_nwe + op_delay	**Sclk** falling to write enable (**nSwe**) valid
Tnweh	delay	Tx_nweh + op_delay	chip select hold from **Sclk** falling
Tdz	delay	Tx_data_tri + op_tristate	data output turn off time
Tdo	delay	Tx_do + op_delay	**Sclk** rising to data output valid
Tdoh	delay	Tx_doh + op_delay	Data output hold from **Sclk** falling
Tdisu	constraint	Tx_disu + ip_delay	required data input setup time
Tdih	constraint	Tx_disuh + ip_delay	required data input hold time

Table 33 - MPC Timing Parameters (continued)

4.0 Universal Asynchronous Receiver / Transmitter (UART)

4.1 Overview

The Universal Asynchronous Receiver Transmitter (UART) is a component that provides industry-standard levels of support for full-duplex asynchronous serial communications, with appropriate mechanisms for both software and hardware flow control. Standard modem handshake signals are incorporated to facilitate communications via intermediate devices in the communications channel. <italic>Figure 19, "UART Functional Block And System Interconnections" overleaf provides a block diagram of the UART and Table 34, “UART Port Descriptions” on page 64 describes the UART ports.

4.2 Design Features

- Full duplex operation, independent transmit and receive channels
- 7 or 8-bit serial data length
- 1 or 2 stop bits
- Even, odd or no parity generation
- Internal selectable baud rate generator, derived from system clock
- Double buffered transmit and receive channels
- Software polling to determine channel status
- Optional interrupt generation on transmit channel becoming empty
- Optional interrupt generation on receive channel becoming full
- Detection of parity, overrun, and framing errors on receive channel, with optional interrupt generation
- Support for modem signals: Request to Send **(RTS)**, Clear to Send **(CTS),** Data Set Ready **(DSR),** Data Terminal Ready **(DTR)**, Ring Indicator **(RI)** and Data Carrier Detect **(DCD)**, with edge detection and optional interrupt generation
- Maximum transmission rates corresponding to 1/16th of the system clock[1]
- Power saving through automatic clock suspension of the transmit and receive channel circuits when both are disabled
- Clock suspended to modem control section when modem signals are disabled
- Digital input filter to improve noise immunity

1. This corresponds to the line rate and NOT the Data bandwidth, which will be typically a maximum of 80% of this speed (owing to the use of Start and Stop bits).

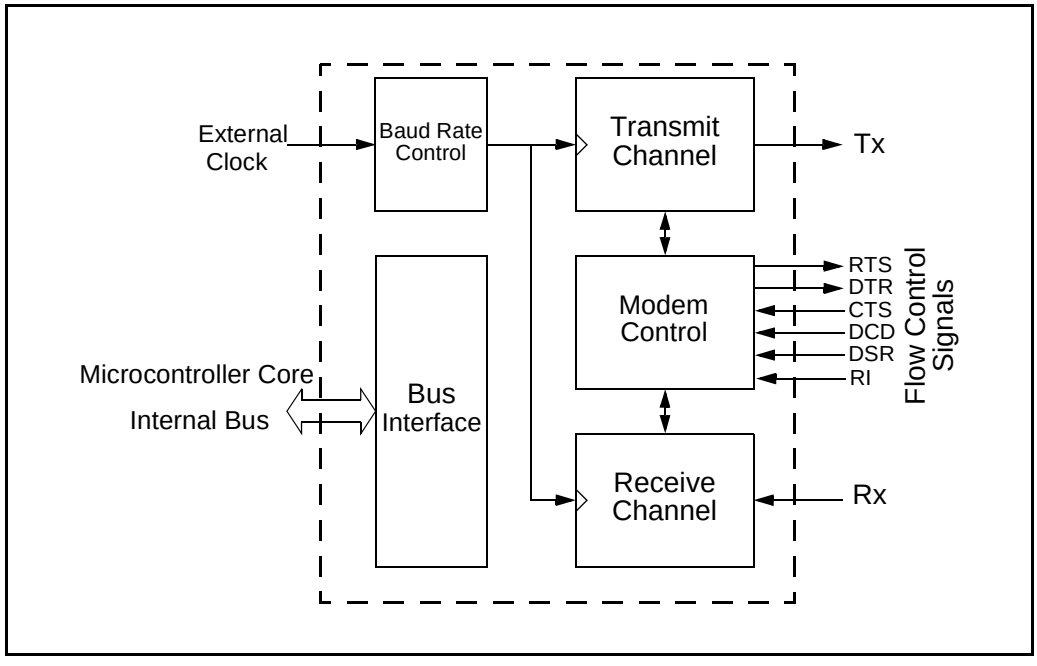


Figure 19 - UART Functional Block And System Interconnections

Port	Direction	Sense	Description
utxd	Data out		Serial transmit data: Idles high when no data is being transmitted
urxd	Data in		Serial receive data: Expects input signal to idle high when no data is being transferred
ucts	Input	active high	Clear To Send: When asserted, indicates that the modem or data set is ready to exchange data; the Modem Status Register monitors actual level and level changes. Optionally used as hardware handshake signal on either the transmit or receive channel
urts	Output	active high	Ready To Send: When asserted, indicates that the UART is ready to exchange data. Optionally used as hardware handshake signal on either transmit or receive channel; set from Modem Control Register
ueclk	Input		External Clock Ueclk is the external source for baud rate generator
udcd	Input	active high	Data Carrier Detect Input When asserted, Udcd indicates that the communications carrier is present.

Table 34 - UART Port Descriptions

Port	Direction	Sense	Description
udtr	output	active high	Data terminal ready Data Terminal ready modem signal.
udsr	input	active high	Data set ready Data Set ready modem signal.
ri	input	active high	Ring indicator Ring Indicator modem signal.

Table 34 - UART Port Descriptions (continued)

4.3 Operational Description

For register details, see section 4.4, "Programmer's Model" on page 71.

4.3.1 Baud Rate Generation

The UART transmit and receive channels are clocked from a single, locally derived clock (referred to below as the **internal clock**), whose period is determined by the reference clock, and the value programmed into the baud rate register. The reference clock is selected, using **Clock Source (CR[2])**, from one of two sources:

- the external clock input **ueclk**, or
- an internal prescaler driven by the system clock, **Sclk**.

In the latter case, a clock prescaler (16 bits long) is configured to generate a reference clock of period **'Sclk** divided by 1,2,4,... up to 32768', as specified by the value programmed into **Division Select (MR[7:4])**.

The **Baud Rate Register** is used to divide the reference clock period within the range 1 (**BRR** = 0) to 256 (**BRR** = 255) to allow approximation to the desired baud rate. The required baud rate register value may be calculated according to the following formula:

BRR = ((Reference Clock) / (16 x Baud Rate)) - 1

The clock is not applied to the transmit or receive channels if neither channel is enabled or currently active. Similarly, the system clock is not applied to the modem control section when the modem enable bit of the control register is reset

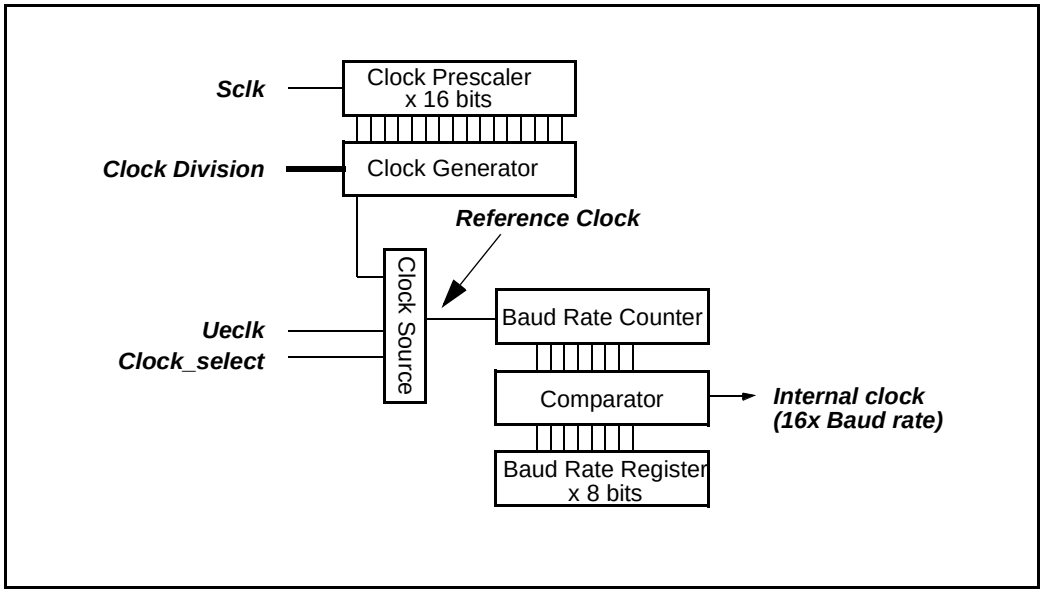


Figure 20 - The Clock Chain

4.3.2 Transmit Channel

The Transmit channel is clocked by the internal clock referenced in <italic>Figure 20, "The Clock Chain" above. At reset, the **Transmit Shift Register** and the **Transmit Register** are empty, **Transmit Register Status (SR[1])** is set, and the channel is in Idle state. If **Transmit Interrupt Enable (CR[5])** is subsequently set, a transmit interrupt request is generated.

The following set of conditions must be satisfied for data transmission to take place:

- **Transmit Channel Control (CR[1])** must be set and
- the **Transmit Register Status (SR[1])** must be clear (this occurs when a byte of data is written to the **Transmit Register**).

If the above conditions are met, then the following sequence of operations occurs:

- The contents of the **Transmit Register** are transferred into the **Transmit Shift Register**, a Start bit is output from the Transmit Data port, and **Transmit Register Status (SR[1])** is set, disabling new transfers until fresh data is provided. If **Transmit Interrupt Enable (CR[5])** is set, then an interrupt is generated. **Transmit Status (SR[3])** is also set to indicate that a transfer is taking place.
- The contents of the **Transmit Shift Register** are then shifted out from the Transmit Data port, low order bit first, at one 16th of the internal clock rate. The value of **Data Length (MR[0])** dictates the number of transmit bits per character; if set, 7 bits are transmitted, while if clear, 8 bits are transmitted.
- If **Parity Checking (MR[1])** is set, then a Parity bit is inserted into the serial data stream. If **Parity Sense (MR[2])** is set, then odd parity is generated; if clear, then even parity is generated.
- The number of Stop bits inserted into the data stream is indicated by **Number of Stop Bits (MR[3])**. If the bit is clear, then 1 Stop bit is inserted; if set then 2 Stop bits are inserted.

Upon completion of the transfer, the transmission condition described above is re-evaluated. If the evaluation is true, then the next character will be transmitted; otherwise, the channel returns to the idle state, and **Transmit Status (SR[3])** is cleared.

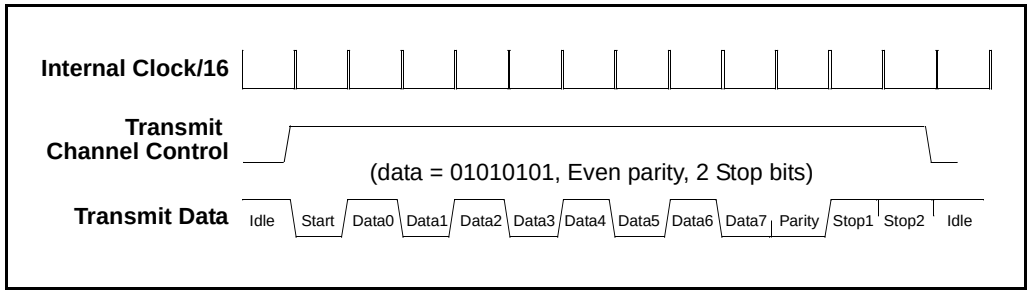


Figure 21 - Serial Transmission Example

4.3.3 Receive Channel

- Data transmission from the Transmit channel is illustrated in <italic>Figure 21, "Serial Transmission Example" above. The Receive channel is clocked at 16 times the desired channel baud rate. At reset, the **Receive Shift Register** and the **Receive Register** are both empty, **Receive Register Status (SR[0])** is clear, and the receive channel is disabled. The channel may be enabled by setting **Receive Channel Control (CR[0])**. The channel idles until the following conditions are met:
- **Receive Overrun Error (SR[6])** and **Framing Error Status (SR[5])** are clear, indicating the receive channel is free to receive new data,
- A transition from high to low is detected on the filtered receive data (**Urxd**) input, which signals the leading edge of a Start bit.

The detection of the leading edge of a Start bit causes the following actions:

- The Receive channel translates into the start state, resetting the x16 clock count to zero, and indicating that a Start bit has been detected by setting **Receive Status (SR[2])**.
- On the eighth clock cycle, the channel resamples for the Start bit, in order to reject spurious transitions. If the Start bit is not present, the channel returns to an idle state.
- Each subsequent 16 clocks, the receiver samples the filtered receive data input, and shifts the sampled bit into the **Receive Shift Register**, lowest significant bit first, until the number of bits indicated by **Data Length (MR[0])** have been assembled. If the data length is 7 bits, then the most significant bit is set to zero.
- The parity of the incoming data is calculated cumulatively with each bit. If **Parity Checking (MR[1])** is set, the result is compared with the sampled incoming Parity bit. If **Parity Sense (MR[2])** is clear, then even parity is expected; if set, then odd parity is expected. If a parity check error is detected, **Parity Error Status (SR[4])** is set.
- The incoming data stream is sampled for the correct number of Stop bits, as indicated by the value of **Number of Stop Bits (MR[3])** - if an expected Stop bit is absent, the **Framing Error Status (SR[5])** flag is set.
- **Receive Register Status (SR[0])** is then checked to determine the status of the receive buffer. If clear, the received data is transferred from the **Receive Shift Register** to the **Receive Register**, **Receive Register Status (SR[0])** is set, and sampling for a new character begins. If set, **Overrun Error (SR[6])** is set, and the incoming data discarded. The receive channel then should be disabled by writing a zero into the **Receive Channel Control (CR[0])** to avoid further sampling for a new start bit.

4.3.4 Receive Data Filter

To reduce the susceptibility of the UART receive channel to extraneous noise, the input signal is passed through a low-pass digital filter to smooth the received serial data stream. The digital filter is implemented as a 3-bit up/down counter (with no roll-over or roll-under) which is clocked at 16x the desired baud rate. The unfiltered serial input is applied to the up/down control of the counter, while the filtered serial output is taken from the most significant bit of the counter. The filter introduces a four clock (pi/4) phase delay between input and output signals, but can reject up to four incorrect samples per received bit. The function of the filter is illustrated below:

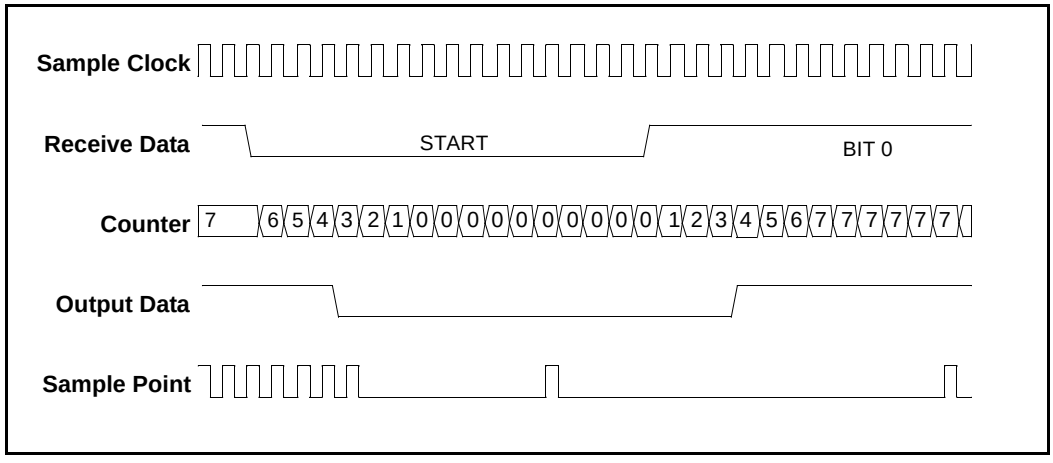


Figure 22 - Receive Data Filter Action

0.0.1 Data Transfer Methods

A variety of methods are available to move data to and from the UART, the selection of which is dictated by the nature of the transfer in terms of real time constraints and the software or hardware overhead to support it. When the **Receive Register** is full, or the **Transmit Register** is empty, the condition is indicated by associated status flags, **Receive Register Status (SR[0])** and **Transmit Register Status (SR[1])** respectively. The implications of each and their derived signals are described below:

4.3.4.1 Software Polled I/O

Software polled I/O is the simplest and quickest mechanism available, in which the processor constantly polls the UART **Status Register** to determine whether there is data to be transferred from the UART upon reception (as indicated by **Receive Register Status (SR[0])** = 1) or whether data can be transferred to the UART for transmission (indicated by **Transmit Register Status (SR[1])** = 1). **Receive Interrupt Enable (CR[4])** and **Transmit Interrupt Enable (CR[5])** should be disabled. This technique is dependent on the data rate not exceeding the worst case polling repetition rate, in which case data may be lost. Alternatively, if the data rate is significantly slower than the polling repetition rate, then unnecessary and hence inefficient polling will result.

4.3.4.2 Interrupt Driven I/O

In interrupt driven I/O, the processor is interrupted from executing its current task to service a transfer request from the UART. **Receive Interrupt Enable (CR[4])** and/or **Transmit Interrupt Enable (CR[5])** should be enabled, as appropriate. When either **Receive Register Status (SR[0])** or **Transmit Register Status (SR[1])** is set, a corresponding **Receive Interrupt** or **Transmit Interrupt** is generated, which is processed by the Interrupt Controller before being applied to the processor. Reading the **Receive Register** or writing to the **Transmit Register** will clear the corresponding status flag, and hence negate the interrupt. Using interrupt driven I/O assures prompt service of the UART channel, but there is some associated latency and software overhead.

4.3.4.3 Direct Memory Access

Since the interrupt generating condition may be removed by either reading the **Receive Register** or writing to the **Transmit Register** as appropriate, the interrupt service routine may be replaced by an operation which mimics the function of the interrupt service routine and transfers data between a memory buffer and the UART, and vice-versa; the internal DMA controller is capable of this function. **Receive Interrupt Enable (CR[4])** or **Transmit Interrupt Enable (CR[5])** should be enabled, as appropriate. When either **Receive Register Status (SR[0])** or **Transmit Register Status (SR[1])** is set, a corresponding **Receive Interrupt** or **Transmit Interrupt** is generated. These interrupts should be disabled in the Interrupt Controller if DMA is to be used, and the appropriate interrupt can be routed to the DMA request input (see Chapter 6.0, “DMA Controller (DMAC)” on page 82 for details).

For reception of data, the DMA source channel must be programmed with the UART **Receive Register** address (as a static address), and the DMA destination channel address should be set to the appropriate static or incrementing-address destination. See Chapter 6.0, “DMA Controller (DMAC)” on page 82 for detailed information about other set-up requirements.

With the DMA controller suitably programmed and enabled (using the software enable mechanism) data received by the UART will trigger the DMAC automatically. The DMAC will request the internal bus, and upon the bus being granted, will transfer the UART data to the required destination. This mechanism incurs no on-going software overhead or CPU activity and is highly efficient. Transmission of data can be handled in a similar manner.

4.3.5 Flow Control

4.3.5.1 Manual Flow Control

If **Flow Control Type (CR[3])** is reset, then automatic flow control is disabled and flow control must be performed in software. Either an in-channel signalling protocol (such as XON/XOFF) must be used, or the **Ucts** and **Udcd** signals must be monitored by software polling or interrupts.

The Ready To Send (**Urts**) signal may be changed under processor control by writing to **Ready To Send Status (MCR[0])**. If **Modem Status Update (MCR[2])** is set, then the **Modem Status Register** is updated with the Status and Change indication for each modem input signal. **Clear To Send Status (MSR[0])** and **Clear To Send Change (MSR[4])** are updated from **Ucts**, **Data Set Ready Status (MSR[1])** and **Data Set Ready Detect Change (MSR[5])** are updated from **Udsr**, while **Data Carrier Detect Status (MSR[2])** and **Data Carrier Detect Change (MSR[6])** are updated from **Udcd**.

Changes in **Ucts**, **Udcd**, **Udsr** and **ri** will also cause **Modem Signal Status (SR[7])** to be set. If the **Modem Interrupt Enable (CR[7])** is set, then an interrupt will be generated. Reading the **Modem Status Register** automatically clears **Clear To Send Change (MSR[4])**, **Data Set Ready Detect Change (MSR[5])** and **Data Carrier Detect Change (MSR[6])**. It also clears **Modem Signal Status (SR[7])**, and consequently, the interrupt.

4.3.5.2 Automatic Flow Control

If **Flow Control Type (CR[3])** is **set**, then automatic flow control is established using the modem control signals.

The value of **Ready To Send Status (MCR[0])** is ignored. If **Modem Status Update (MCR[2])** is set, then the **Modem Status Register** is updated with the Status and Change indication for each modem input signal. **Clear To Send Status (MSR[0])** and **Clear To Send Change (MSR[4])** are updated from **Ucts**, while **Data Carrier Detect Status (MSR[2])** and **Data Carrier Detect Change (MSR[6])** are updated from **Udcd**, as above.

Changes in **Ucts** or **Udcd** will NOT cause **Modem Signal Status (SR[7])** to be set. Thus, a modem interrupt will not be generated when automatic flow control is in use.

The UART supports two channel configurations - modem flow control and null-modem flow control. The function of **Urts**, **Ucts** and **Udcd** differs between configurations as identified in section 4.3.6, "Modem Configuration" below.

4.3.6 Modem Configuration

4.3.6.1 Modem Flow Control

If **(MCR[3])** is reset, then the modem configuration is selected. A modem converts digital signals into an appropriate format for transmission over a communications channel. Since the communications channel usually has reduced bandwidth compared to the UART, it is necessary to perform flow control to reduce the UART data rate to that which can be sustained by the channel.

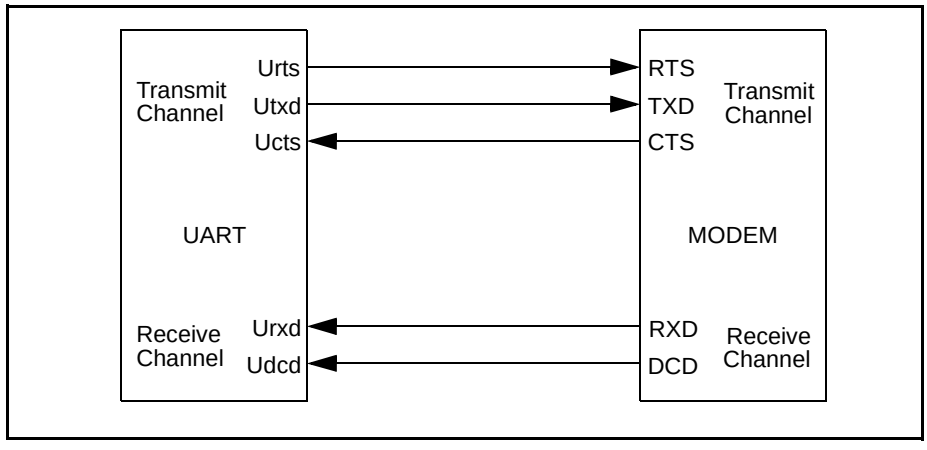


Figure 23 - Modem Configuration 0 - Modem Flow Control

When data is written to the **Transmit Register, Urts** is asserted to indicate that there is data for transmission. It reflects the state of **Transmit Register Status (SR[1])**. If the **Ucts** input is asserted, then the Transmit channel is enabled, and data transmission takes place until the **Transmit Register** is empty, or **Ucts** is negated. **Udcd** is used to indicate to the Receive channel that there is a data carrier present, implying valid data is present which may be sampled by the Receive channel. DCD must be asserted for the Receive channel to be active and CTS must be asserted for the Transmit channel to be active.

4.3.6.2 Null Modem Flow Control

If **Modem Configuration (MCR[3])** is **set**, then the **null-modem** configuration is selected. In the case of null-modem flow control (illustrated in <italic>Figure 24, "Modem Configuration 1 - Null Modem Flow Control" below), the bandwidth of data transfer is limited by the ability of the receiving device to accept new data or by the line characteristics.

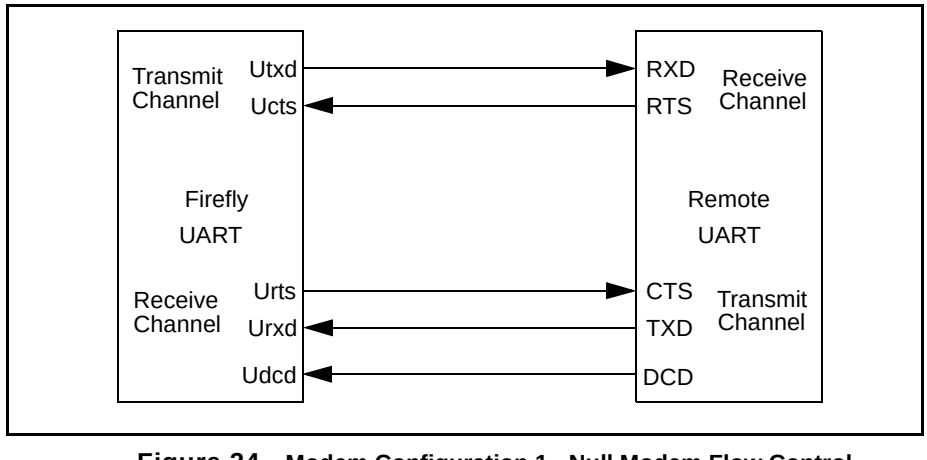


Figure 24 - Modem Configuration 1 - Null Modem Flow Control

The Receive channel indicates that it is ready to receive new data by asserting the Ready To Send (**Urts**) output, which indicates that **Receive Register Status (SR[0])** is idle. If the **Ucts** input is negated, then the Transmit channel remains idle, and no data transmission takes place. DCD must be asserted for the Receive channel to be active and CTS must be asserted for the Transmit channel to be active.

4.4 Programmer's Model

4.4.1 Register Map

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
+000	Serial Control	8	00	RW		
+004	Serial Mode	8	00	RW		
+008	Serial Baud Rate	8	00	RW		
+00C	Serial Status	8	02	R	R	
+010	Receive	8	00	R	R	Reading this register clears the Receive Register Status bit (RBF) in the SRR. Similarly, writing to it clears Transmit Register Status (TBE).
+010	Transmit	8	00	W	W	
+014	Modem Control	8	00	RW		
+018	Reserved					
+01C	Modem Status	8	00	R	R	

Table 35 - UART Address Map

1. An R in this column indicates that a read is destructive, and a W, that a write is destructive.

Addresses are specified as offsets from the base address as defined for the system. All accesses must be 8-bit. Attempts of any other access width (e.g., 32-bit wide) will cause a Data Abort exception.

4.4.2 Register Details

4.4.2.1 Serial Control Register (CR) And Serial Mode Register (MR)

The **CR** defines the current operation of the Transmit and Receive channels and the interaction with the processor via the interrupt signals. The **Control Register** bits shown below are ANDed with the **Serial Status Register** bits listed in Table 39 -, "Serial Status Register (SR)" on page 74 and the result passed to the interrupt controller. The interrupt controller then puts these interrupts on channel 14 and on receiving the interrupts, you should read the **Serial Status Register** to determine the cause of the interrupt. Table 36 -, "Serial Control Register (CR) Details" overleaf shows the register details.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7	Modem Interrupt Enable	Enables and disables the interrupt generated when the remote modem is ready to accept data; Interrupt enabled when bit is set. [0] Clear [1] Set	0	RW	
6	Error Interrupt Enable	Enables and disables the interrupt generated when any type of error is detected; Interrupt enabled when bit is set [0] Clear [1] Set	0	RW	
5	Transmit Interrupt Enable	Enables and disables the interrupt generated when the **Transmit Register Status** bit is set, indicating the **Transmit Register** is empty; Interrupt enabled when bit is set. [0] Clear [1] Set	0	RW	
4	Receive Interrupt Enable	Enables and disables the interrupt generated when the **Receive Register Status** bit is set, indicating the **Receive Register** is full; Interrupt enabled when bit is set [0] Clear [1] Set	0	RW	
3	Flow Control Type	[0] Software [1] Hardware	0	RW	
2	Clock Source	[0] Internal [1] External	0	RW	
1	Transmit Channel Control	Enables and disables the Transmit channel of the UART [0] Disable [1] Enable	0	RW	
0	Receive Channel Control	Enables and disables the Receive channel of the UART [0] Disable [1] Enable	0	RW	

Table 36 - Serial Control Register (CR) Details

1. An R in this column indicates a destructive read, and a W, that writes are destructive..

When an external clock is selected, the interface should be run at the desired baud rate and not at x16 the rate.

The **Serial Mode Register (MR)** defines the mode of operation of the Transmit and Receive channels, with the format of the serial bit stream and clock information. **Clock Source** is used to choose between the internal clock or an external clock, while **Division Select** determines which of 16 clocks from a division chain should be applied to the internal clock. <italic>Figure 37, "Serial Mode Register (MR)" below shows the register details.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7:4	Division Select	[0y0000 - 0y1111] :corresponds to division values of 1,2,4, 8……..32768	0y0000	RW	
3	Number Of Stop Bits	[0] One [1] Two	0	RW	
2	Parity Sense	This data is only relevant when "Parity Checking" is enabled. Position of parity bit depends on "Data Length". [0] Even [1] Odd	0	RW	
1	Parity Checking	[0] Disable [1] Enable	0	RW	
0	DataLength	[0] Eight: 8-bit data [1] Seven: 7-bit data	0	RW	

Table 37 - Serial Mode Register (MR)

1. An R in this column indicates reads are destructive, and a W, that writes are destructive.

4.4.2.2 Baud Rate Register (BRR)

The **Baud Rate Register** controls the generation of baud rates to within a small tolerance (described in section 4.3.1, "Baud Rate Generation" on page 65) of the desired rate. Table 38, “Baud Rate Register (BRR)” below shows the register details.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7:0	Clock Scaling Factor	Scaling factor for internal serial clock. [0 - 255] : corresponds to scaling factors 1 - 256.	0	RW	

Table 38 - Baud Rate Register (BRR)

1. An R in this column indicates reads are destructive, and a W, that writes are destructive.

4.4.2.3 Serial Status Register (SR)

The **Serial Status Register** reflects the current status of the Transmit and Receive channels, and receives errors. It interacts with interrupt enable bits in the **Serial Control Register** (**CR**) to generate interrupts under various conditions. Table 39, “Serial Status Register (SR)” below shows the register details.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7	Modem Signal Status	[0] Unchanged [1] Changed	0	R	R
6	Overrun Error status	[0] OK [1] Error	0	R	R
5	Framing Error Status	[0] OK [1] Error	0	R	R
4	Parity Error Status	[0] OK [1] Error	0	R	R
3	Transmit Status	[0] Idle [1] Active	0	R	
2	Receive Status	[0] Idle [1] Active	0	R	
1	Transmit Register Status	Cleared when Transmit Register is written to [0] Idle [1] Active	1	R	R
0	Receive Register Status	Cleared when Receive Register is read [0] Idle [1] Active	0	R	R

Table 39 - Serial Status Register (SR)

1. An R in this column indicates that reads are destructive.

4.4.2.4 Transmit Register (TR)

The **Transmit Register** detailed below in Table 40, “Transmit Register (TR)” holds data to be transmitted. Upon the start of a Transmit cycle, the contents are transferred to the **Transmit Shift Register** and **Transmit Register Status (SR[1])** is set to indicate that new data may be written, upon which the flag is cleared. The LSB is transmitted first.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7:0	Transmit Data	Data to be transmitted from serial transmit port [0 - 255]	0	W	W

Table 40 - Transmit Register (TR)

1. A W in this column indicates writes are destructive.

4.4.2.5 Receive Register (RR)

The **Receive Register** detailed below in Table 41, “Receive Register (RR)” below is a read-only register which holds data received from the serial receive port. When loaded at the end of a Receive cycle, the receiver is disabled and **Receive Register Status (SR[0])** is set to indicate that there is new data. This flag is cleared when data is read. The MSB will contain the most recently received bit.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7:0	Receive Data	Data received from serial receive port [0 - 255]	0	R	R

Table 41 - Receive Register (RR)

1. A R in this column indicates reads are destructive.

4.4.2.6 Modem Control Register (MCR)

The **Modem Control Register** detailed in Table 42, “Modem Control Register (MCR)” below defines the function of modem control and status signals and their interaction with the Transmit and Receive channels for hardware handshake mechanisms.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7:4	Reserved		0		
3	Modem Configuration	Controls Urts if [0] Normal [1] Null Modem	0	RW	
2	Modem Status Update	Allows update of the Modem Status Register [0] Disable [1] Enable	0	RW	
1	Data Terminal Ready Status	Only set under software control. [0] Clear [1] Set	0	RW	
0	Ready To Send Status	Only used under software control. [0] Clear [1] Set	0	RW	

Table 42 - Modem Control Register (MCR)

1. An R in this column indicates reads are destructive, and a W, that writes are destructive.

When using any of the modem control signals of the UART, the **Modem Status Update** bit (**MCR[2]**) must be set. This enables updates of the **Modem Status Register** to occur, which in turn reflect the status of the modem control signals. Successful data transmission and reception to and from the modem, whether using automatic or manual flow control[1], and normal or null modem flow control settings, can only take place in response to the correct sequence of control signals being observed, and acted upon by the programmer or the UART hardware, via the

1. Manual flow control involving a software protocol without the use of any modem flow control hardware signal is the only exception to the foregoing paragraph.

Modem Status Register updates. For the updates to occur, the **Modem Status Update** bit (**MCR[2])** must be set. The **Modem Status Register** can not be written to. (For more information on automatic and manual flow control, see sections 4.3.5.2, "Automatic Flow Control" on page 69 and 4.3.5.1, "Manual Flow Control" on page 69, respectively; for information on normal and null modem flow control settings, see sections 4.3.6.1, "Modem Flow Control" on page 70 and 4.3.6.2, "Null Modem Flow Control" on page 70, respectively.)

4.4.2.7 Modem Status Register (MSR)

If **Modem Status Update (MCR[2])** is set, then the **Modem Status Register** detailed in Table 43, “Modem Status Register” below reflects the current status and past changes on the modem control signal inputs. If **Modem Interrupt Enable (CR[7])** is set, then interrupts are generated on changes of these signals.

Bit	Function	Description	Reset Value	Type	Volatile[1]
7	Ring Indicator Detect Change	[0] Unchanged [1] Changed	0	R	
6	Data Carrier Detect Change	[0] Unchanged [1] Changed	0	R	R
5	Data Set Ready Detect Change	[0] Unchanged [1] Changed	0	R	
4	Clear To Send Change	[0] Unchanged [1] Changed	0	R	R
3	Ring Indicator Status	[0] Not Ready [1] Ready	0	R	R
2	Data Carrier Detect Status	[0] Not Ready [1] Ready	0	R	R
1	Data Set Ready Status	[0] Not Ready [1] Ready	0	R	R
0	Clear To Send Status	[0] Not Ready [1] Ready	0	R	R

Table 43 - Modem Status Register

1. An R in this column indicates that reads are destructive.

5.0 Interrupt Controller (INTC)

5.1 Introduction

The ARM7TDMI™ processor has two interrupt inputs named **FIQ** (Fast Interrupt Request) and **IRQ** (Interrupt Request). The FIQ channel has a higher priority and a greater number of dedicated banked registers than the IRQ channel. Both inputs accept asynchronous transitions that are delayed by one cycle for synchronisation before there is any effect on processor execution flow. For more details on how the processor handles interrupts, refer to “Chapter 2 (Programmer’s Model)” of the **ARM7TDMI Technical Reference Manual** (Rev 4) (ARM Publication number DDI 0210 A, downloadable as a PDF file from the ARMl Website http://www.com/arm/documentation?OpenDocument).

Many applications require a greater number of interrupt channels than the two provided by the ARM7TDMI processor; therefore, an interrupt controller is required to interface between multiple interrupt sources and the core.

5.1.1 Design features

- 32 independently controlled interrupt channels;
- Hardware priority encoding is provided for FIQ and IRQ interrupts to indicate the highest priority active channel for each interrupt type.

Each channel provides the following functions, all under software control:

- generation of either an **FIQ** or **IRQ** request;
- independent masking of the channel interrupt source;
- status bits to indicate the state of the channel both before and after masking;
- programmable for edge-triggered or level-sensitive sources;
- programmable for both active low or active high interrupts;
- a **FIQ** interrupt can be downgraded to an **IRQ** interrupt; and
- an **IRQ** interrupt can be upgraded to a **FIQ** interrupt.

5.2 Architecture

The Interrupt Controller architecture is illustrated in Table 25, “Interrupt Controller Functional Block and System Interconnections” below. Note that interrupt sources can come from internal modules or from external interrupt pins.

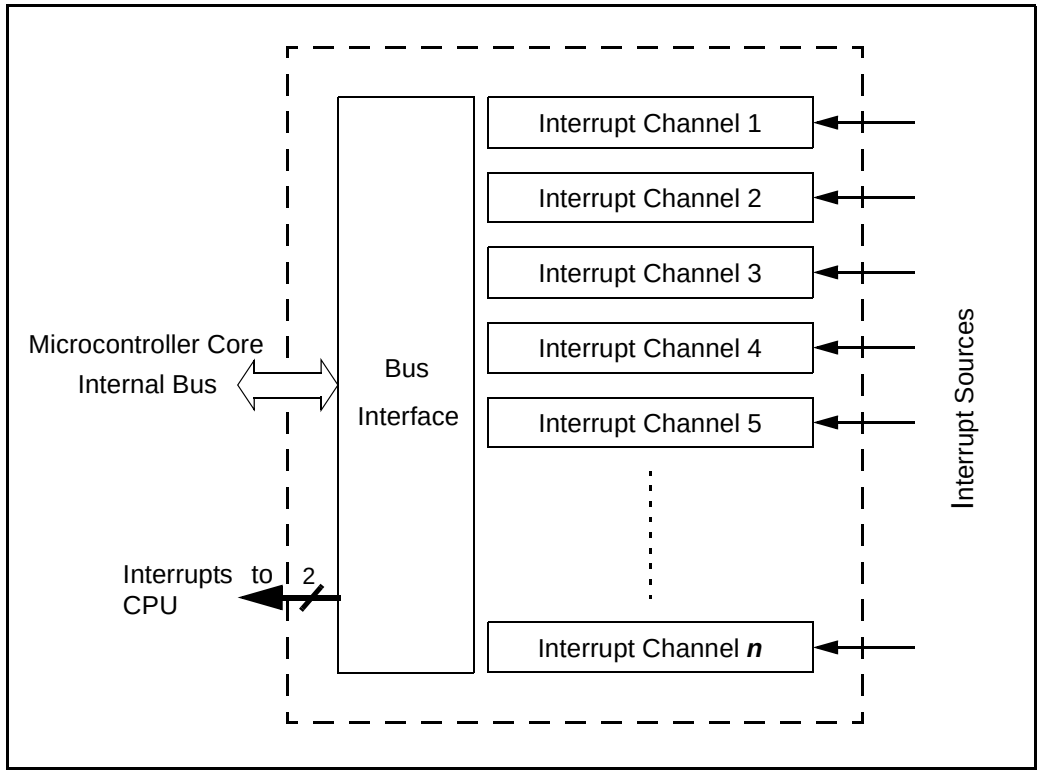


Figure 25 - Interrupt Controller Functional Block and System Interconnections

5.3 Operational Description

5.3.1 Interrupt Controller Structure

The Interrupt Controller consists of identical, independent channels, with each channel capable of handling one interrupt source. The channel is split into two parts, the Interrupt Processor and the Priority Encoder. The channels are supported by the bus interface, which ties all the channels into a common register view. Each channel is represented in the control/status registers by a single bit. Channel 0 is represented by bit 0, channel 1 by bit 1 and so on.

The **Interrupt Sources Register** channel allocation is detailed in Table 7, “Interrupt Sources” on page 24.

5.3.2 Interrupt Processor

The Interrupt Processor block allows the interrupt source to be processed in a number of ways. Each of the options is controlled by a single bit of a register corresponding to that particular interrupt channel. The options available are:

- programmable polarity of interrupt source [**Polarity Register**];
- edge or level sensitivity [**Edge/Level Triggered Register**];
- enabling/disabling of interrupt [**Interrupt Enable Register**];
- reset of edge sensitive inputs (level sensitive inputs must be reset at the source device) [**Reset Edge Triggers**];
- programmable type assignment (i.e., **FIQ** or **IRQ** interrupt to ARM®) [**Interrupt Type Register**].

Further read-only registers are provided to read the interrupt status at several points in the processing chain, providing significant flexibility for real-time programming control.

The points at which the signal can be read are:

- at input (the raw interrupt source signal), visible via the **Interrupt Sources** register;
- after polarity selection but before the **Interrupt Enable Register,** via the **Interrupt Active** register (NB, after polarity selection, all signals are active high);
- after the **Interrupt Enable Register,** via the **Interrupt Status** register (NB, only unmasked interrupts will be visible at this point);
- after type selection, via **FIQ Active Sources** and **IRQ Active Sources** registers.

All channels are capable of handling asynchronous inputs. The synchronisation is handled within the interface to the ARM processor itself and there is no restriction on the timing of inputs to the Interrupt Controller.

5.3.3 Priority Encoder

The Priority Encoder module of the Interrupt Controller can be used to shorten interrupt latency times. If not used, all priority sorting must be carried out in software. The cost of shortening the interrupt latency using the priority encoder is that the relative priorities of all interrupts are fixed, since the encoder uses a daisy-chain approach. This puts each interrupt channel in a chain with the highest priority sources at the top (see Table 7, “Interrupt Sources” on page 24). There is a separate encoder for both **FIQ** and **IRQ** interrupt types.

When these two Priority Encoder registers are read, they will each return an integer value describing which of the 32 channels is the highest priority active and enabled interrupt. The integer result is in the range 0 to 32, where 0 is the highest priority, 31 is the lowest and 32 represents no active interrupt available. These priority levels relate to the interrupt inputs, where **interrupt_source**(0) is the highest priority and **interrupt_source**(31) is the lowest. The integer is multiplied by four (shifted left two bits to word-align it) so it may be used directly to index into a branch table to select the appropriate interrupt service routine.

The **Encoded Priority Register** is shown in <italic>Figure 26 -, “Encoded Priority Register” below. An example code sequence showing how to use this register in an interrupt service routine is found in section 5.5, "Using the Interrupt Controller” on page 81 .

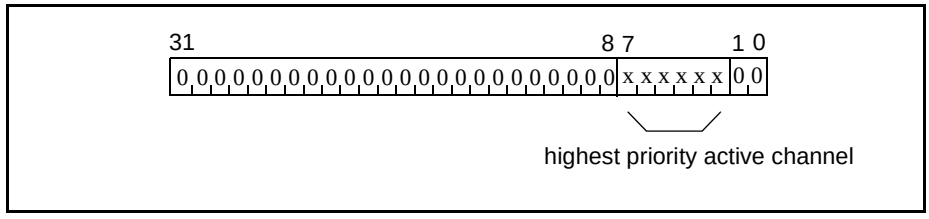


Figure 26 - Encoded Priority Register

Note that because interrupts can arrive asynchronously, they are effectively blocked at the INTC while reading these and any of the other registers in the interrupt controller. Edge triggered interrupt sources, which only last a single cycle, therefore should be avoided, as they potentially could be ignored. So, the active pulse width should be a minimum of two clock cycles.

0.0.2 Bus Interface Control

The Bus Interface handles all the register selects and control functions of the Interrupt Controller. Seen from the internal bus, the Interrupt Controller is a collection of memory-mapped 32-bit registers. The address map is given in Table 44, “INTC Register Map” below. The **FIQ** encoded priority value can be read from address offset 0x028. The **IRQ** encoded priority value can be read from address offset 0x02C. The five remaining read-only registers shown describe the status of each channel.

Offsets 0x030 to 0xFFC are blank, with no registers allocated to this area. The Interrupt Controller will cause a bus error if the system attempts to access these locations. It will also cause a bus error if an attempt is made to write to the read-only registers. The **Reset Edge Triggers** register at offset 0x010 is a transient function (i.e., it resets the edge triggering but does not hold it reset, leaving it ready to receive further interrupts.) The location does not store a value. A bus error occurs if a read is attempted to this location.

All bus accesses to the Interrupt Controller are handled in a single cycle.

5.3.4 External Interface

There are a total of 32 interrupt channels in Firefly, 8 of which are preassigned and are associated with functions within particular macrocells. The remaining 24 are effectively ‘spare’ and are called external interrupts (**ext_int**xx), which can be fed directly from Firefly ports to the Interrupt Controller. For details of interrupt channel allocations, see section 2.7.2, "Interrupt Sources” on page 24.

5.4 Programmer's Model

Table 44, "INTC Register Map" below lists the **Interrupt Controller** registers, all of which are accessed as 32-bit registers, with unused bits defined as zero for read operations. Attempts to access any register with a half-word or byte transfer result in a bus error, as do any accesses to reserved registers or writes to read-only registers.

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
+000	Interrupt Sources	32	*	R		Contains the current raw value of all interrupt channels.
+004	Polarity	32	FFFFFFFF	RW		[0] Active Low [1] Active High
+008	**Interrupt Active**	32	00000000	R		Indicates the channels whose interrupts are currently active [1] Set: channel interrupt is active [0] Clear
+00C	**Interrupt Trigger**	32	00000000	RW		[0] Level [1] Edge
+010	Reset Edge Triggers	32	00000000	W		[1] Reset
+014	**Interrupt Enable**	32	00000000	RW		Indicates the channels whose interrupts are currently enabled [1] Set: channel interrupt is enabled [0] Clear
+018	**Interrupt Status**	32	00000000	R		Indicates the channels whose interrupts are currently both active AND enabled [1] Active And Enabled
+01C	Interrupt Type	32	00000000	RW		[0] IRQ [1] FIQ
+020	FIQ Status	32	00000000	R		[1] Active
+024	IRQ Status	32	00000000	R		[1] Active
+028	FIQ Encoded Priority	32	00000000	R		Active bits 2-6 see 5.3.3, "Priority Encoder" on page 78 for details
+02C	IRQ Encoded Priority	32	00000000	R		Active bits 2-6 see 5.3.3, "Priority Encoder" on page 78 for details

Table 44 - INTC Register Map

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
+030- FFC	Reserved	32	00000000	R		Do not use

Table 44 - INTC Register Map (continued)

1. An R in this column indicates a read is destructive while a W indicates a write is destructive.

Note: * Values dependent on the reset values of the individual interrupt sources.

5.5 Using the Interrupt Controller

The ARM® processor has a simple mechanism for interrupt priority handling. When an IRQ interrupt is received, it sets a mask bit in the **Program Status Register** to prevent any further IRQ interrupts being serviced (see the ARM7TDMI™ (Thumb® Core) Handbook, Chapter 3 “Programmers Model” for more information about the **Program Status Register**). However, the ARM may service a **FIQ** interrupt during this time. When a **FIQ** interrupt is received, it sets mask bits in the **Program Status Register** to prevent any further **IRQ** or **FIQ** interrupts being serviced. This mechanism is enhanced by the addition of the Interrupt Controller, to allow multiple interrupt sources and to facilitate interrupt service.

In a system with multiple interrupt sources, it is usually required that a higher priority interrupt will be able to interrupt a lower priority service routine. If both are either **FIQ** or **IRQ** interrupts, then the interrupt service routines must be written as reentrant code. The writing of reentrant code is beyond the scope of this manual, but is fully explained in the ARM “ARM Software Development Toolkit User And Reference” guides.

Two simple examples of using the Interrupt Controller in conjunction with the ARM processor interrupt mechanism are given below. The first is written in assembly code and uses the **FIQ Encoded Priority Register** to determine the channel of an FIQ interrupt, while the second is written in C and uses the **IRQ Active Source Register** to determine the interrupting channel and branch to the appropriate service routine.

Example 1:

```
fiq_service
     LDR R9,[R8,#0x28]; read the priority register
     ADD PC,PC,R9; index into branch table
     NOP       ; take account of pipeline
     B fiq_isr0; branch table start
     B fiq_isr1
     B fiq_isr2
     .
     .
     B fiq_isrn
```

The assembly code example above illustrates the use of the **FIQ Encoded Priority Register** and a means of achieving extremely low interrupt latency. To carry out any form of processing, it would be more common to save some of the register values and restore them afterwards. This can be implemented within the individual functions that are invoked, thus allowing a high priority, but simple function to execute in minimum time.

The **fiq_service** function may be located at address 0x1C, the FIQ interrupt vector address, so that a branch instruction to this function (**fiq_service**) is not required.

In this example, to improve latency the base address of the interrupt controller has already been loaded into register 8 of the FIQ banked ARM registers during initialization.

Example 2:

```
static void __irq irq_service(void)
{
     unsigned long irq_active;

     irq_active = *((unsigned long*)INTBASE + 0x24);
     if (irq_active & 0x8) irq_priority_0();
     else if (irq_active & 0x80) irq_priority_1();
     else if (irq_active & 0x800) irq_priority_2();
               |
               |
     else if (irq_active & 0x40) irq_priority_n();
}
```

The algorithm above is a very simple reorganization of the fixed priorities of the hardware. Further complexity could easily be implemented; however, more complexity is likely to affect interrupt latency.

Another common problem occurs when one interrupt channel normally needs to be low priority, but under certain circumstances, may want to promote itself to complete a specific task. This promotion may be accomplished by the routine writing to the **Interrupt Type Register** and changing the Type of its channel from an **IRQ** to an **FIQ**. As there is still an active interrupt, a **FIQ** interrupt is immediately generated, thus promoting that task. In the case of Edge Triggered interrupts, this must be carried out before the edge trigger is reset by writing to the **Reset Edge Triggers** location.

It is also possible for a **FIQ** to downgrade to an **IRQ** interrupt in a similar way; however, the **IRQ** will still be locked out until the **FIQ** has finished, when it will be processed in the normal way.

6.0 DMA Controller (DMAC)

6.1 Overview

Data transfer between memory blocks, or between memory and a peripheral, can be extremely cycle-intensive for a processor; for example, the ARM7TDMI™ processor requires at least nine clock cycles to move a word of data from one address in memory to another. The microcontroller core contains a DMA controller that assists the processor to move large blocks of data around a system.

The DMA system is initialised by the processor with a source and a destination for the data transfer. On receipt of a DMA request, the DMA Controller acquires control of the system address and data buses and proceeds to transfer data until a stop condition is met. The DMA Controller may then be auto-initialised or manually reprogrammed as desired. Features of the DMA Controller include:

- Maximum transfer rates of 100 MBytes per second (single-addressing), 50 MBytes per second (dual-addressing) at 25 MHz, zero wait-state transactions;
- 32-bit (4 GByte) addressing range, address increment, decrement and hold;
- Data transfer sizes of 8-, 16-, and 32-bits (statically sized);
- 16-bit (65536 item) maximum transfer count;
- Single-addressed (fly-by) transfer performed by individual channels;
- Dual-addressed (memory-memory and peripheral-peripheral) transfers supported by linked channel pairs;
- Transfers can be triggered by software;
- Maskable level or edge sensitive hardware transfer triggers with selectable polarity;
- Maskable hardware transfer acknowledge signals with selectable polarity;

- Block and Packet mode transfers supported;
- Wait state insertion when indicated by slow memory;
- Auto-initialization of channels on completion;
- Chained DMA transfers supported for scatter-gather operations;
- Selectable interrupt generation on completion;
- Fixed channel priority;
- Optional bus locking to prevent interruption of DMA services by other bus masters;
- Programmable DMA triggers from internal and external sources;
- Abort mechanism for illegal access with interrupt generation and transfer halt.

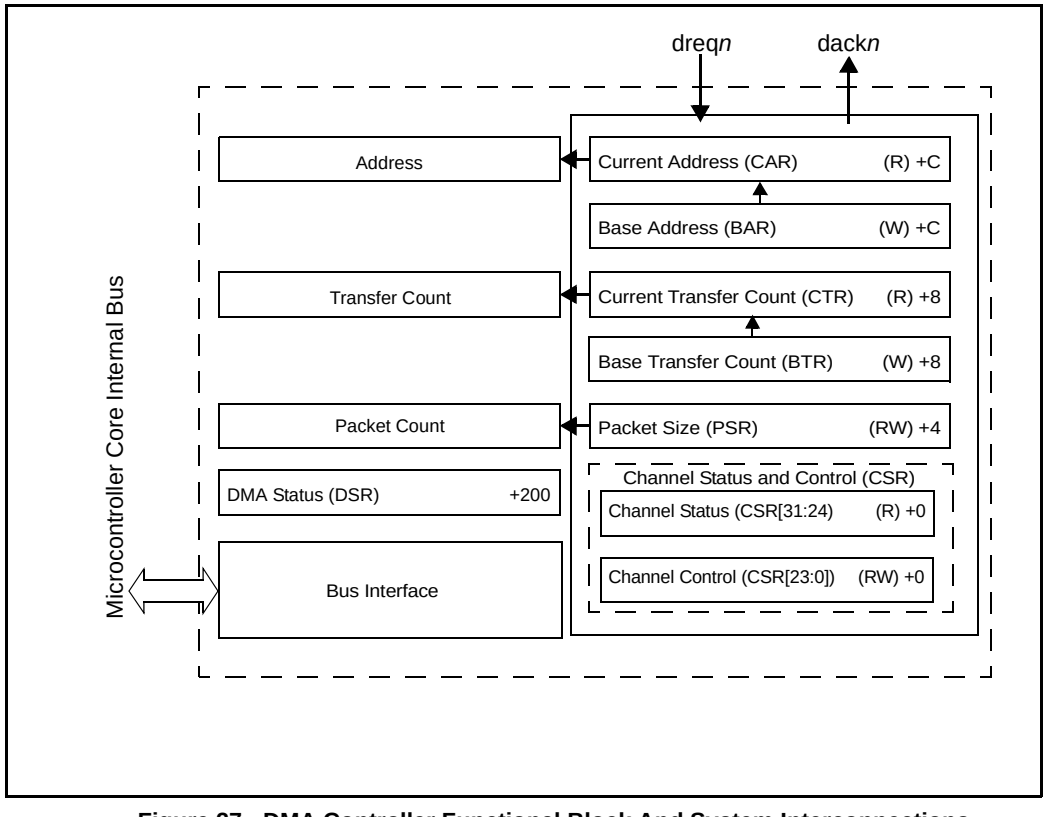


Figure 27 - DMA Controller Functional Block And System Interconnections

Port	Direction	Sense	Description
dreq***n***	Input	Programmable active high/low	DMA service request for channel ***n***.
dack***n***	Output	Programmable active high/low	DMA service acknowledge for channel ***n*** - implicitly addresses a peripheral

Table 45 - DMA controller signals

6.1.1 DMA Controller Trigger Selection

Each channel DMA request (**dreq**) may be connected to one of several sources by programming the appropriate value into the **Channel Select** bits of the **System Configuration Register**, which is described in section 2.7, "System Configuration".

6.2 Operational Description

6.2.1 Single-Addressed (Fly-by) Transfers

DMA transfers are one-to-one transactions, with a data source and destination. High speed data transfer applications require use of the full bandwidth of the internal data bus, which corresponds to one data transaction per bus cycle. Since only a single address may be explicitly associated with the data (and presented on the address bus as either source or destination), the corresponding address must be implicitly signalled. This is performed using a hardware handshake protocol, which allows the DMA controller to directly access the implicitly addressed device.

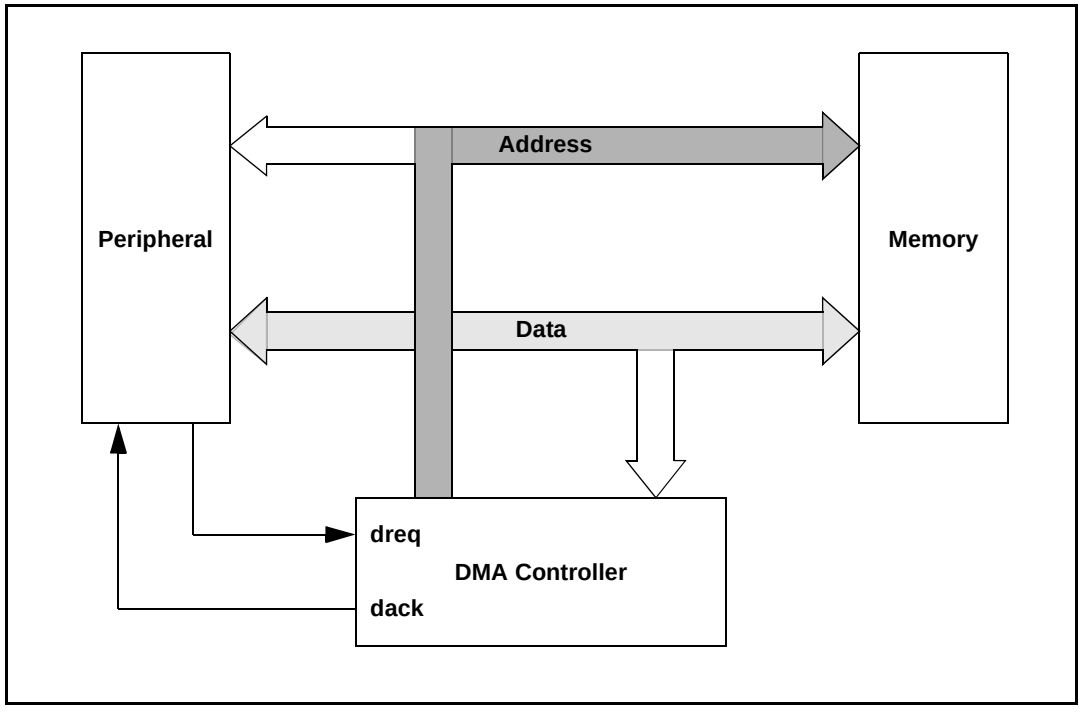


Figure 28 - Architecture Of Single-Addressed DMA

Each DMA channel is capable of generating an address and a hardware acknowledge signal for a particular data transaction. Data is presented to the bus by the source and written to the destination during a single bus transaction; it is **not** buffered by the DMA controller. An address is broadcast onto the bus simultaneously. This transfer mode is referred to as a 'single-addressed' or 'Fly-by' transaction, and **Transfer Mode (CSR[9])** must be ***clear***. If the address is the source of the transfer, then **Transfer Direction (CSR[8])** should be ***clear***; if the address is the destination, then the bit should be ***set***.

Transfers are usually triggered by hardware transfer requests.

6.2.1.1 Block Transfers

If the target device can process data with the same bandwidth as the bus then the data may be transferred in consecutive bus cycles; this is referred to as a 'Block Transfer', and **Transfer Type (CSR[10])** must be ***set***. The function of the hardware request signal is determined by the value of **Request Trigger Type (CSR[11])**.

6.2.1.2 Edge-Triggered Block Transfers

If **Request Trigger Type (CSR[11])** is ***clear***, the channel becomes edge sensitive and monitors for the assertion of the request input. Once triggered, the channel will ignore subsequent transitions on the hardware request input until channel activity is completed. The input must be negated before the channel can be retriggered.

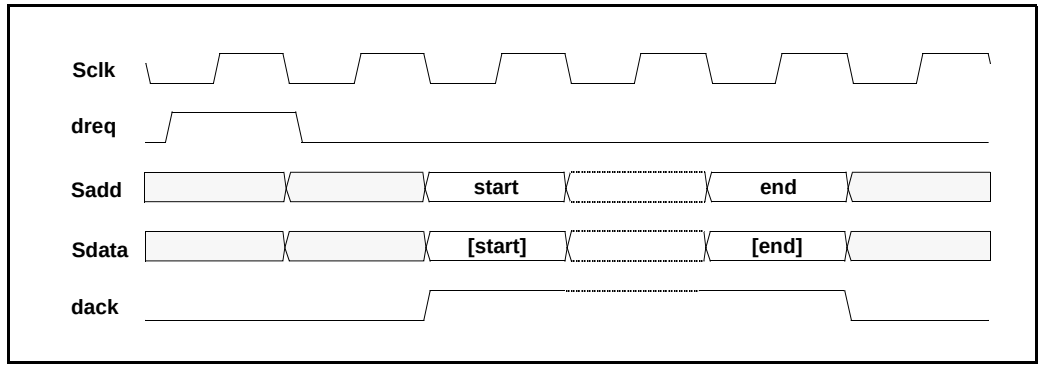


Figure 29 - Edge Triggered Block Transfer

6.2.1.3 Level-Triggered Block Transfers

If **Request Trigger Type (CSR[11])** is ***set***, then the channel continues to test the hardware request input throughout a transaction; if the input is negated, the channel activity is halted on completion of the current bus cycle and the channel returns to the Idle state. If no other channel is requesting DMA activity, then at least one null bus cycle will be inserted.

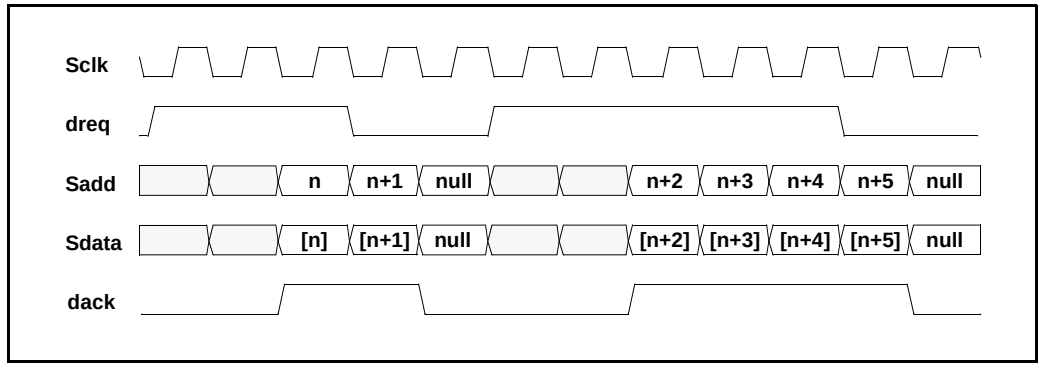


Figure 30 - Level Triggered Block Transfer

6.2.1.4 Edge-Triggered Packet Transfers

Some devices of reduced bandwidth and with limited internal buffering require the interruption of DMA service while previously transferred data is processed. Transfer is recommenced when the transfer request is retriggered. This transfer type is referred to as 'Packet Transfer' and **Transfer Type (CSR[10])** must be ***clear***. The size of the buffer to be transferred must be programmed into the channel **Packet Size Register (PSR)**.

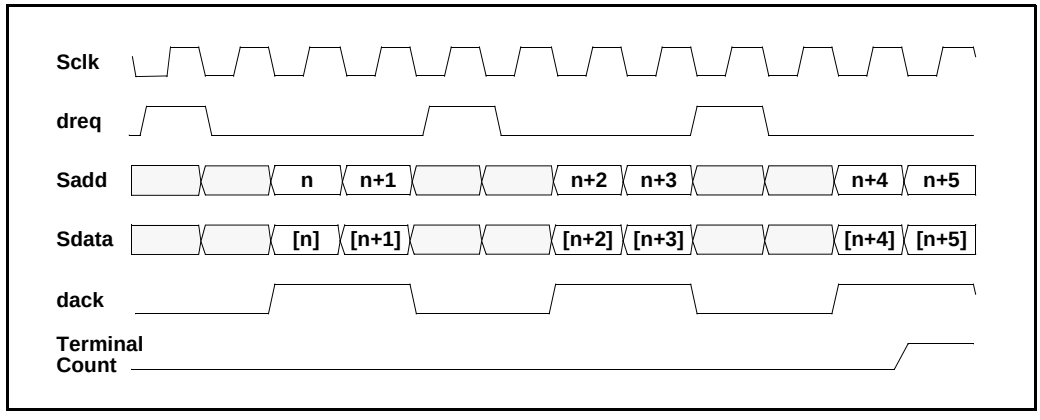


Figure 31 - Edge Triggered Packet Transfer (Size = 2)

Note that if Packet Transfer and Level Sensitive input are selected and then the request is negated *before* the number of transfers completed equals the number set in the **Packet Size Register**, then the remaining words to transfer will be discarded by the DMA controller; retriggering will cause the DMA Controller to reload the packet size from the channel **Packet Size Register**. Therefore, it is recommended that the input be edge sensitive.

6.2.1.5 Software-Triggered Transfers

Software requests are issued when **Software Request (CSR[2])** is ***set***. Software requests are usually terminated by channel completion, but may also be terminated in software by ***clear***ing **Software Request (CSR[2])**.

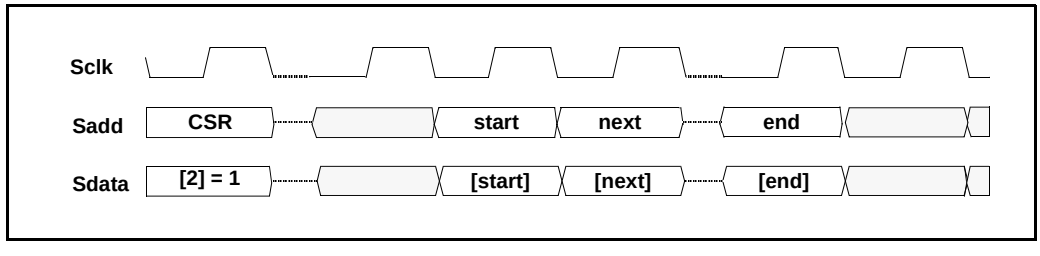


Figure 32 - Software Triggered Block Transfer

6.2.2 Dual-Addressed (Buffered) Transfers

When data must be transferred from one memory block to another, or a peripheral device does not provide support for implicit addressing through a hardware handshake, then an explicit address is required for both source and destination. Each DMA transfer then requires two bus transactions: the first, to read the data from the source and the second, to write the data to the destination. An explicit address is generated for each transaction, and the data must be buffered internally by the DMA Controller between read and write.

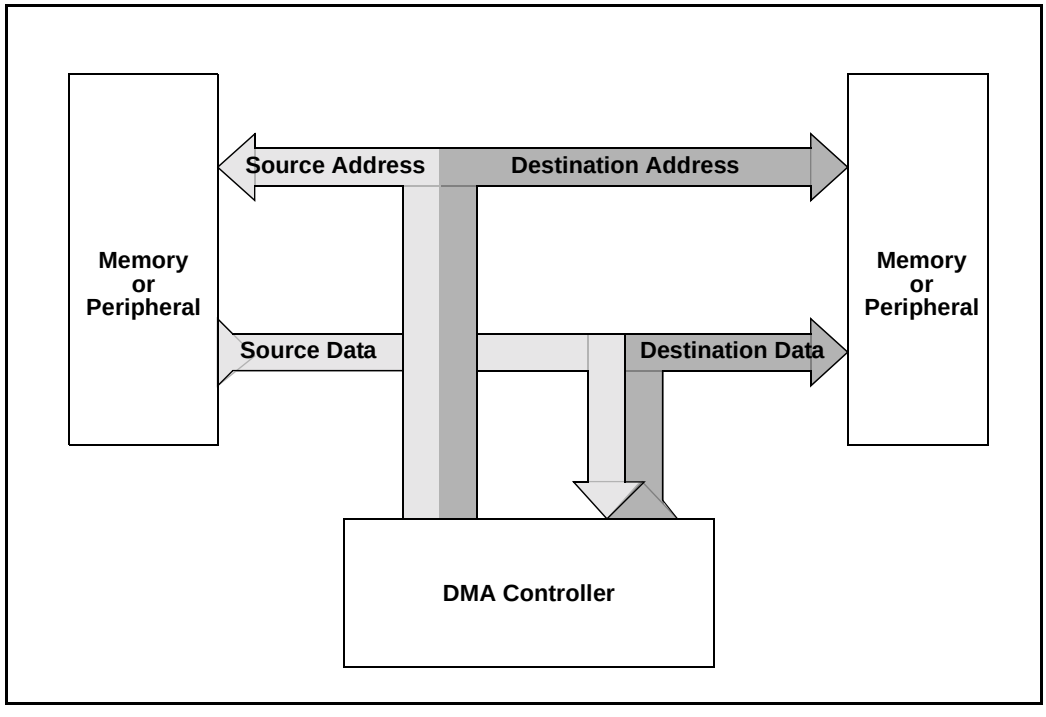


Figure 33 - Architecture Of Dual-Addressed DMA

The number of source data read transactions which may be performed before the buffered data must be written to the destination is determined by the internal buffer depth. In this DMA controller, the buffer depth is one word.

If dual-addressed transfers are required, then two adjacent channels must be selected in order to provide the source and destination addresses. The higher priority channel (lower channel number) becomes the “Read” channel and **Transfer Direction (CSR bit [8])** must be ***clear***. The lower priority channel of the pair becomes the Write channel and **Transfer Direction (CSR[8])** must be ***set***. **Transfer Mode (CSR[9])** should be ***set*** for both channels.

6.2.2.1 Block Transfers

Some devices, such as memory, can process data at the full bus rate; if so, then the data may be transferred in consecutive bus cycles. This is referred to as a ‘Block Transfer’ and **Transfer Type (CSR[10])** must be ***set***. The function of the hardware request signal is determined by the value of **Request Trigger Type (CSR[11])**.

6.2.2.2 Edge-Triggered Block Transfers

If **Request Trigger Type (CSR[11])** is ***clear***, the channel becomes edge sensitive and monitors for the assertion of the request input of the Read channel. Once triggered, the channel will ignore subsequent transitions on the hardware request input until activity is completed on both Read and Write channels. The input must return to the negated level before the channel can be retriggered.

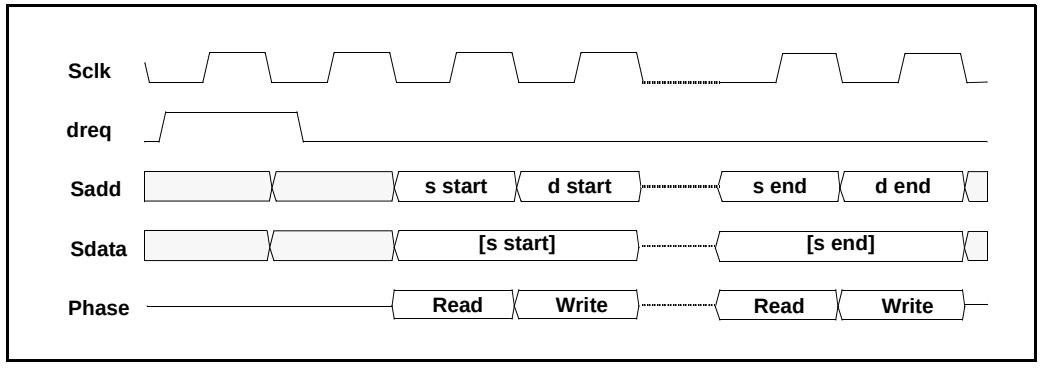


Figure 34 - Edge Triggered Block Transfer

6.2.2.3 Level-Triggered Block Transfers

If **Request Trigger Type (CSR[11])** is ***set***, then the channel continues to test the hardware request input throughout a transaction. If the input is then negated, channel activity is halted after the remaining buffered data has been written to the destination and the channel returns to the Idle state.

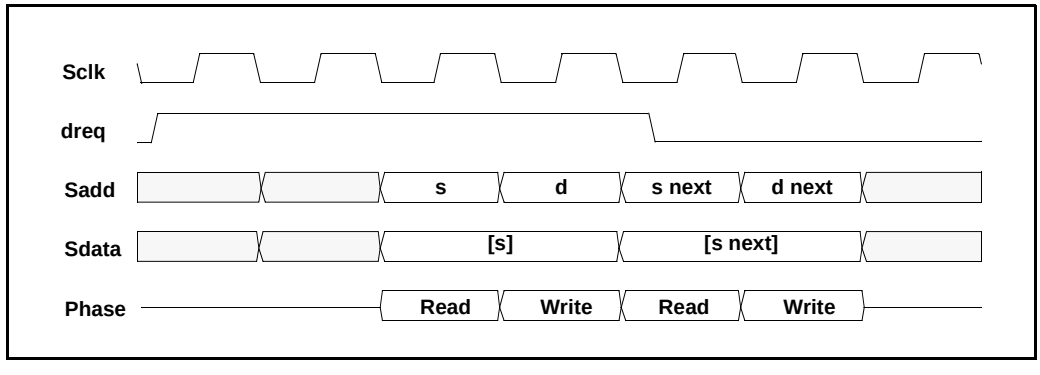


Figure 35 - Level Triggered Block Transfer

If the request is negated during the Write phase of the DMA transfer with no other channel requesting DMA activity, then at least one empty bus cycle will be inserted before the bus is granted to another bus master.

6.2.2.4 Edge-Triggered Packet Transfers

However, some devices of reduced bandwidth and with limited internal buffering, such as UARTs, require the interruption of DMA service while previously transferred data is processed. Transfer is recommenced when the transfer request is retriggered. This transfer type is referred to as 'Packet Transfer' and **Transfer Type (CSR[10])** must be ***clear***. The size of the buffer to be transferred is dictated by the internal buffer depth and the **Packet Size Register** will be ignored.

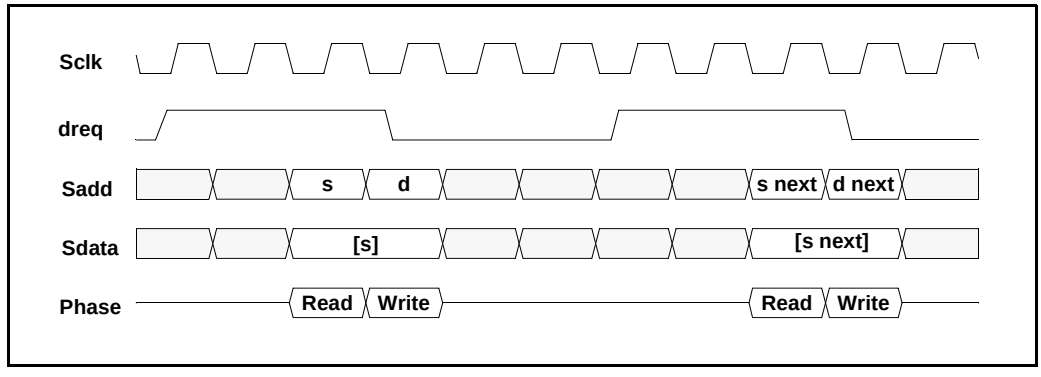


Figure 36 - Edge Triggered Packet Transfer

6.2.2.5 Software Transfer Requests

Software requests are issued when **Software Request (CSR[2])** is ***set***. Software requests are terminated by channel completion and may also be terminated by *clearing* **Software Request (CSR[2])**. If **Transfer Type (CSR[10])** is ***set***, then the DMA controller will transfer the entire block of data in consecutive read and write cycles.

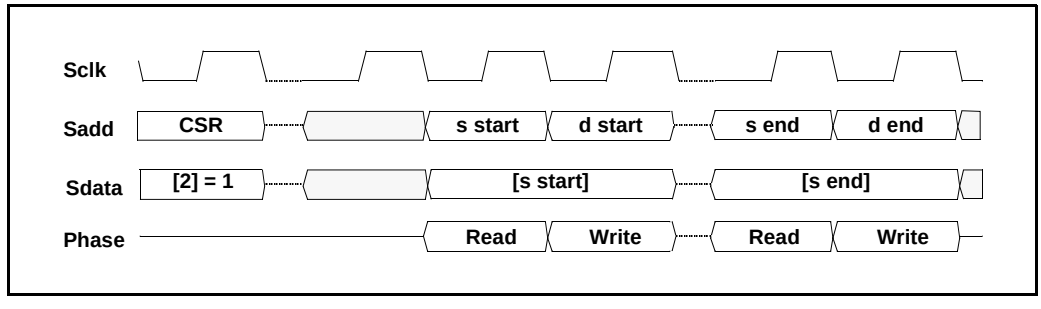


Figure 37 - Software Triggered Block Transfer

It should be noted that the use of Block Transfer starves the processor of bandwidth. If the user wishes to guarantee some bus bandwidth to the processor, then **Transfer Type (CSR[10])** must be ***clear***, indicating Packet Transfers. This guarantees at least one Idle bus cycle between each DMA transfer.

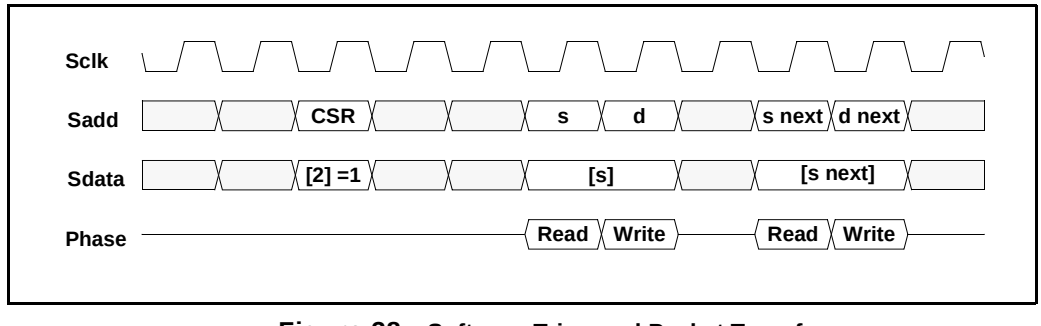


Figure 38 - Software Triggered Packet Transfer

The Read channel should be programmed with appropriate values for the **Base Transfer Count Register** and the **Base Address Register**. Only the **Base Address Register** need be programmed for the Write channel; for dual-addressed transfers, the **Packet Size Register** of both channels and the **Base Transfer Count Register** and **Current Transfer Count Register** of the Write channel have no effect on the operation.

6.2.3 Configuration

6.2.3.1 Hardware Request (dreq) And Hardware Acknowledge (dack)

Hardware requests are enabled when **Hardware Request Status (CSR[1])** is ***set*** (see Table 47, “Channel Control And Status Register (CSR)” on page 94).The trigger polarity is programmed either high or low by **Hardware Request Polarity (CSR[3])**. DMA request source status and information are detailed in Table 6, “DMA Channel 1 Trigger Selection” on page 24 and Table 47, “Channel Control And Status Register (CSR)” on page 94.

DMA transfers are acknowledged by assertion of the **dack** output signal. The signal is active high when **Hardware Acknowledge Polarity (CSR[4])** is ***clear***, or low when ***set***. Single-addressed transfers require the assertion of a hardware acknowledge, and consequently **Hardware Acknowledge Status(CSR[5])** must be ***set***. For dual-addressed transfers, a hardware acknowledge is not usually necessary and is masked out when the appropriate bit is ***clear***.

6.2.3.2 Address Calculation

The address calculation performed by the DMAC is determined by the values of **Address Direction (CSR[7])** and **Address Mode (CSR[6])**.

- If **Address Mode (CSR[6])** is ***set***, then the address will be held for each data transaction, as is required when accessing a FIFO, for example.
- If **Address Mode (CSR[6])** is ***clear***, then the address will be incremented if **Address Direction (CSR[7])** is ***set***, otherwise it will be decremented.
- The address offset is dependent on the Value of **Operand Size (CSR[13:12])**, with offsets of 1, 2, and 4 for byte, half-word, and word transfers respectively.

The DMA controller does not directly support byte packing and unpacking, so **Size (CSR[13:12])** must have the same value in each channel. However, the Memory / Peripheral Controller is capable of packing and unpacking operands as appropriate. See Chapter 3.0, “Memory/Peripheral Controller (MPC)” on page 37 for details.

6.2.3.3 Re-initialization

When a DMA transfer is complete, the DMA supports several options for re-initialization.

- No initialization - channel returns to Program state. **Auto initialization (CSR[14])** should be ***clear*** and the state of **Chain Interrupt Enable (CSR[15])** will be ignored.
- Auto-initialization - channel re-initialises with values from the previous transfer as stored in the **Base** registers, and returns to Idle state. This mechanism supports repeated transfers into and between memory buffers. **Auto initialization (CSR[14])** should be ***set*** and **Chain Interrupt Enable (CSR[15])** should be ***clear.***
- Chained Transfer initialization - the channel re-initialises with values from the **Base** registers, which are updated by the supervising processor as a result of an interrupt service request (see section 6.2.3.4, "Chained Transfers") below. Both **Auto initialization (CSR[14])** and **Chain Interrupt Enable (CSR[15])** should be ***set***. Interrupts must be enabled, so **Interrupt Enable (CSR[16])** should also be ***set.*** This option provides for scatter / gather type operations, as described below.

6.2.3.4 Chained Transfers

Many applications require support for the scattering and gathering of blocks of data. The DMA controller offers a “Chained Transfer” mode, which allows multiple blocks of data to be transferred with minimal processor intervention. When a channel programmed for chained transfers is enabled, the current registers are updated with the **Base** register contents and an internal flag, **Last Block**, is ***set***. **Chain Empty (CSR[26])** is also ***set***, triggering an interrupt if **Interrupt Enable (CSR[16])** is ***set***.

Reading the **Channel Status Register** will ***clear*** **Chain Empty (CSR[26])**, negating the interrupt. New block values may now be written to the **Base** registers. The **Base Address Register** (in the case of dual-addressed transfers, of the Read channel) should be written last, which will clear the **Last Block** flag. If this occurs before “Terminal Count” (or “Early Termination”) is reached, then the chain is established, and the new **Base** register values are transferred to the **Current** registers upon completion of the current block. The first transfer of the new block will then proceed when appropriately triggered. Since the new register values are loaded automatically, there is no requirement to re-enable the channel - **Channel Status (CSR[0])** will be ***set*** from the previous block transfer.

If the new **Base** register values are not written (the **Last Block** internal flag remains set), and the channel reaches “Terminal Count” or “Early Termination” occurs, then the chain is broken. The channel returns to the “Program” state and indicates that the chain has broken with **Chain Ended (CSR[27])** ***set*** and **Channel Status (CSR[0])** ***clear***. If new block parameters are programmed, **Channel Status (CSR[0])** must be ***set*** to re-enable the channel.

If the service routine does not read the **Channel Status Register**, **Chain Empty (CSR[26])** is also cleared after the **Base Address Register** is written.

6.2.3.5 Interrupts

Interrupts are enabled for a given channel by **Interrupt Enable (CSR[16])**. This permits the generation of an interrupt under the following conditions:

- no initialization is programmed and “Terminal Count” is reached;
- a new transfer block is required, or the block chain is exhausted, when Chained initialization is programmed; or
- upon detection of a bus error while undertaking a bus transaction.

The actual generating condition can be determined by reading the **DMA Status Register** to identify the DMA channel requesting interrupt service, then by testing the corresponding **Channel Status Register**. Note that this action also ***clears*** the register (with the exception of the **Write In Progress (CSR[31])** indicator), thereby negating the interrupt request. In dual-addressed transfers, **Interrupt Enable (CSR[16])** of the write channel should be ***clear***, disabling interrupts from that channel.

6.2.3.6 Channel Priorities And Bus Locking

Channel priorities are hard-wired, with channel 1 having the highest priority. A higher priority channel may pre-empt a lower priority channel, except under one of the following conditions:

- for single-addressed transfers, if **Bus Lock (CSR[17])** is ***set*** (this guarantees the full bus bandwidth to the currently active DMA transfer); or
- for dual-addressed transfers, if the read or write phase is currently active. The bus is also automatically locked **between** the Read and Write phases to ensure atomic DMA operations. Dual-addressed transfers may only be pre-empted between the last write of one buffer and the first read of the next, unless **Bus Lock (CSR[17])** is ***set***, guaranteeing data consistency.

Other higher priority bus masters are also prevented from pre-empting the DMA controller under these conditions.

6.2.3.7 DMA Devices External To The Firefly MF1 Core

As the implicitly addressed DMA targets of single-addressed transfers may be either internal or external to the microcontroller, it is necessary to indicate to the memory/peripheral controller the location of the DMA target, so that the bus interface may be switched to pipe data in the correct direction, or to remove bus drive as appropriate.

Peripheral Location (CSR[18]) should be ***clear*** if the target device is internal to the microcontroller, and should be ***set*** if external. Note that for dual-addressed transfers, this bit should always be ***clear***, as the data buffer for dual-addressed transfers is internal to the DMA controller.

6.3 Programmer's Model

Upon reset, each DMA channel is set to an inactive "Program" state, from which it may be configured by a supervising processor. Since values written into the channel in any other state may have undesired side effects, **it is essential** that the channel be put into Program state first. **Channel Status (CSR[0])** must be ***clear***; no other register bits should be altered at this time. The programmer may verify that the channel is in Program state by checking that **Channel Status (CSR[0])** is ***clear*** by reading the **Control & Status Register**. When the channel is in Program state, there is no channel activity. However, all internal registers may be read and written.

6.3.1 Channel initialization

The **Packet Size Register**, **Base Transfer Count Register**, and **Base Address Register** may be modified in Program state; changes will not be immediately reflected in the **Current Transfer Count Register** and **Current Address Register**. Once these registers are configured, the channel may be enabled by setting **Channel Status (CSR[0])**, at which point the values programmed into the **Base** registers are transferred to the **Current** registers, and the channel enters Idle state. The settings of the other bits in the **Control & Status Register** will dictate the subsequent operation of the channel.

6.3.2 DMA Registers

The DMA controller is of modular construction. A single DMA channel comprises one DMA single- address (Fly-by) engine. The exact behaviour of each channel and any channel pairing is configurable by software. The software programmable internal registers are mapped into the DMA controller address space by channel, with an additional status register. Addresses are specified as offsets from the DMA controller base address.

Each DMA channel contains a set of registers which allow the appropriate configuration of the channel in software to perform a variety of DMA operations.

6.3.3 Register Map

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
DMA Channel 1						
+000	Control and Status	32	00000000	RW	R	Contains 'read volatile' bits which are reset to 0 every time the register is read.
+004	Packet Size	32	00000000	RW		Bits 31:8 are reserved

Table 46 - DMA Register Map

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
+008	Base Transfer Count	32	00000000	W		Bits 31:16 are reserved
+008	Current Transfer Count	32	00000000	R		Bits 31:16 are reserved
+00C	Base Address	32	undefined	W		
+00C	Current Address	32	00000000	R		
DMA Channel 2						
+010	Control and Status	32	00000000	RW	R	Contains 'read volatile' bits which are reset to 0 every time the register is read.
+014	Packet Size	32	00000000	RW		Bits 31:8 are reserved
+018	Base Transfer Count	32	00000000	W		Bits 31:16 are reserved
+018	Current Transfer Count	32	00000000	R		Bits 31:16 are reserved
+01C	Base Address	32	undefined	W		
+01C	Current Address	32	00000000	R		
Other Registers						
+020	Reserved					
+1F0	Reserved					
+200	Channel Status	32	00000000	R		

Table 46 - DMA Register Map (continued)

1. An R in this column shows the value is reset to zero whenever the register is read.

6.3.4 Register Details

The register word offset must be added to the channel address offset and controller base address to form the register address.

The DMA controller also integrates an internal data buffer, which is not visible to an external processor. It is used to hold temporary data between the read and write phases of a memory-memory transfer. On this microcontroller, the data buffer is one word.

All the registers are accessed as 32-bit registers, with unused bits defined as zero for read operations. Attempts to access any register with a half-word or byte transfer result in a bus error, as do any accesses to reserved registers or writes to read-only registers.

6.3.4.1 Channel Control And Status Register (CSR)

The **Channel Control and Status Register** is made up of a 24-bit read/write register, occupying bits [23:0] and an 8-bit read-only register, occupying bits [31:24]. The lower 24 bits define the configuration of the DMA channels, and their operating behaviour (**Channel Control Register**). At Reset, these bits are set to zero. The upper 8 bits [31:24] indicate the current status and termination conditions for the DMA channel (**Channel Status Register**). These bits are ***cleared*** to zero at reset, or when the **CSR** is read (with the exception of **Write In Progress (CSR[31])**). Writing to the **CSR** has no effect on the status values.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31	Write In Progress	In dual-address mode, indicates that the read channel is idling while awaiting the completion of the write channel [1] Set	0	R	
30	Reserved		0	R	
29	Bus Halted	Indicates an error was signalled on b_error or the bus mode signals [1] Set	0	R	R
28	Reserved		0	R	
27	Chain Ended	In chained transfer mode, indicates a new transfer buffer was not programmed before the current buffer was exhausted [1] Set	0	R	R
26	Chain Empty	In chained transfer mode, indicates a new transfer buffer should be programmed into the channel base registers [1] Set	0	R	R
25	Reserved				
24	Terminal Count	Indicates the channel word count has decremented to zero, completing the transfer and halting channel operation; Cleared when read. [1] Set	0	R	R
23:19	Reserved		0	R	
18	Peripheral Location	Indicates the relationship of the peripheral to the controller [0] Internal [1] External	0	RW	
17	Bus Lock	Locks bus for duration of transfer, preventing interruptions by higher priority devices [0] Disable [1] Enable	0	RW	

Table 47 - Channel Control And Status Register (CSR)

Bit	Function	Description	Reset Value	Type	Volatile[1]
16	Interrupt Enable	Interrupt enabled when bit is set [0] Clear [1] Set	0	RW	
15	Chain Interrupt Enable	If Auto initialization is set, enables generation of an interrupt at the beginning of each transfer block so the base register values can be updated prior to the next auto initialization (i.e., chained transfer); [0] Clear [1] Set: new values for base registers should be supplied in response to interrupt to produce a chained transfer.	0	RW	
14	Auto initialization	If enabled, the current transfer is reinitialised with the current values of the base register on completion. [0] Disable [1] Enable	0	RW	
13:12	Operand Size	Determines size of data; valid values are: [0y00] Byte [0y01] Half Word [0y10] Word	0y00	RW	
11	Request Trigger Type	[0] Edge: request must return to inactive state before new transfer can be triggered [1] Level: request must remain in active state for successive transfers	0	RW	
10	Transfer Type	[0] Packet: size dictated by **Packet Size Register** or by depth of internal data buffer [1] Block: size equal to Transfer Count	0	RW	
9	Transfer Mode	[0] Single-address: transfer between memory and peripheral with peripheral implicitly addressed using hardware handshaking [1]Dual-address: transfer between memory areas with both source and destination explicitly mapped into memory	0	RW	
8	Transfer Direction	Determines whether the transfer is reading from or writing to memory [0] Read [1] Write	0	RW	
7	Address Direction	Determines how the address is modified on consecutive transfers when Address Mode is dynamic [0] Decrement [1] Increment	0	RW	

Table 47 - Channel Control And Status Register (CSR) (continued)

Bit	Function	Description	Reset Value	Type	Volatile[1]
6	Address Mode	Determines if the address is modified on consecutive transfers [0] Dynamic: address modified according to the Address Direction bit [1] Static	0	RW	
5	Hardware Acknowledge Status	Indicates how hardware acknowledgement signal is handled [0] Disable: signal remains in inactive state [1] Enable: signal set during bus cycles in which a peripheral is implicitly addressed	0	RW	
4	Hardware Acknowledge Polarity	Determines polarity of hardware acknowledge signal. [0] Active High [1] Active Low	0	RW	
3	Hardware Request Polarity	Determines polarity of hardware request signal. [0] Active High [1] Active Low	0	RW	
2	Software Request	When enabled, this bit indicates the software is requesting a transfer. The transfer will take place immediately if the "Channel Status" is currently set or will take place when "Channel Status" becomes set. [0] Disable: Software transfer request disabled [1] Enable: Software transfer request enabled; upon completion, bit is cleared.	0	RW	R
1	Hardware Request Status	Determines how a hardware request signal is handled [0] Disable: signal activity ignored [1] Enable: signal responded to in accordance with correct polarity and trigger type	0	RW	R
0	Channel Status	[0] Program: In this state, all DMA operations are halted; the DMA is programmable. [1] Ready: In this state, all DMA current registers have been programmed and the channel is idling, ready for operation.	0	RW	R

Table 47 - Channel Control And Status Register (CSR) (continued)

1. An R in this column shows the value is reset to zero whenever the register is read.

6.3.4.2 Packet Size Register (PSR)

This register defines the size, in number of data transfers, of device register queues (such as FIFOs) on implicitly addressed peripherals in single-addressing mode, thereby avoiding buffer overflow. The **Packet Size Register** should be loaded with the required packet size minus one; thus, a value of zero indicates a packet size of one, while a value of 0xFF indicates a packet size of 256. The register is ***cleared*** at reset, and may be both read and written.

Bit	Function	Description	Reset value	Type	Volatile[1]
31:8	Reserved		0	R	
7:0	Packet Size	[0x00 - 0xFF] Value equivalent to required packet size - 1.	0	RW	

Table 48 - Packet Size Register (PSR)

1. An R in this column indicates thtat the value is reset to zero whenever the register is read.

6.3.4.3 Base Transfer Count Register And Current Transfer Count Register (BTR, CTR)

The **Base** and **Current Transfer Count Registers** define the initial transfer block size, and the remaining number of data transfers, respectively. During channel initialization, the **Base Transfer Count Register** is loaded with the required transfer block size minus one; thus, a value of zero indicates one transfer, while 0xFFFF indicates 65536 transfers. This value is subsequently written through to the **Current Transfer Count Register** when the channel is enabled. During a DMA transfer cycle on that channel, the transfer count held in the **Current Transfer Count Register** is read, decremented and rewritten to the **Current Transfer Count Register**. If the current transfer count value equals zero, the transfer of data is completed with the current transaction and the channel halts subsequent transfers. Upon completion of the block transfer, the **Current Transfer Count Register** may optionally be updated from the **Base Transfer Count Register** in order to auto-initialise the channel for repetitive DMA activity. Table 49, “Base Transfer Count Register (BTR)” below describes the **Base Transfer Count Register** and Table 50, “Current Transfer Count Register (CTR)” overleaf describes the **Current Transfer Count Register**.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:16	Reserved		0	R	
15:0	Base Transfer Count	This is the initialization value for the transfer block size. [0x0000 - 0xFFFF] Value equivalent to required block size - 1	0	W	

Table 49 - Base Transfer Count Register (BTR)

1. An R in this column indicates that the value is reset to zero whenever the register is read.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:16	Reserved		0	R	
15:0	Current Transfer Count	Current value for the transfer block count. [0x0000 - 0xFFFF] Value equivalent to remaining transfer count - 1.	0x0000	R	R

Table 50 - Current Transfer Count Register (CTR)

1. An R in this column indicates that the value is reset to zero whenever the register is read.

6.3.4.4 Base Address Register And Current Address Register (BAR, CAR)

The **Base** and **Current Address Registers** define the initial and current addresses within the block of memory or I/O being transferred. During channel initialization, the **Base Address Register** is loaded with the starting address of the block; this is subsequently written through to the **Current Address Register** when the channel is enabled. During a DMA transfer cycle on that channel, the address held in the **Current Address Register** is output from the controller, simultaneously incremented or decremented as appropriate, and then rewritten to the **Current Address Register**, which then points to the next data item to be transferred. Upon completion of the block transfer, the **Current Address Register** may optionally be updated from the **Base Address Register** in order to auto-initialise the channel for repeated DMA transfers. If the size of the transfer (as indicated by the **Transfer Count Registers**) causes the **Current Address Register** to overflow or underflow the 32-bit address space, the address wraps around. Table 49 -, "Base Transfer Count Register (BTR)" and Table 50 -, "Current Transfer Count Register (CTR)" below provide details of the **Transfer Count Registers**.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:0	Base Address	This is the initialization address of the current transfer block. [0x00000000 - 0xFFFFFFFF]	0x00000000	RW	

Table 51 - Base Address Register (BAR)

1. An R in this column indicates that the value is reset to zero whenever the register is read.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:0	Current Address	This is the calculated address of the next transfer. [0x00000000 - 0xFFFFFFFF]	0x00000000	R	R

Table 52 - Current Address Register (CAR)

1. An R in this column indicates that the value is reset to zero whenever the register is read.

6.3.4.5 DMA Status Register (DSR)

The **DMA Status Register** indicates the current condition of each channel and hence, allows the programmer to rapidly determine which DMA channels require service. Each bit corresponds to one channel, with the lowest order bit corresponding to channel 1. A status bit is set in the **DMA Status Register** if interrupts are enabled for the associated channel and any bit (with the exception of **Write In Progress (CSR[31])**) is set in the **Channel Status Register** (see Table 47 -, "Channel Control And Status Register (CSR)" for more information about the **Channel Status Regiser** and Table 53 -, "DMA Status Register (DSR)" below for details of the **DMA Status Register**.

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:0	DMA Controller Status	Reflects service request status of each channel of the DMA Controller.	0x00000000	R	

Table 53 - DMA Status Register (DSR)

1. An R in this column indicates that the value is reset to zero whenever the register is read.

7.0 Timer/Counter (TIC)

7.1 Overview

The Timer/Counter module comprises a pair of identical, generic 32 bit interval timer elements, each with its own dedicated prescaler. These are controlled by the ARM® processor core via the internal bus. The Firefly MF1 Core contains two of these modules, providing a total of four timer elements.

Each prescaler is driven by the system clock, to feed one timer element. The preset time interval is generated by counting prescaler output pulses. An interrupt may be generated after the desired number of clock periods.

A pulse width modulation (PWM) mode can be configured by combining both timer elements in a Timer/Counter module together. In this mode one timer element controls the high period of an external output, with the other controlling the low period, offering the user complete control over the period and shape of the external output pulse train.

7.1.1 Design features

The main features of each Timer/Counter module are:

- Two, independently controlled Timer/Counter elements
- Prescalers generating a Timer clock from the system clock
- Prescale count of 8 bits
- Division ratio selection by software
- Multiple Timer/Counter modes:
  - Countdown to zero
  - Free-running
  - Reload and count on trigger
  - PWM waveform generation by combining the two Timer/Counter elements
- Fully software programmable time-out period
- Maskable interrupt on time-out

7.2 Architecture

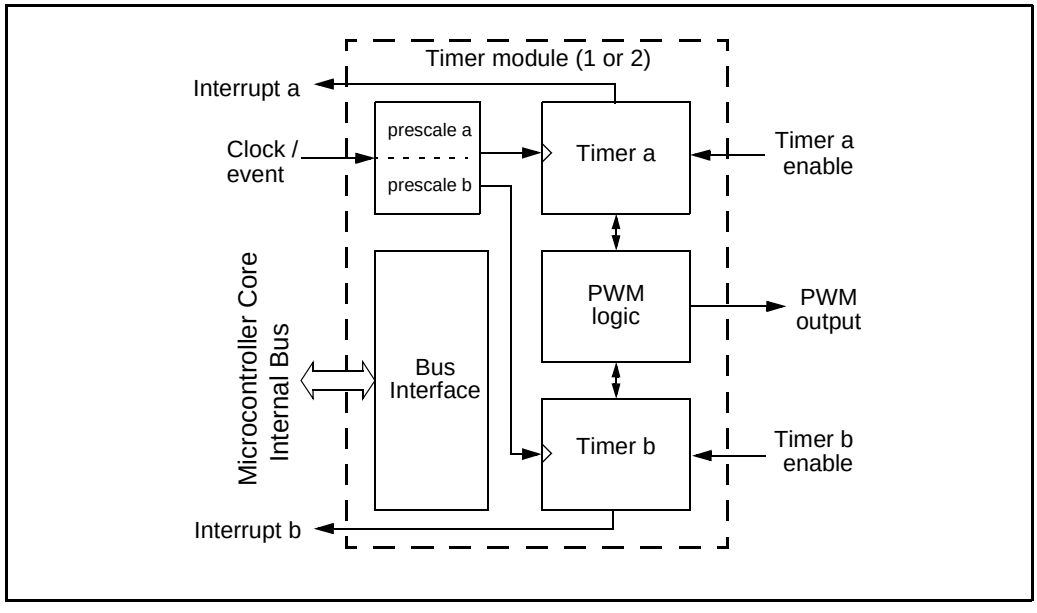


Figure 39 - Timer/Counter Functional Block And System Interconnections

7.3 Operational Description

Each Timer/Counter module consists of two independent synchronous counter elements, both 32 bits long. Each counter has an associated prescale counter, which is used to scale the input clock to its main counter. Both prescalers are fed from the internal system clock.

The counters are started by controlling the **Software Control Request** bit of the **Control and Status Register**. When a counter reaches zero the **Overflow Status** bit is set in the **Control and Status Register** and an interrupt pulse will be generated; however, this interrupt may be masked by resetting the **Interrupt Enable** bit.

Each counter has the following operating modes:

- Count-Down to zero
- Free Running
- Reload and count on trigger
- Pulse Width Modulation (PWM)

(**Note:** Only one PWM output is available from each Timer/Counter module.)

7.3.1 Prescaler Operation

The prescaler operates as an 8-bit auto-reload counter, clocked from the system clock. The prescaler continually counts down towards zero. Upon reaching zero, it outputs a clock pulse to the main counter and is reloaded with the value in the reload field of the control/status register.

The PreScaler is disabled when the **Clock PreScale Factor** bit-field is loaded with the value 0x01, and the counter counts system clock cycles.

When a value other than 0x01 is stored in the bit-field, the Prescaler counts from that value down to zero, generating a pre-scaled clock pulse as it reloads, thus dividing the system clock by the value (**PreScale + 1**). The value 0x00, when programmed into the PreScaler causes the clock to be divided by 257.

The PreScaler counts the falling clock edges and the counter is activated on the following rising clock edge; thus, all interrupts generated by the counter reaching zero are activated on the rising edge of the system clock.

7.3.2 Halt-On-Zero (Mode 0)

The counter decrements from '**0xFFFFFFFF**'. When the counter reaches zero, it is reloaded from the reload register and halts until it is retriggered. This retriggering is achieved either by a transition on the external enable, or by setting the **Software Control Request** bit under software control. On reaching zero the **Overflow Status** bit is set and a Timer interrupt may be generated.

If required, the initial count from **'0xFFFFFFFF'** may be avoided by starting the counter in Mode 2 with the required value, then stopping before it reaches zero and changing the mode to Free Running. In this case, the counter will restart in the new mode from the value at which it last stopped.

7.3.3 Free Running (Mode 1)

When the counter is initially started in Free Running mode, it starts from the value '**0xFFFFFFFF'.** Upon reaching zero the counter is reloaded with the preset value and resumes counting from that value. A Timer interrupt may be generated, note that in this mode **Overflow Status** bit will not be set and the counter will continue to run.

If required, the initial count from **'0xFFFFFFFF'** may be avoided by starting the counter in Mode 2 with the required value, then stopping before it reaches zero and changing the mode to Free Running. In this case, the counter will restart in the new mode from the value at which it last stopped in Mode 2.

7.3.4 Reload-On-Trigger (Mode 2)

The counter is reloaded when it receives a start trigger on the timer enable input or **Software Control Request** bit and then, will begin to decrement. Upon reaching zero the counter will halt, setting the **Overflow Status** bit flag and generate an interrupt (if enabled).

7.3.5 Pulse Width Modulation (PWM) (Mode 3)

One counter is reloaded and started automatically when the other counter reaches zero. Upon reaching zero the counter halts, toggles the state of the PWM output and triggers the other counter to run. This mode generates a variable width pulse train on the PWM output with counter A controlling the high phase and counter B controlling the low phase.

Proper functioning of the PWM mode requires that both counters A and B are set to mode 3. Counter B should be set to start first. If counter A is started first the Timer will not begin correct PWM operation until both counters A and B have timed out for the first time.

As each counter uses a PreScaler, which have two modes of operation, there are two algorithms for the PWM output.

```
PreScale = 0x01;
        pwmLo = RLDA + 1;
        pwmHi = RLDB +1;
PreScale <>0x01;
        pwmLo = ((PreScale+1) * RLDA) -1 ;
        pwmHi = ((PreScale+1) * RLDB) +1 ;
```

7.4 Programmer's Model

7.4.1 Register Map

Address Offset (hex)	Register Name	Access Size (bits)	Reset Value (hex)	Type	Volatile[1]	Description
Timer A						
+000	Control and Status	32	000200FF	RW		
+004	Reload Value	32	FFFFFFFF	RW		
+008	Current Value	32	FFFFFFFF	R		
+00C - +01C	Reserved	32				
Timer B						
+020	Control and Status	32	000200FF	RW		
+024	Reload Value	32	FFFFFFFF	RW		
+028	Current Value	32	FFFFFFFF	R		
+02C - +FFC	Reserved					

Table 54 - Timer/Counter Address Map

1. An R in this column indicates that a read is destructive, and a W, that a write is destructive.

Addresses are specified as offsets from the system defined base address.

All the registers are accessed as 32-bit registers, with unused bits defined as zero for read operations. Attempts to access any register with a half-word or byte transfer results in a bus error, as do any accesses to reserved registers or writes to read-only registers. In all cases, the following register descriptions apply equally to timer A and timer B.

7.4.2 Register Details

7.4.2.1 Control/Status Register (CONSTATA/CONSTATB)

The **Control And Status Register** contains the configuration information and status flags for the associated interval timer.

Bit	Function	Description	Reset value	Type	Volatile[1]
31:23	Reserved	Reserved for future use			
22	Interrupt Enable	Interrupt enabled when bit is set [0] Clear [1] Set	0	RW	

Table 55 - Control/Status Registe

Bit	Function	Description	Reset value	Type	Volatile[1]
21	Overflow Status	Set when main counter reaches zero except in free running mode. Reset by any action that restarts the counter. [1] Set	0	R	
20:19	Mode	Controls the timer operating mode [00] Halt On Zero [01] Free Running [10] Reload on Trigger [11] Pulse Width Modulation	00	RW	
18	Software Control Request	Controls the state of the main counter when hardware enable is active. Resetting this bit causes a halted counter to begin counting. Stopping a counter that is already halted or starting one that is running has no effect. [0] Halt [1] Count	0	RW	W
17	Hardware Enable Polarity	Determines the sense of the external enable input, when present. [0] Active Low [1] Active High	1	RW	W
16	Current Status	[0] Halted [1] Running : Indicates counter is enabled and running	0	R	
15:8	Reserved	Reserved for future use			
7:0	Clock Prescale Factor	Clock division ratio applied by prescaler. [0x00- 0xFF] : Values of 0x01 - 0xFF correspond to ratio values of 1 - 255. 0x00 corresponds to a ratio value of 256.	0xFF	RW	

Table 55 - Control/Status Registe (continued)

1. An R in this column indicates that a read is destructive and a W, that a write is destructive.

7.4.2.2 Counter Reload Register (RLDA/RLDB)

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:0	Counter Reload Value	This value, along with the prescaler value, determines the period of the count. [0x00000000 - 0xFFFFFFFF]	0xFFFFFFFF	RW	

Table 56 - Counter Reload Register

1. An R in this column indicates that a read is destructive and a W, that a write is destructive.

The **Counter Reload Register** holds the 32-bit reload value for the main counter which, in conjunction with the prescaler's reload value, determines the period of the count. Thus the interval timed can be calculated as:

$$T_{timer} = \frac{ReloadValue \times (PreScale + 1)}{f_{BCLK}}$$

BCLK is the internal module bus clock of the Firefly MF1 Core which is derived from SCLK. At Reset, the initial value is 0xFFFFFFFF.

7.4.2.3 Counter Read Register (RDA/RDB)

Bit	Function	Description	Reset Value	Type	Volatile[1]
31:0	Current Counter Value	This value, is the current value of the counter. [0x00000000 - 0xFFFFFFFF]	0xFFFFFFFF	R	R

Table 57 - Counter Read Register

1. An Rin this column indicates that a read is destructive and a W, that a write is destructive.

The **Counter Read Register** holds the current 32-bit count value of the counter. At Reset, the initial value is 0xFFFFFFFF.

8.0 Software Debug Facilities

8.1 Overview

Traditional application software development for embedded systems has often concentrated on the use of in-circuit emulation techniques (ICE). This worked well when used with a stand-alone microprocessor, since all the pins were available and the microprocessor could simply be unplugged and replaced by the ICE header.

When the microprocessor is buried inside the chip, this technique requires an expensive bond-out chip with the pins of the microprocessor brought to the surface. This may compromise the operation of the chip at speed and is often not practical for a typical project. The required capital investment to develop high speed analysis and debugging systems based on traditional in-circuit emulation techniques has risen dramatically, while the target market is demanding cheaper and more flexible tools. Further-more, system integration is increasingly performed at higher levels of the design, through analysis and debugging of high level language source code and real time kernel calls.

Zarlink microcontrollers incorporate the following two methods of hardware and software debug:

1. Diagnostic Broadcast, coupled with a logic analyser and inverse assembler; and
2. ARM7TDMI™ Debug Interface, via the EmbeddedICE™ module.

These two methods are described in this chapter.

8.2 Diagnostic Broadcast

The use of Diagnostic Broadcast to provide support for logic analyzer based debug is defined in this section. Diagnostic Broadcast makes information about the state of the internal bus available in the external environment, without compromising the performance of the system. The information includes the current address and data, cycle type (e.g., memory or instruction fetch, size of access) and current bus master.

Broadcast information can be viewed using a logic analyser. The state capabilities of the logic analyser can be used to view information about the transactions in progress on the bus. When coupled with an Inverse Assembler for the ARM7TDMI it becomes a powerful debugging tool, providing a real-time trace of the instructions executed over the past few cycles. The trace can be triggered in several different ways (e.g., on specific addresses, data values, or particular control signals.) This provides a very flexible way to trap and analyse a given event or circumstance.

Data transmission to the external environment is via a diagnostic bus, which consists of 32 data bits and 4 status bits. Broadcast occurs only during internal bus operations, since the address and the data are inherently visible during external accesses. However, the status bits are still generated, indicating what type of transfer is taking place.

8.2.1 Modes Of Broadcast

The mode of operation of the diagnostic bus dictates the nature of the information broadcast. Four modes are supported:

1. Address broadcast,
2. Data broadcast,
3. Address and data broadcast (“Full broadcast”) and
4. Broadcast disabled.

Each active broadcast mode supports a different view of the internal operation:

- address broadcast indicates program flow in real time;
- data broadcast indicates information flow in real time; and
- address and data broadcast indicates both program and information flow.

Ideally, both address and data information will be broadcast. However, if the current memory access is non-sequential and full broadcast has been requested, then two cycles will be required to broadcast bus activity. In such cases, the non-sequential access will be stalled for one cycle. This reduces the performance available from the system, which in some applications, may be unacceptable. If performance loss is not acceptable during diagnostic broadcast, the operation of the system must be modified to use partial traces of the processor behaviour.

In order to reduce pin count, the broadcast data port is normally multiplexed with the external address and data busses. No standard is enforced on which pins are used, since it will depend on the widths of the address and data busses on the particular chip. The only restriction is that 32 pins are assigned somewhere to broadcast the information. Broadcast will normally use the same pins as the test interface (**TBUS[31:0]**).

8.2.2 Diagnostic Status Information

In addition to address and data transmission, there are four diagnostics bits (**B_DIAG(3:0)**) that will broadcast a coded representation of the Internal bus control/status signals together with the identity of the current bus master. The two types of information are time multiplexed over a bus clock cycle, as shown in <italic>Figure 40 Bus Cycle And Bus Master Diagnostic Broadcast on <italic>page 107:

- BCLK low: bus master information
- BCLK high:cycle coding information

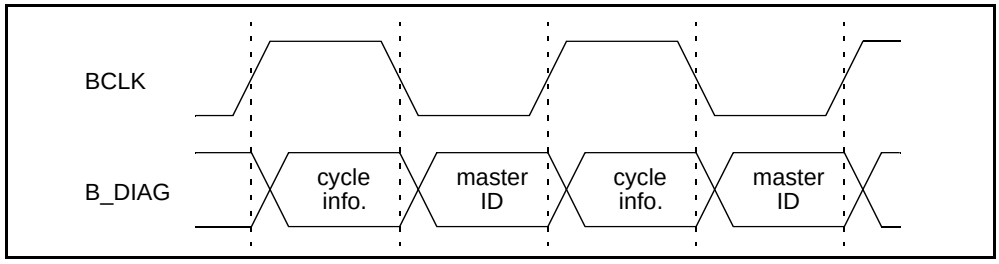


Figure 40 - Bus Cycle And Bus Master Diagnostic Broadcast

8.2.2.1 Master ID Information

In a multi-master system, it is valuable to know which module is the current bus master. This knowledge allows more complete interpretation of the available information. During the low phase of **BCLK**, the **B_DIAG** signals indicate the bus master which initiated the current transaction. This is encoded as a number representing the priority of the bus master; hence, the highest priority master is zero (0000). There are two exceptions to this encoding rule: Code 14 (1110) and Code 15(1111).

Code 14 is used to indicate that broadcast information has been lost, which can occur if internal bus activity continues while the external bus is still in use. Internal bus activity also is allowed to continue during stop states, which are used to allow time for slow memories to turn off after reading and therefore, to prevent bus clashes. If an external memory access is requested during a stop state, wait states are inserted until the stop state has completed (and thus, the memory read has completed). If broadcast was not prevented during such a stop state, a bus clash could occur between the broadcast signals and the remaining memory signals.

Code 15 is used to indicate that two clock cycles will be required to broadcast bus activity. One situation in which two cycles will be required is if the current memory access is non-sequential and full broadcast has been requested. In this example, Code 15 would be broadcasted during the first (address) cycle and the master ID code would be broadcasted during the second (data) cycle. The encoding for the bus master ID is shown in Table 58, “Encoding Of Bus Master ID For Diagnostic Broadcast” below.

Bus Master / Condition	Diagnostic Code
System Services Module	0y 0000
DMA	0y 0001
Spare	0y 0010
ARM7TDMI	0y 0011
Broadcast Information Lost	0y 1110
Bus Hold Cycle	0y 1111
Reserved	Remaining codes

Table 58 - Encoding Of Bus Master ID For Diagnostic Broadcast

8.2.2.2 Cycle Information

During the high phase of bclk, bdiag indicates what type of cycle is currently in progress. This encoding is described in Table 59, “Encoding Of Cycle Type For Diagnostic Broadcast” below.

There are two codes for each instruction fetch code, representing the two possible states of the instruction pipeline (executed or not executed). The executed/not-executed information indicates whether the instruction at the execute stage of the processor’s pipeline is being executed. This does not refer to the data that is present on the data bus, which would typically appear two instructions (or the length of the pipeline) later.

The execution information permits program flow to be re-created externally (e.g., by a logic analyser). An ARM-state (32-bit) instruction may not execute because it contains a conditional execution bitfield and the correct conditions for execution to occur may not have been met. All other cases will result in the broadcast data indicating that the instruction has been executed.

It should be noted that when the pipeline is flushed, due to the program flow being changed, the instructions that are already present within the pipeline are changed to NOP (No OPeration) instructions, which are still executed.

Code	Cycle Type	Code	Cycle Type
0000	8-bit data fetch	1000	32-bit data fetch
0001	8-bit data write	1001	32-bit data write
0010	8-bit Instruction fetch - executed	1010	32-bit Instruction fetch - executed
0011	8-bit Instruction fetch - not executed	1011	32-bit Instruction fetch - not executed
0100	16-bit data fetch	1100	Reserved
0101	16-bit data write	1101	Non-memory cycle
0110	16-bit Instruction fetch - executed	1110	Wait cycle
0111	16-bit Instruction fetch - not executed	1111	Reserved

Table 59 - Encoding Of Cycle Type For Diagnostic Broadcast

8.2.3 Diagnostic Configuration Registers

One configuration register is used to control the generation of diagnostic broadcast information. This register is summarised in Table 60, “Diagnostic Configuration Register” below. Diagnostic Broadcast is generated by the System Services Module (SSM) and the offset shown is from the base address of the SSM.

Address Offset (hex)	Register Name	Size (bits)	Reset Value (hex)	Type
+000	Diagnostic Configuration Register	32	0000	RW

Table 60 - Diagnostic Configuration Register

The **Diagnostic Configuration Register** is used to control the broadcast of diagnostic information through the external bus interface. These options are summarised in Table 61, “Diagnostic Configuration Register Details” and discussed in sections Table 8.2.3.1, “Level Of Broadcast Information” and Table 8.2.3.2, “Action On Stop States” below.

Bit-field	Function	Description	Reset Value	Type
1 : 0	Broadcast Information	0y 00 None 0y 01 Address Only 0y 10 Data Only 0y 11 Address and Data	0y 00	RW
2	Stop State Action	0y 0 Halt: internal bus 0y 1 Continue: internal bus activity	0y 0	RW

Table 61 - Diagnostic Configuration Register Details

8.2.3.1 Level Of Broadcast Information

The level of broadcast information, as described in section <italic>8.2.1, "Modes Of Broadcast” on <italic> page 106, is controlled by the **Broadcast Information** field.

When no broadcast is selected (**Broadcast Information** = 0y00), both diagnostic data and status information are turned off and their output ports set to high-impedance, preventing data clashes with any multiplexed function and eliminating unnecessary power consumption.

8.2.3.2 Action On Stop States

When using slow external memory, the turn-off time of the memory can be greater than the half cycle turnaround time allowed for by the internal bus protocol. Therefore, it can be necessary to insert wait states after an external memory read to avoid contention on the external bus. These are known as “Stop-states”. Internal bus activity may continue while Stop-states are occurring on the external bus.

Diagnostic trace data is normally multiplexed over the external data and/or address buses; therefore, it must be disabled when the external bus is required for other purposes (i.e., any external memory access, or during a stop state.) There are two courses of action which may be taken during stop states:

1. disable internal bus activity until Stop-states have finished; or
2. continue internal bus activity, leaving a “hole” in the broadcast trace.

These options may be selected using control bits within the Diagnostic Broadcast Controller.

If internal bus activity has not been halted as a consequence of a Stop-state on the external bus (i.e. bit 2 = 1, and bits 1:0 are non-zero), then the reserved code 1110 will be broadcasted in place of the master identity to indicate that there is a hole in the broadcast.

8.3 ARM7TDMI™ Debug Interface

8.3.1 Overview

The ARM7TDMI™ processor contains hardware extensions for advanced debugging features. These are intended to ease the user’s development of application software, operating systems and the hardware itself. The features are supplemented by a debug controller in the System Services Module, providing the ability to control the state of the whole system during a debug session.

The processor's debug extensions allow it to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint), or asynchronously by a debug-request. When this happens, the processor is said to be in *debug state*. At this point, the processor's internal state and the system's external state may be examined. Once examination is complete, the core and system state may be restored and program execution resumed.

The processor can be forced into debug state one by three methods:

1. a request on one of the external debug interface signals (i.e., a hardware event);
2. executing a given instruction fetch (i.e., a breakpoint); or
3. accessing the address of a specific data item (i.e., a watchpoint).

Once in debug state, the processor isolates itself from the memory system. The internal state of the processor can then be examined without affecting the rest of the system. This can be done via a JTAG serial interface, which allows instructions to be serially inserted into the processor's pipeline without using the external data bus. Thus, when in debug state, a store-multiple (STM) could be inserted into the instruction pipeline and this would dump the contents of ARM7TDMI's registers. This data can be serially shifted out via the JTAG port.

The System Services Module can be used to control the state of the rest of the system during a debug state. The default behaviour is for the rest of the system to continue activity as normal. However, it is often required to stop system activity when debug state is entered, to prevent the conditions that caused the debug entry from being destroyed before they can be determined. The SSM can be used to freeze the internal state of a macrocell, while leaving its register interface operational, in order to allow its state to be determined as required.

8.3.1.1 Debug System

The ARM7TDMI forms one component of a debug system that interfaces from the high-level debugging performed by the user to the low-level interface supported by ARM7TDMI. The debug system has three parts, shown in <italic>Figure 41 ARM7TDMI™ Debug System on <italic>page 111.

1. The Debug Host

   The *Debug Host* is a computer (for example, a PC) running a software debugger such as ADW, the ARM® Debugger for Windows. The debug host allows the user to issue high level commands such as "set breakpoint at location XX", or "examine the contents of memory from 0x0 to 0x100".

2. The Protocol Converter

   The Debug Host will be connected to the ARM7TDMI development system via an interface (RS232, for example). The messages broadcast over this connection must be converted to the interface signals of the ARM7TDMI and this function is performed by the *Protocol Converter*.

3. Debug Target

   The *Debug Target* is the development system containing the ARM7TDMI core, which has hardware extensions to ease debugging and is the lowest level element of the system. The debug extensions allow the user to stall the core from program execution, examine its internal state and the state of the memory system and then resume program execution.

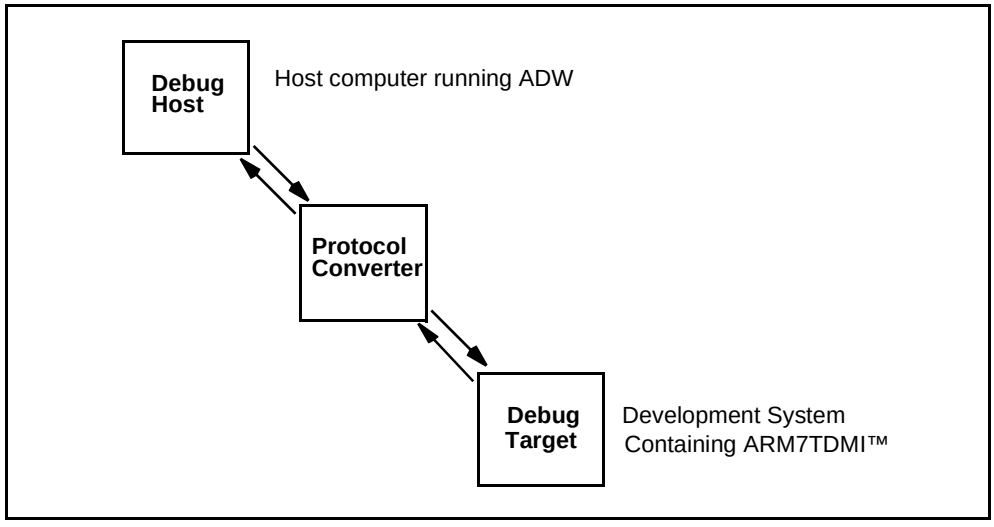


Figure 41 - ARM7TDMI™ Debug System

The Debug Host and the Protocol Converter are system dependent. The rest of this chapter describes the facilities provided on the debug target system.

8.3.1.2 EmbeddedICE™ Module

The ARM7TDMI™ can be broken down into essentially three major blocks, shown in <italic>Figure 42 EmbeddedICE™ Module on <italic>page 112:

1. Processor: the CPU core, with hardware support for debug;
2. EmbeddedICE: a set of registers and comparators used to generate debug exceptions (e.g., breakpoints); and
3. TAP controller: controls the action of the scan chains via a JTAG serial interface.

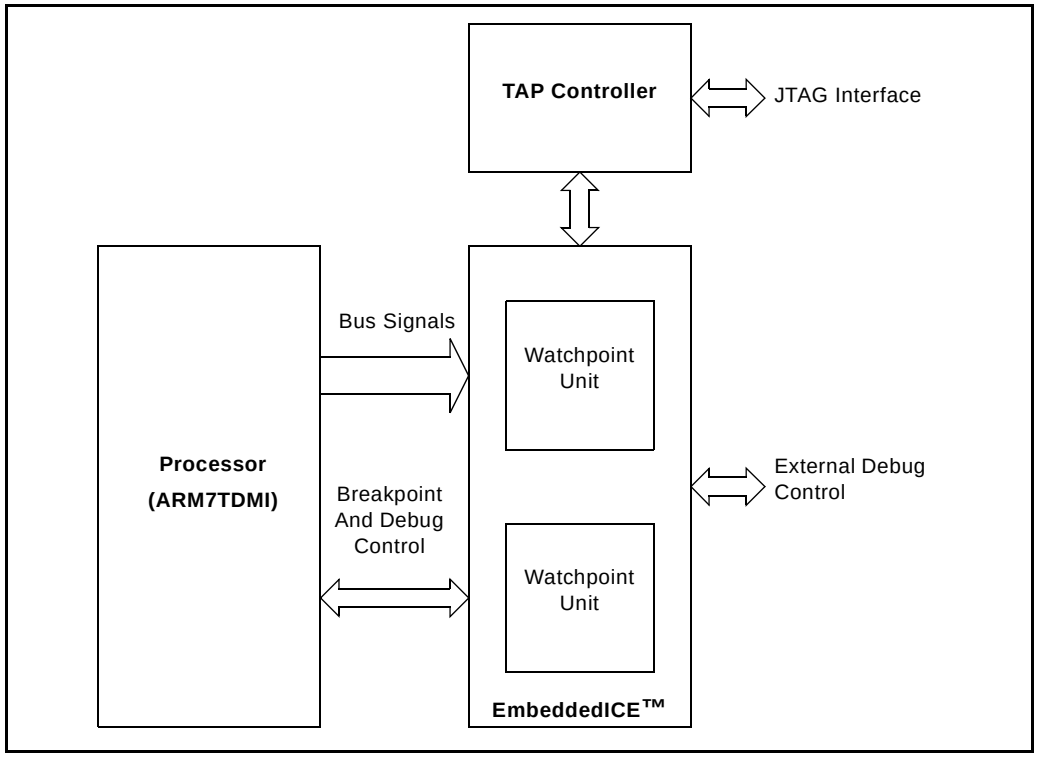


Figure 42 - EmbeddedICE™ Module

The ARM7TDMI™ EmbeddedICE™ module, hereafter referred to simply as EmbeddedICE, provides integrated on-chip debug support for the ARM7TDMI core. It consists of two real-time watchpoint units, together with a control and status register. One or both of the watchpoint units can be programmed to halt the execution of instructions by the ARM7TDMI core. Execution is halted when a match occurs between the values programmed into EmbeddedICE and the values currently appearing on the address bus, data bus and various control signals. Any bit can be masked so that its value does not affect the comparison.

The watchpoint modules are programmed via a set of registers accessible over the JTAG scan chain interface. Either watchpoint unit can be configured to be a watchpoint (monitoring data accesses) or a breakpoint (monitoring instruction fetches). Watchpoints and breakpoints can be made to be data-dependent. Two independent registers, **Debug Control** and **Debug Status**, provide overall control of the EmbeddedICE's operation.

8.3.1.3 System-level extensions

The Debug Controller in the System Services Module provides a set of extensions to the control offered by the ARM7TDMI Debug Interface. The extensions are designed to provide additional control over the system and also to handle the extra control signals provided by the EmbeddedICE™ module.

The functions provided by the SSM include:

- hardware debug triggers, with individual enables;
- software debug trigger and exit;
- individual control of macrocell activity; and
- debug mode indicator to system.

The SSM provides a number of hardware trigger inputs. These triggers can be individually enabled by setting a bit in the debug configuration register. Hence, different hardware events can be selected to force the processor into its debug state.

The SSM also provides the ability to enter into system-level debug state from software, without forcing the ARM7TDMI into debug state. Hence, execution of code will continue from ROM rather than EmbeddedICE. Therefore, the SSM provides the facility for a ROM resident debugger (e.g., Angel™) to emulate the facilities provided by the EmbeddedICE debugger.

Software entry is achieved by writing a key value into the **Debug Control Register**. The value must match the predetermined key for debug entry to take place, in order to prevent erroneous access by an accidental write to the register location. The same register can be used to exit debug mode, either returning immediately to normal operation, or via reset.

When debug mode has been entered, the SSM can be used to control the state of the non-processor part of the system. The default behaviour is for the system to continue activity as normal. However, it is often required to stop system activity when debug mode is entered, to prevent the conditions that caused the debug entry from being destroyed before they can be determined.

The SSM has a number of outputs that can be used to freeze the internal state of an individual macrocell, while leaving its register interface operational, in order to allow its state to be determined as required. These outputs are controlled by bits in the **Debug Configuration Register**.

Finally, the SSM generates signals to other parts of the system indicating that debug has been entered. Separate signals may be generated, allowing independent control of each part of the system, which is useful for co-operative debug sessions in multi-processor systems.

8.3.2 Using The Arm® Debug Interface

The use of the ARM Debug Interface is described in Chapter 8 of the **ARM7TDMI Technical Reference Manual** (Rev 4) (ARM Publication number DDI 0210 A, downloadable as a PDF file from the ARMl Website http://www.com/arm/documentation?OpenDocument). The EmbeddedICE is described in Chapter 9 of the same handbook, which explains how to set up the watchpoint units. Their use will not be further explained here.

8.3.3 System Level Debug Control Registers

The SSM Debug Controller contains two registers. These registers are summarised in Table 62, “System-level Debug Control Registers” below and described in the following subsections.

Address Offset (hex)	Register Name	Size (bits)	Reset Value (hex)	Type
+008	Debug Configuration Register	32	0000 0000	RW
+00C	Debug Control Register	32	0000 0000	W

Table 62 - System-level Debug Control Registers

8.3.3.1 Debug Configuration Register

The **Debug Configuration Register** is separated into 4 sections, as shown in Table 63, “Debug Configuration Register” below.

Bit-field	Function	Description	Reset Value	Type
3 : 0	Debug Trigger Enable	0y 0 Disable: Debug Trigger 0y 1 Enable: Debug Trigger	0y 0000	RW
7 : 4	Debug Trigger Status	0y 0 Inactive: Debug Trigger 0y 1 Active: Debug Trigger	0y 0000	R
11 : 8	External Debug Enable	0y 0 Disable: ext. debug signal 0y 1 Enable: ext. debug signal	0y 0000	RW
31 : 12	Macrocell Debug Enable	0y 0 Disable: macrocell activity 0y 1 Enable: macrocell activity	0x 00000	RW

Table 63 - Debug Configuration Register

The SSM can have up to four separate inputs for requesting debug state entry. They can be connected to different areas of the system, or to other processors in the system for co-operative debug applications.

The **debug trigger enable** section permits selective enabling and disabling of external debug trigger inputs. A reset value of zero disables a trigger input.

The **debug trigger status** section permits the debug system, through the ARM processor, to determine the source of an externally triggered entry into DEBUG mode. The implemented width of this section will be the same as the debug trigger enable section, with one bit for each trigger input.

The SSM can also have up to four outputs, indicating that the local sub-system has entered debug state. These outputs are used for communicating this status information to external recipients, as is required to synchronise entry into/exit from debug state/mode between multiple processors.

The **external debug enable** section enables independent control of whether or not debug status is to be indicated to an external device. For example, in a multiple processor system, it may be required to halt processor A, but not processor B. A reset value of '0' will mean that a signal connected to this section will not change when debug mode is attained.

The **macrocell debug enable** section enables independent control of internal clocking within individual functions when the system is in debug mode. These bits are connected to the macrocells as shown:

- bit (12) - DMA
- bit (13) - TIC1
- bit (14) - TIC2
- bit (15) - UART
- bit (16-19) - Spare but available as hit points on the cell
- bit (20-31) - Reserved

Writing a 1 to these bits starts the macrocell again, writing a zero stops it. The other macrocells (MPC, Interrupt Controller and SSM) do not need to be stopped by this mechanism.

All other bits are unused (write 0 for compatibility).

8.3.3.2 Debug Control Register

The **Debug Control Register** provides a software method of putting the bus into debug mode, without putting the processor core into debug. Hence, execution of code will continue from ROM, rather than EmbeddedICE™. The register therefore provides the facility for a ROM resident debugger (e.g., Angel™) to emulate the facilities provided by the EmbeddedICE™ debugger. This address location, when written to from the internal bus, results in transitions between RUN and DEBUG modes. In addition, if the register is written from DEBUG mode, the Bus Mode Controller may generate a number of cycles in RESET mode, before returning to either RUN or DEBUG modes.

In order for any write to this location to take effect, it will be necessary for **b_data** to contain a special eleven bit key. The bus is put into debug if the value 0x6BA is written to this register, with bit 0 of the register = 1. The register can also force the bus into RESET by setting bit 0 = 0. The meaning of the bits in the **Debug Control Register** is shown in Table 64, “Debug Control Register” below.

Bit-field	Function	Description	Type
0	Mode Entry	0y 0 Enter Run Mode 0y 1 Enter Debug Mode	W
4 : 1	Reset Cycles	Number of reset cycles before entering new mode	W
15 : 5	Debug Entry Key	11 bit key value	W
31 : 16	Reserved		W

Table 64 - Debug Control Register

The **Debug Control Register** requires that macrocells be designed to respond to RST mode in the same way that they do to POR mode. An exception is the processor core, which will ignore RST mode when it is in its Debug state. Some parts of the Debug and Diagnostic Broadcast will also not respond to RST mode. Within the internal bus interface, only the **Debug Enable Register** falls into this category.

8.3.4 Entry Into Debug Mode

Following a debug request, there will always be a delay of at least one clock cycle before system-wide debug state can be attained. It is necessary for the all bus masters (with the exception of the processor) to relinquish the bus before the state of the system can be frozen. For this reason, there may be a delay of several cycles, while the arbiter waits for the current bus activity to complete.

9.0 Appendix A: Firefly MF1 Core Datasheet

9.1 Introduction

This Appendix contains the datasheet for the Firefly MF1 Core, implemented in 0.35 micron CMOS, which is provided as one of the library 'views' whenever the Core is delivered to a user.

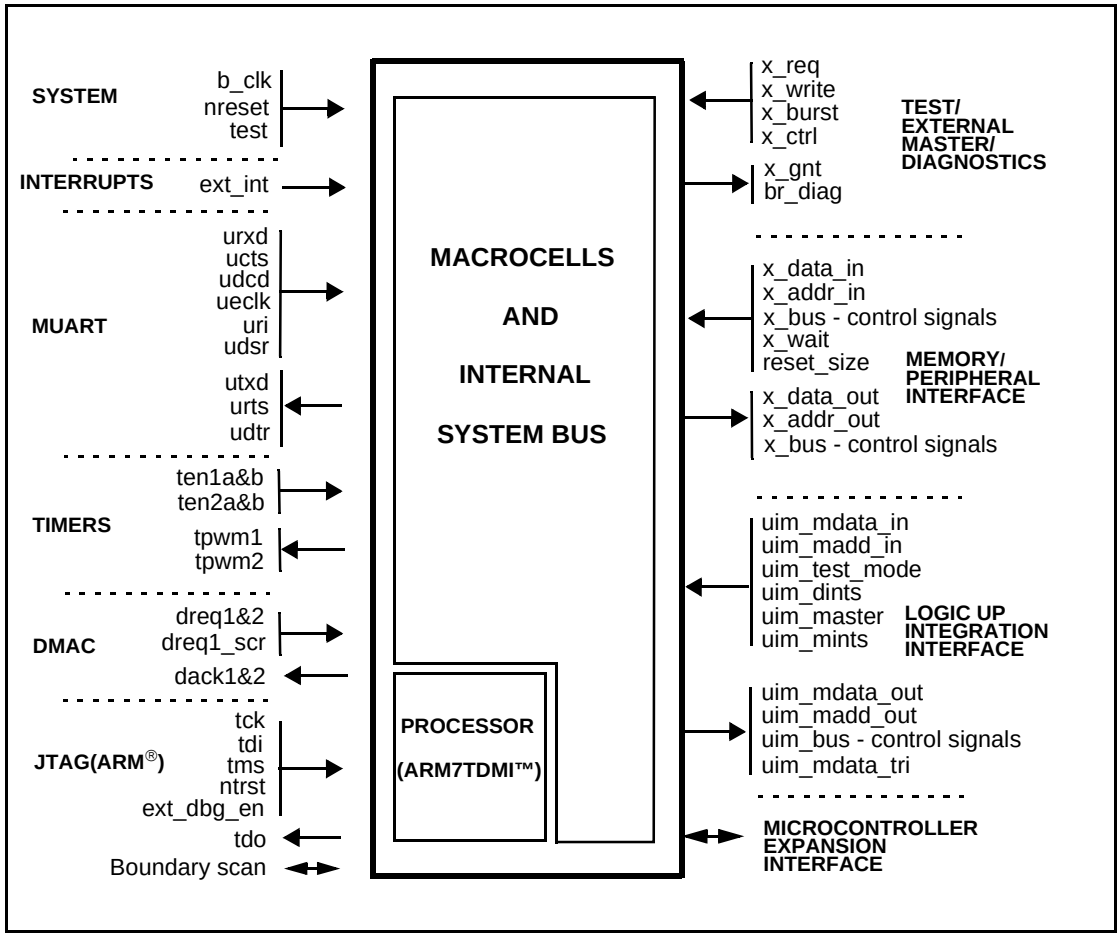


Figure 43 - Block Diagram

9.2 General Description

The Firefly MF1 Core consists of an ARM7TDMI™ core, a purpose built system bus and a number of peripherals, which combine to create a powerful microcontroller. The Firefly MF1 Core can be embedded and treated just the same as any other library component in a standard base array. The cell includes additional circuitry which facilitates easy integration of Customer logic, which may have been developed in discrete hardware, without any redesign. It also includes an expansion interface which permits new macrocells to be added to enhance the Microcontroller core. These unique features satisfy the time to market needs of Customers in the emerging embedded processing market.

9.3 Features

The **Firefly MF1 Core** incorporates an ARM7TDMI™ processor core, the industry's 32-bit processor leader in performance efficiency (MIPS/Watt), to which Zarlink has added:

- A System Services Module (SSM) which controls bus functions and provides diagnostic facilities.
- A programmable memory interface (MPC)
  - Support for 8,16 and 32-bit data transfers and memory widths
  - A flexible address interface – providing 4 memory areas, each of 4 MBytes
- A Serial I/O port (UART)
- A flexible 32-channel interrupt controller with programmable priority (INTC)
- A Direct Memory Access (DMA) controller providing 2 fly-by channels or 1 memory to memory channel (DMAC)
- Four 32-bit timer/counters (TIC)
- An easy route to facilitate the connection of additional application specific logic through the Up Integration Module (UIM) (patent pending)
- Choice of debug options
- Built in test modes to ease the production of manufacturing test patterns
- MicroController Expansion Interface to permit customization of Microcontroller core.

9.4 Description of Operation

This cell data sheet is designed to give details of port names and timings. It should be used in conjunction with the **Firefly MF1 Core Design Manual** (publication DM5003), which contains full programmable operation details.

CELL DESCRIPTION	**ARM7TDMI™** based microcontroller including: 1x**UART**, 2x**Timer/counters**, 2x**DMA** channels, 32x**Interrupt channels**, **Memory Controller**, **Up-Integration port** and **Micro controller Expansion Interface.**
VERILOG CALL LINE	**FIREFLY1** (B_CLK, NRESET, X_REQ, X_WRITE, X_BURST, X_CTRL, TEST, X_GNT, BDIAG, TCK, TDI, TMS, NTRST, EXT_DBG_EN, TDO, ID_REG, SDOUTBS, COMMRX, COMMTX, IR, NENOUTI, NM, RANGEOUT0, RANGEOUT1, TAPSM, TBIT, BUSDIS, DRIVEBS, DBGRQI, ECLK, ECAPCLK, ECAPCLKBS, HIGHZ, ICAPCLKBS, NHIGHZ, NTDOEN, PCLKBS, RSTCLKBS, SCREG, SDINBS, SHCLKBS, SHCLK2BS, TCK1, TCK2, X_ADDR_IN, X_DATA_IN, X_NCS_IN, X_NOE_IN, X_NWE_IN, X_NUB_IN, RESET_SIZE0, X_WAIT, X_NUB_OUT, X_ADDR_OUT, X_DATA_OUT, X_NCS_OUT, X_NOE_OUT, X_NWE_OUT, X_TRI_DATA_OP, X_TRI_DATA_IP, X_TRI_ADDR_OP, X_TRI_CTRL_OP, UIM_TEST_MODE, UIM_MINTS, UIM_MDINTS, UIM_MASTER, UIM_MDATA_IN, UIM_MADD_IN, UIM_MDATA_OUT, UIM_MADD_OUT, UIM_NMCS, UIM_NMWE, UIM_NMOE, UIM_NMUB, UIM_MDATA_TRI, DREQ1, DACK1, DREQ2, DACK2, DREQ1_SRC, UTXD, URXD, URTS, UCTS, UDCD, UECLK, URI, UDSR, UDTR, DACKTBE, DACKRBF, TEN1A, TEN1B, TEN2A, TEN2B, TPWM1, TPWM2, EXT_INT, B_MODE, A_REQ_SPARE, A_GNT_SPARE, B_ADDR, B_SIZE, B_WRITE, B_DATA, NCS, INT_CLKEN, B_WAIT, B_ERROR, B_REQ, B_SEQ, B_SUPER, B_OPC, B_LOCK, B_HOLD)
NOTES	This cell contains a number of programmable macrocells. Reference should be made to earlier chapters of this Manual (DM5003) for full details of their functionality
WARNINGS	Certain ports **MUST** be bonded out, as they are used for manufacturing test. - see PORT DESCRIPTIONS for further details.

Table 65 - Cell description table

9.5 Cell Parameters (General)

(One Load Unit = 15 fF)

Parameters	Pin	Value	Units
Input Loading - TYP (based on 2 i/p loads and ave. 150 fF internal track load from cell hit point to gate input)	all inputs	180	fF
Drive Capability (based on BUFX3 and ave. 150 fF internal track load from gate output to cell hit point)	all outputs	91	Load Units

9.6 Cell Integration Details

The **Firefly MF1 Core** is designed to be placed within a CLA200 base array to create a complete ASIC.

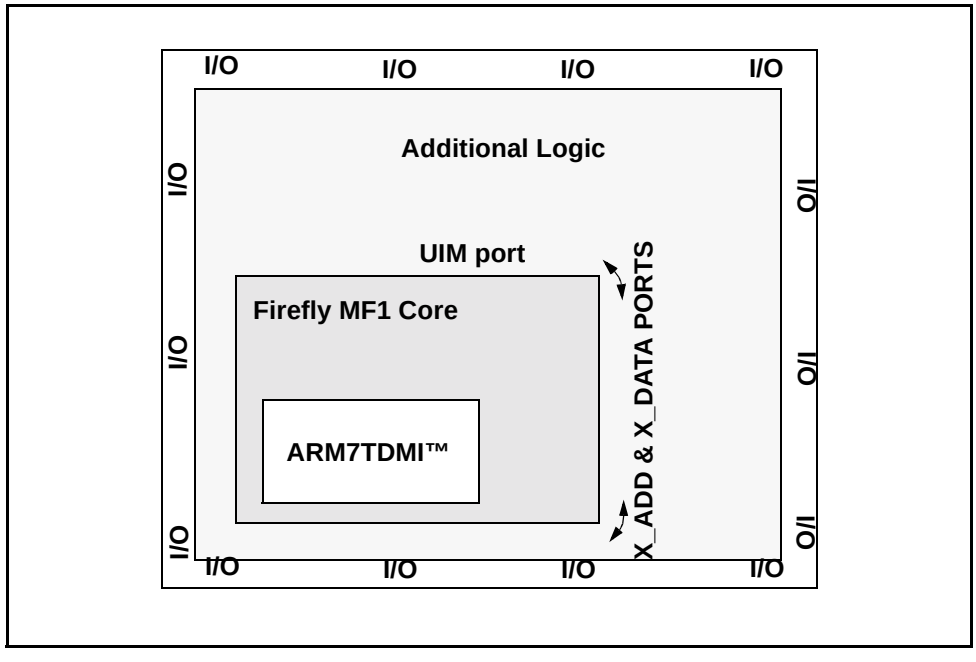


Figure 44 - Example Of Cell Placement On A Gate Array Base

9.6.1 Cell Area

The following tables show the approximate equivalent area of the Firefly MF1 Core and some example free cell locations once the cell has been placed on a gate array base. NOTE: Not all gate array bases have been detailed and reference should be made to *Section 1.5* of the **CLA200 CMOS Gate Arrays Design Manual** (DM4805) for other variants.

Rows (y)	172 (1927um)
Columns (x)	500 (2800 um)
Cell Locations	86000

Table 66 - MAIN Region

Rows	262
Columns	526
Cell Locations	137812
Number of pads	148
Available cell locations following integration	51812
See the CLA200 ASIC Handbook (Volume One) for details of other arrays	

Table 67 - ARRAY CLA206

Rows	374
Columns	750
Cell Locations	280500
Number of pads	212
Available cell locations following integration	194532
See the CLA200 ASIC Handbook (Volume One) for details of other arrays	

Table 68 - ARRAY CLA209

9.6.2 Port Descriptions

The following table details all the ports on the Firefly MF1 Core in terms of name, direction, active level (if applicable) and a brief functional description. The table indicates which ports MUST be bonded to external pins in order that manufacturing test and various methods of debug operate correctly (see section 2.9, “Embedded Microcontroller Debug Options” of the **Firefly MF1 Core Design Manual**, publication DM5003). It also lists ports whose bonding are optional. See the guide **System Design using the Zarlink Firefly MF1 Cell** for additional information (please contact your nearest Zarlink Design Centre for a copy of this document.)

Port Name	Type	Function	Bond out option
SYSTEM			
b_clk	I	Micro controller clock	Required
nreset	I	0 = Micro controller Reset	Required
test	I	1 = Micro controller Cell Test	Required
INTERRUPTS			
ext_int<2:1>	I	External interrupts - Fully programmable.	Req’d for manf test
ext_int<24:3>	I	Additional (spare) External interrupts - Fully programmable.	Optional; tie low for manf. test

Table 69 - Port Descriptions

Port Name	Type	Function	Bond out option
MUART			
utxd	O	UART Transmit Data	Req'd for Angel™ monitor & man test
urxd	I	UART Receive Data	Req'd for Angel™ monitor & man test
urts	O	1 = Ready to send	Req'd for Angel™ monitor & man test
ucts	I	1 = Clear to send	Req'd for manf. test
udtr	O	1 = Data terminal ready modem signal	Req'd for manf. test
udsr	I	1 = Data set ready modem signal	Req'd for manf. test
udcd	I	1 = Data carrier detect modem signal	Req'd for manf. test
ueclk	I	External clock	Req'd for manf. test
uri	I	1 = Ring indicator modem signal	Req'd for manf. test
dacktbe	I	1 = Transmit buffer empty (for DMA flyby)	Optional
dackrbf	I	1 = Receive buffer full (for DMA flyby)	Optional
TIMERS			
ten1a,b	I	1 = Timer1 enabled	Req'd for manf. test
ten2a,b	I	1 = Timer2 enabled	Req'd for manf. test
tpwm1,2	O	Pulse width modulation signals	Req'd for manf. test
DMAC			
dreq1,2	I	DMA Request (Programmable levels)	Req'd for manf. test
dack1,2	O	DMA Acknowledge (Programmable levels)	Req'd for manf. test
dreq1_src<3:0>	I	DMA (channel1) Request sources routed via System Configuration register.	Optional; tie low for manf. test
JTAG(ARM®)			
tck	I	JTAG: Test clock	Req'd for Embed'dICE
tdi	I	JTAG: Test data input	Req'd for Embed'dICE
tms	I	JTAG: Boundary scan test Mode Select	Req'd for Embed'dICE
ntrst	I	JTAG: Boundary scan reset	Req'd for Embed'dICE
tdo	O	JTAG: Boundary scan data output	Req'd for Embed'dICE
ext_dbg_en	I	1 = JTAG debug enabled	Req'd for Embed'dICE
TEST / EXTERNAL MASTER / DIAGNOSTICS			
bdiag<3:0>	O	Indicates current internal bus status (see Handbook for details)	Req'd for Logic Analyser
x_req[1]	I	1 = External Master - Bus request	Req'd for manf. test
x_gnt[1]	O	1 = External Master - Bus granted	Req'd for manf. test
x_write[1]	I	1 = Transaction type is an internal Write.	Req'd for manf. test
x_burst[1]	I	1 = Data read or write transaction	Req'd for manf. test
x_ctrl[1]	I	Used in conjunction with x_burst	Req'd for manf. test

Table 69 - Port Descriptions (continued)

Port Name	Type	Function	Bond out option
MEMORY / PERIPHERAL INTERFACE			
x_data_in<31:16>	I	Upper 16 bit data in	Optional
x_data_in<15:0>[1]	I	Lower 16 bit data in (OR Lower 16 bit test/external master data in)	Req'd for manf. test
x_addr_in<21:16>	I	Upper address in	Optional
x_addr_in<15:0>[1]	I	Lower address in (OR Upper 16 bit test/external master data in)	Req'd for manf. test
x_ncs_in<3:0>	I	Chip select inputs (UIM test mode only)	Req'd for UIM logic test
x_nwe_in<3:0>	I	Write enable inputs (UIM test mode only)	Req'd for UIM logic test
x_noe_in	I	Output enable input (UIM test mode only)	Req'd for UIM logic test
x_nub_in	I	Upper Byte select input (UIM test mode only)	Req'd for UIM logic test
x_wait	I	1 = External Wait state requested	Optional
reset_size0<1:0>	I	External Memory [nscs(0)] - reset size: 00 - 8 bit 01 - 16 bit 10- 32 bit 11- illegal	Optional; tie low for man test
x_data_out<31:16>	O	Upper 16 bit data out	Optional
x_data_out<15:0>[1]	O	Lower 16 bit data out (OR Lower 16 bit test/external master data out)	Req'd for manf. test
x_addr_out<21:16>	O	Upper address out	Optional
x_addr_out<15:0>[1]	O	Lower address out (OR Upper 16 bit test/external master data out)	Req'd for manf. test
x_ncs_out<3:0>	O	0 = Chip select active	Req'd for manf. test
x_nwe_out<3:0>	O	0 = Write enable active	Req'd for manf. test
x_noe_out	O	0 = Output enable active	Req'd for manf. test
x_nub_out	O	0 = Upper Byte select active	Optional
x_tri_data_op	O	1 = Data_out bus (x_data_out) tri-state	Req'd if I/O's are used
x_tri_data_ip	O	1 = Data_in bus (x_data_in) tri-state	Req'd if I/O's are used
x_tri_addr_op	O	1 = Address_out bus (x_addr_out) tri-state	Req'd if I/O's are used
x_tri_ctrl_op	O	1 = Control signals (x_ncs/nwe/noe/nub_out) tri-state	Req'd if I/O's are used
LOGIC UP INTEGRATION INTERFACE (UIM port)			
uim_mdata_in<31:16>	I	Upper 16 bit data in	Optional
uim_mdata_in<15:0>[1]	I	Lower 16 bit data in (OR external Micro Controller bus master data in)	Optional
uim_madd_in<21:16>	I	Upper address in	Optional
uim_madd_in<15:0>[1]	I	Lower address in (OR external Micro Controller bus master data in)	Optional
uim_mints<3:0>	I	1 = equivalent x_ncs area UIM controlled	Optional

Table 69 - Port Descriptions (continued)

Port Name	Type	Function	Bond out option
uim_mdints<1:0>	I	01 = dack1 relates to UIM logic slave 10 = dack2 relates to UIM logic slave (dack must be programmed high for UIM)	Optional
uim_test_mode	I	1 = UIM test mode selected (only used when 'test =1')	Req'd for UIM logic test
uim_master	I	1 = Up integrated logic is a bus master. Note: This should be used in conjunction with x_req and x_gnt.	Optional
uim_mdata_out<31:16>	O	Upper 16 bit data out	Optional
uim_mdata_out<15:0>	O	Lower 16 bit data out (OR external Micro Controller bus master data out)	Optional
uim_add_out<21:16>	O	Upper address out	Optional
uim_add_out<15:0>	O	Lower address out (OR external Micro Controller bus master data out)	Optional
uim_nmcs<3:0>	O	0 = Chip select active	Optional
uim_nmwe<3:0>	O	0 = Write enable active	Optional
uim_nmoe	O	0 = Output enable active	Optional
uim_nmub	O	0 = Upper Byte select active	Optional
uim_mdata_tri	O	1 = Data_out bus (uim_mdata_out) tri-state	Optional
ARM7TDMI™ Boundary Scan			
sdoutbs	I	See ARM7TDMI data sheet for details	Optional
commrx, commtx, ir, nenouti, nm, rangeout0, rangeout1, tapsm, tbit, busdis, drivebs, dbgrqi, eclk, ecapclk, ecapclkbs, highz, icapclkbs, nhighz, ntdoen, pclkbs, rstclkbs, screg, sdinbs, shclkbs, shclk2bs, tck1, tck2	O	See ARM7TDMI data sheet for details	Optional
ARM7TDMI™ IDREG			
id_reg<31:0>	I	Unique code attached to the ARM® core	Contact Zarlink design centre for details

Table 69 - Port Descriptions (continued)

Port Name	Type	Function	Bond out option
MICROCONTROLLER EXPANSION INTERFACE			
areq_spare	I	Contact your local design centre for details of this interface.	Optional
b_super, b_hold, ncs, b_mode, int_clken, a_gnt_spare	O	Contact your local design centre for details of this interface.	Optional
b_addr, b_size, b_write, b_data, b_wait, b_error, b_req, b_seq, b_opc, b_lock.	I/O	Contact your local design centre for details of this interface.	Optional

Table 69 - Port Descriptions (continued)

NOTE[1]: These signals form part of the off-core bus master interface details of which can be found in 2.6, “Off-core Bus Masters” on page 21.

Note: It is recommended that designers place a Bus Hold cell (Type CLBHOLD for CLA200 series CMOS Gate Arrays, Type HOLDX1 for GSC200 CMOS Standard Cell series) on each of the indicated Firefly MF1 ports (which are available at the Firefly MF1 Microcontroller Expansion Interface) shown below.
The purpose of the cells is to hold each microcontroller internal bus signal to a logical value during the time that the internal bus is held in the reset state (when no bus masters are allocated and the bus remains undriven). If the cells are omitted, then no logical effect will be seen and the microcontroller will continue to function correctly as normal; however, during periods of excessively long active resets (nSReset asserted), a small increase in power consumption may be observed.

Firefly MF1 Ports where hold cells are recommended include:

B_ADDR[31..0], B_OPC, B_REQ, B_SEQ, B_SIZE[1..0], B_SUPER, B_WRITE, B_WAIT, B_MODE.

Other Microcontroller Expansion Interface ports:

There are already hold cells on the signals B_DATA[31..0] and B_ERROR. B_HOLD is of VHDL Port Type ‘Buffer’ and as such does not facilitate the addition of a Bus Hold cell to itself. This is not a material omission. The signals are q_spare, ncs, intclken, a_gnt_spare do not require a Hold cell to be added.

9.7 Microcontroller Expansion Interface

The Firefly MF1 Core includes a selection of macrocells which are considered to be a good foundation for a generic Microcontroller. However, it is recognized that this mix of macrocells may not meet the needs of all Customers. Therefore, the Firefly MF1 Core includes an Expansion Interface which permits the customisation of the microcontroller core via the addition of new macrocells. For example, if an additional serial I/O port and a parallel port are required, these can be sourced from our library and the core can be enhanced accordingly. For full up to date details of the macrocells available and the use of the expansion interface, contact your Design Centre.

Note that the macrocells defined in this data sheet should be considered as the minimum set and it is only possible to add more functions. If there is a requirement to alter the ’minimum set,’ then contact your local Design Center for full details of the manufacture of custom microcontrollers.

9.8 AC Performance

All outputs which are expected to go either straight to I/O’s or single inputs are loaded with a ‘typical’ 180 fF load - this equates to a gate with an input loading of 2 Load Units (1xLU = 15 fF) and a nominal track load of 150 fF. Outputs used for UIM memory interface are loaded with a typical 1000 fF load. A representative loading has been applied to the Expansion Interface. All these loads were taken from an example ‘real’ design.

Note: MIN and MAX delays depend upon temperature range, junction temperature, supply voltage, input edge speed, and process spreads. Where appropriate, to make system design easier, MIN and MAX values have been calculated; otherwise, Typical values are shown. All temperatures shown are *junction* temperatures, unless otherwise indicated. The addition of significant active customer logic may cause an increase in junction temperatures in the resultant ASIC, and therefore, could require deratings to be applied as discussed below.

Accurate delays are calculated by Zarlink design kits on approved simulators.

All timings in this document are based on simulation results, running a series of tests in a standard test harness. For details of process, voltage and temperature derating, see *Section 10 "Delay Derating"* of the **CLA200 Design Manual Volume One, Publication number: DM4805**.

9.9 MPC On-Chip Wait-State Control

9.9.1 Timing Diagram

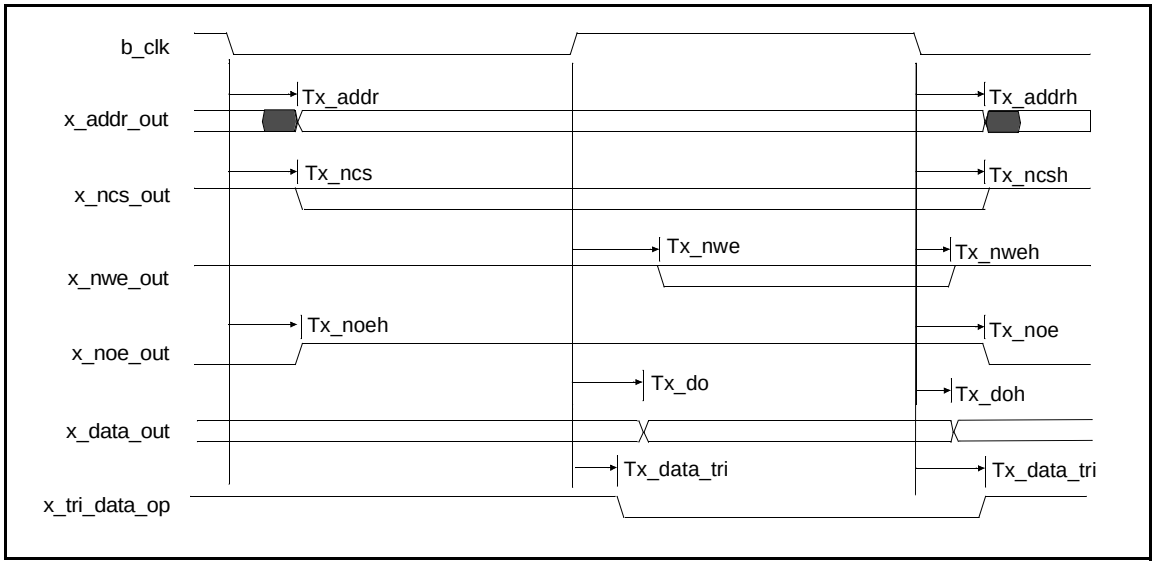


Figure 45 - MPC Memory Write Cycle

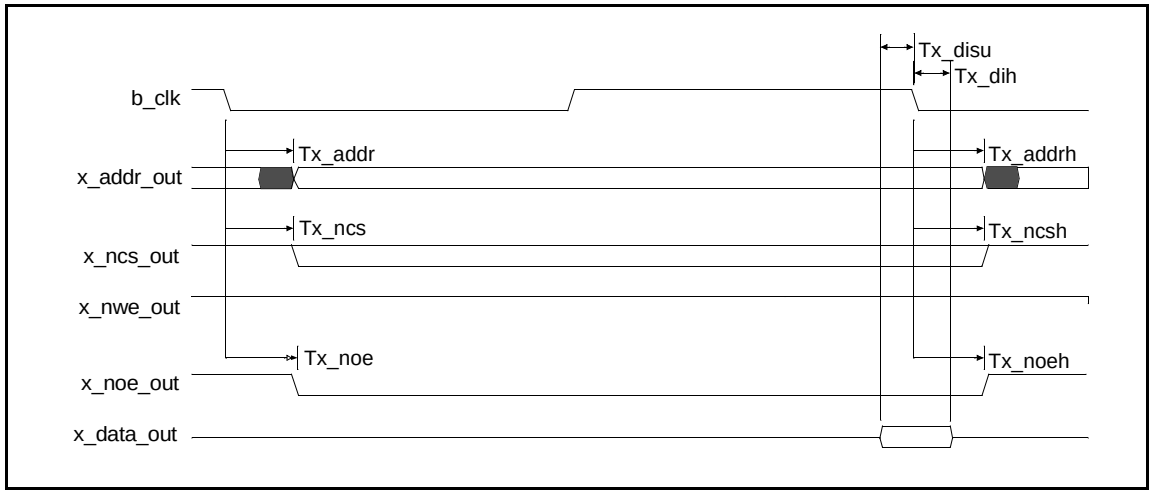


Figure 46 - MPC Memory Read Cycle

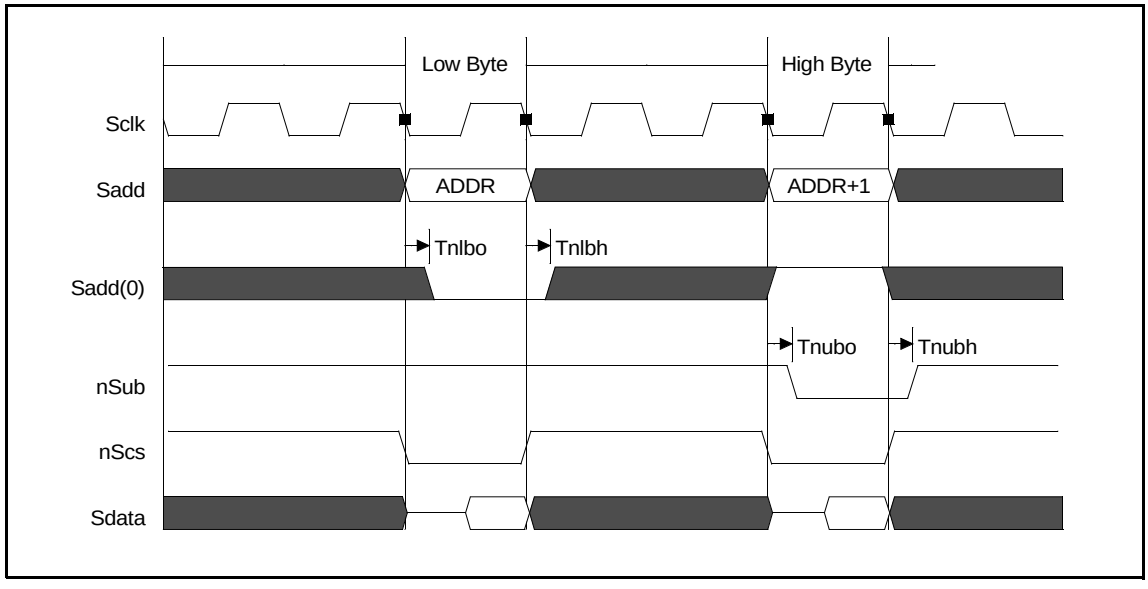


Figure 47 - 16-bit Byte Selectable Memory Timing

9.10 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 180 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Tb_clkl	14.0	-	ns	b_clk low period
Tb_clkh	14.0	-	ns	b_clk high period
Tx_addr	3.2	9.4	ns	b_clk to Address Valid
Tx_addrh	2.4	6.5	ns	Address hold after b_clk
Tx_ncs	3.0	8.5	ns	b_clk to chip select valid
Tx_ncsh	1.8	5.2	ns	Chip select hold after b_clk
Tx_noe	4.4	11.7	ns	b_clk to output enable active
Tx_noeh	1.9	5.2	ns	Output enable hold after b_clk
Tx_nwe	1.0	2.7	ns	b_clk to Write enable
Tx_nweh	1.2	3.3	ns	Write enable hold after b_clk
Tx_disu	2.5	-	ns	Data setup before b_clk
Tx_dih	0	-	ns	Data input hold time
Tx_do	3.9	9.7	ns	Data valid time after b_clk
Tx_data_tri	2.0	6.2	ns	Data tri-state control time after b_clk
Tx_doh	Tb_clkl	Tb_clkl	ns	Data out hold time after b_clk
Tnlbo	-	7.3	ns	b_clk to NLB valid
Tnlbh	3.0	-	ns	NLB hold time
Tnubo	-	9.5	ns	b_clk to NUB valid
Tnubh	2.4	-	ns	NUB hold time

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note that Minimum and Maximum values do NOT happen under the same conditions and hence cannot always be directly compared.

9.11 MPC External Wait-State Control

9.11.1 Timing Diagram

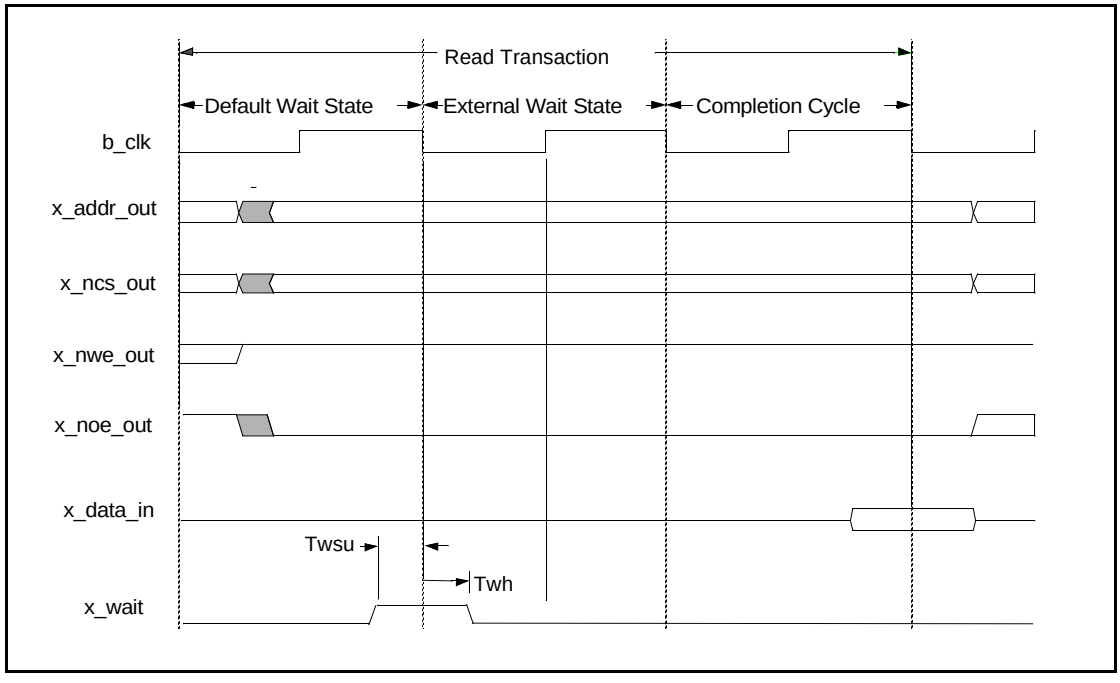


Figure 48 - MPC Memory Read Cycle With Off-chip Wait-state Control

9.11.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 180 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Twsu	2.0	-	ns	Swait setup time before Sclk
Twh	2.0	-	ns	Swait hold time after Sclk

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note that Minimum and Maximum values do NOT happen under the same conditions and hence cannot always be directly compared.

9.12 Logic Up-Integration Module

9.12.1 Timing Diagrams

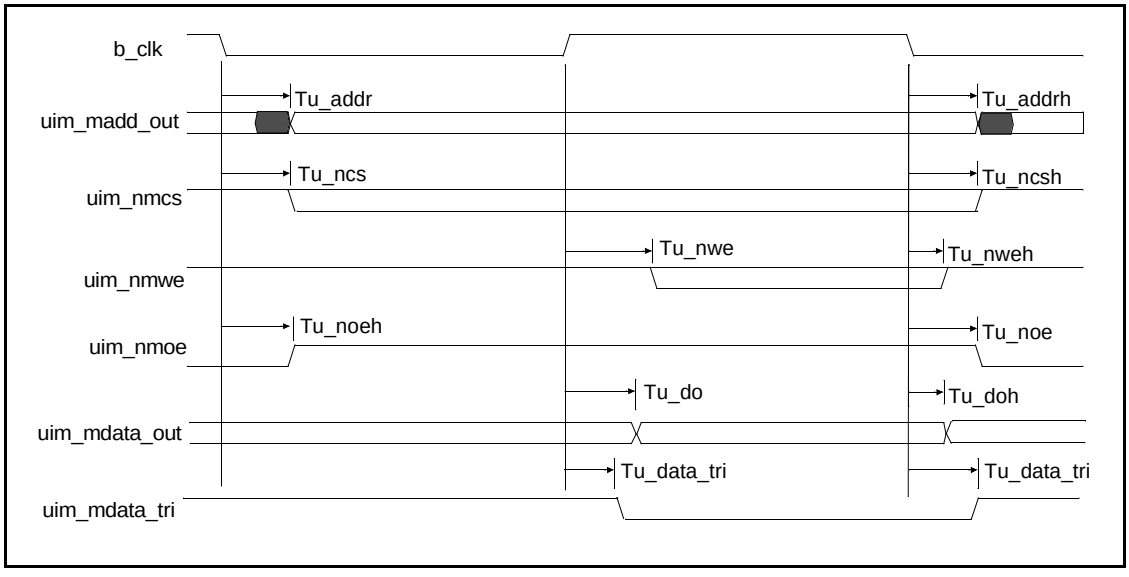


Figure 49 - UIM Memory Write Cycle

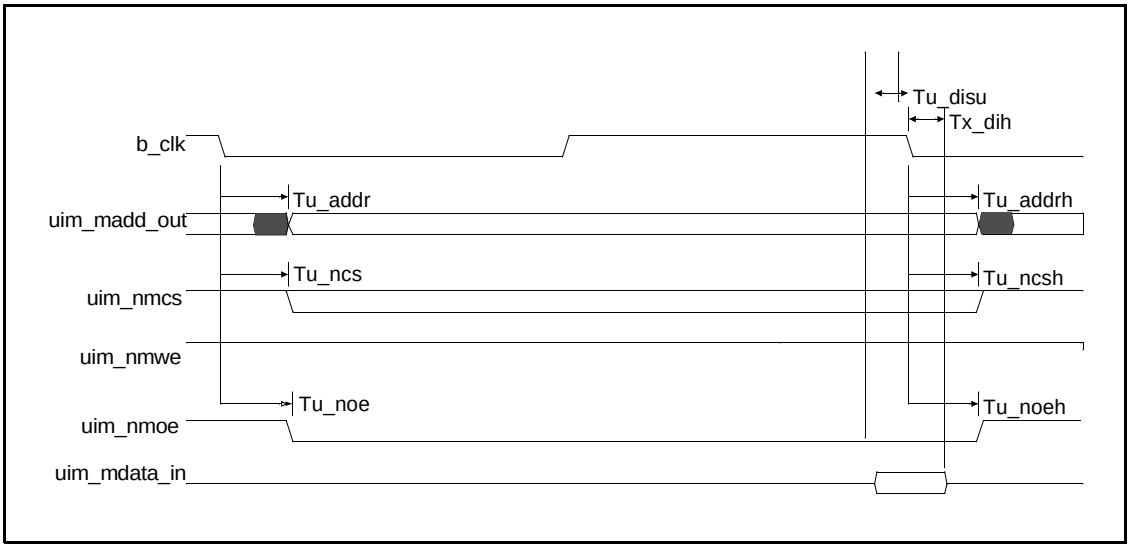


Figure 50 - UIM Memory Read Cycle

9.12.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 1000 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 1000 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Tu_addr	3.6	10.5	ns	b_clk to Address Valid
Tu_addrh	2.5	6.7	ns	Address hold after b_clk
Tu_ncs	4.3	13.0	ns	b_clk to chip select valid
Tu_ncsh	2.8	8.4	ns	Chip select hold after b_clk
Tu_noe	4.8	13.8	ns	b_clk to output enable active
Tu_noeh	3.4	10.4	ns	Output enable hold after b_clk
Tu_nwe	1.7	5.0	ns	b_clk to Write enable
Tu_nweh	1.8	5.4	ns	Write enable hold after b_clk
Tu_disu	4.5	-	ns	Data setup before b_clk
Tu_dih	0	-	ns	Data input hold time
Tu_do	4.1	9.7	ns	Data valid time after b_clk
Tu_data_tri	2.0	6.5	ns	Data tri-state control time after b_clk
Tu_data_trih	2.0	7.0	ns	Data tri-state control hold time after b_clk
Tu_doh	Tb_clkl	Tb_clkl	ns	Data out hold time after b_clk

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note that Minimum and Maximum values do NOT happen under the same conditions and hence cannot be directly compared.

9.13 DMA Single Address Transfer

9.13.1 Timing Diagram

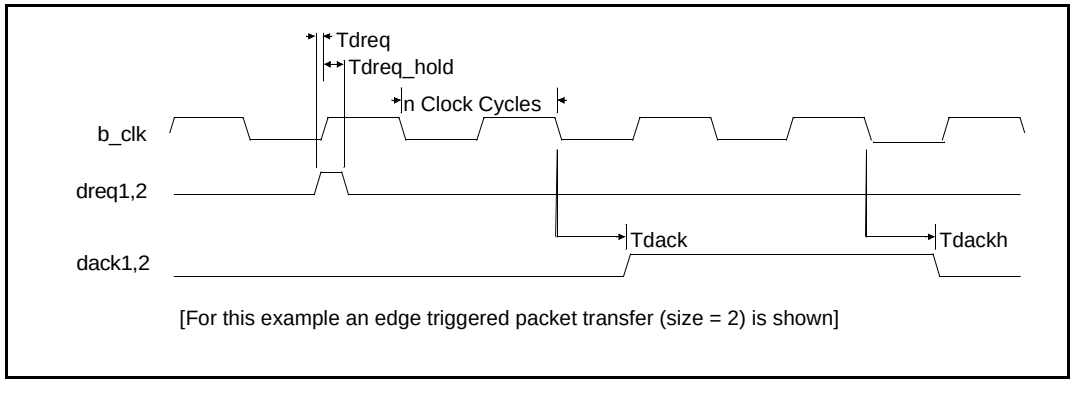


Figure 51 - DMA Request And Acknowledge Signals

9.13.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 180 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Tdreq	2.0	-	ns	dreq setup before b_clk (dreq1 & 2)
Tdrq_hold	2.0	-	ns	dreq hold time after b_clk (dreq1 & 2)
Tdack	2.0	7.8	ns	b_clk to dack active (dack1 & 2)
Tdackh	2.1	7.3	ns	dack hold after b_clk (dack1 & 2)

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note that Minimum and Maximum values do NOT happen under the same conditions and hence cannot be directly compared.

9.14 External Interrupt Input: Timing for Edge Sensitive Mode

9.14.1 Timing Diagram

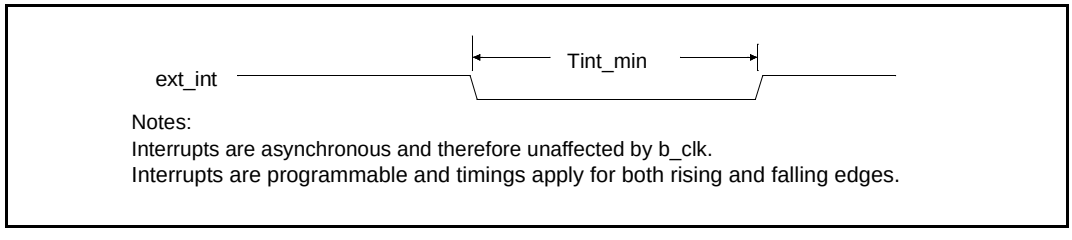


Figure 52 - Ext_int (1 & 2) Signals - Edge Triggered Mode

9.14.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 180 fF, input rise/fall 500 ps**)**

Parameters	MIN	MAX	Units	Notes
Tint_min	2 clock cycles	-	ns	Minimum external interrupt width

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note that Minimum and Maximum values do NOT happen under the same conditions and hence cannot always be directly compared.

9.15 External Interrupt Input: Timing for Level Sensitive Mode

When Level-sensitive interrupts are programmed, they must remain active until a suitable response is generated. i.e. until they have been serviced.

9.16 UART Interface

9.16.1 Timing Diagrams

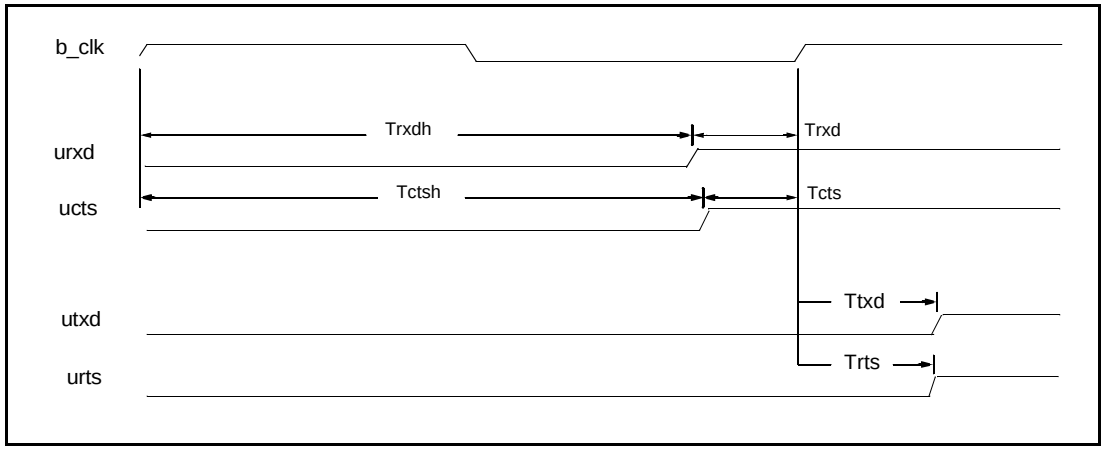


Figure 53 - UART Interface Signals

The UART input signals are essentially asynchronous with respect to b_clk; however, they are internally synchronised prior to presentation to the UART receive function. The UART 16x oversamples the input signal, therefore failure to recognise a change in signal level at one clock edge does not imply that the character will be received incorrectly. For the character to be corrupted, the input signal must be skewed by eight or more clock cycles for any particular bit

9.16.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 180 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Trxd	0	-	ns	Receive Data set-up to b_clk
Tcts	0	-	ns	Clear To Send set-up to b_clk
Ttxd	-	10.0	ns	Transmit Data delay from b_clk
Trts	-	10.0	ns	Ready To Send delay from b_clk

9.17 Broadcast Diagnostics

9.17.1 Timing Diagram

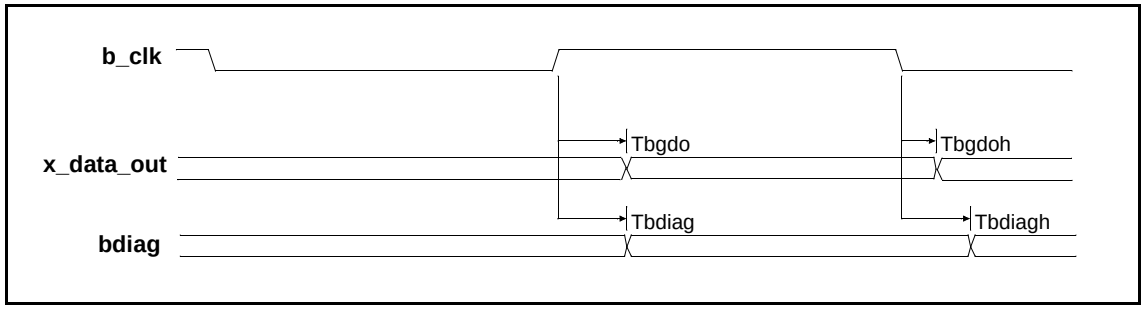


Figure 54 - Broadcast Diagnostic Signals

9.17.2 Timing Parameters

(Typical @25^{o}C, Typical process, 3.3 Volt, output load 180 fF)

Parameters	TYPICAL	Units	Notes
Tbgdo	3.1	ns	**x_data_out** valid after **b_clk** with broadcast diagnostics enabled
Tbgdoh	1.7	ns	**x_data_out** hold time after **b_clk** with broadcast diagnostics enabled
Tbdiag	8.2	ns	bdiag data valid after **b_clk**
Tbdiagh	7.3	ns	bdiag output data hold time after **b_clk**

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

9.18 External Master

9.18.1 Timing Diagram

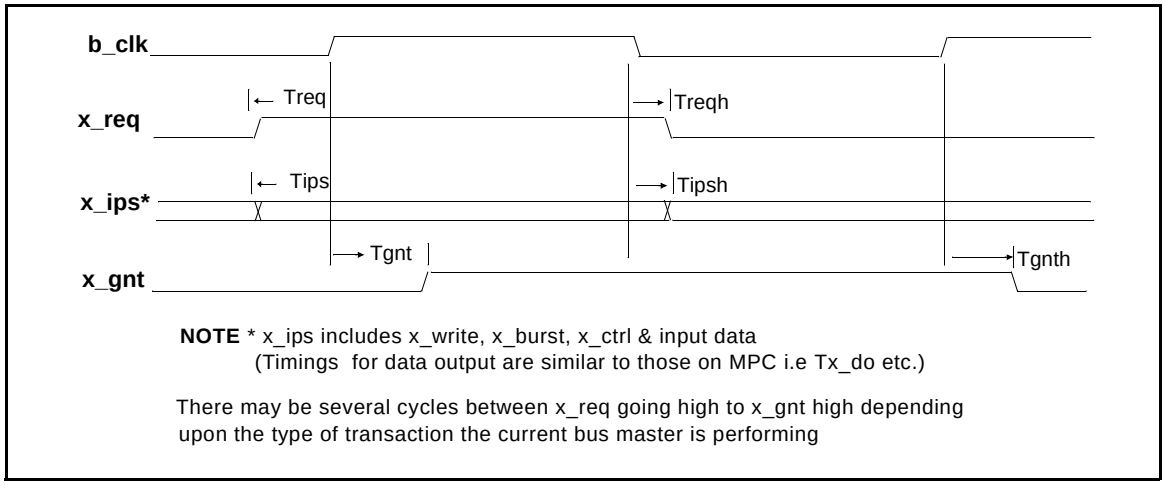


Figure 55 - External Master Control Signals

9.18.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

(MIN @-40^{o}C, Fast process, 3.6 Volt, output load 180 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Treq	2.0	-	ns	Request signal setup time to b_clk rise
Treqh	2.0	-	ns	Request signal hold after b_clk fall
Tips	2 0	-	ns	Control signal setup time to b_clk rise
Tipsh	2.0	-	ns	Control signal hold after b_clk fall
Tgnt	-	6.0	ns	b_clk rise to x_gnt valid
Tgnth	1.0	-	ns	x_gnt hold after b_clk

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note that Minimum and Maximum values do NOT happen under the same conditions and hence cannot be directly compared.

9.19 Timing Through UIM Port (for External Masters)

9.19.1 Timing Diagram

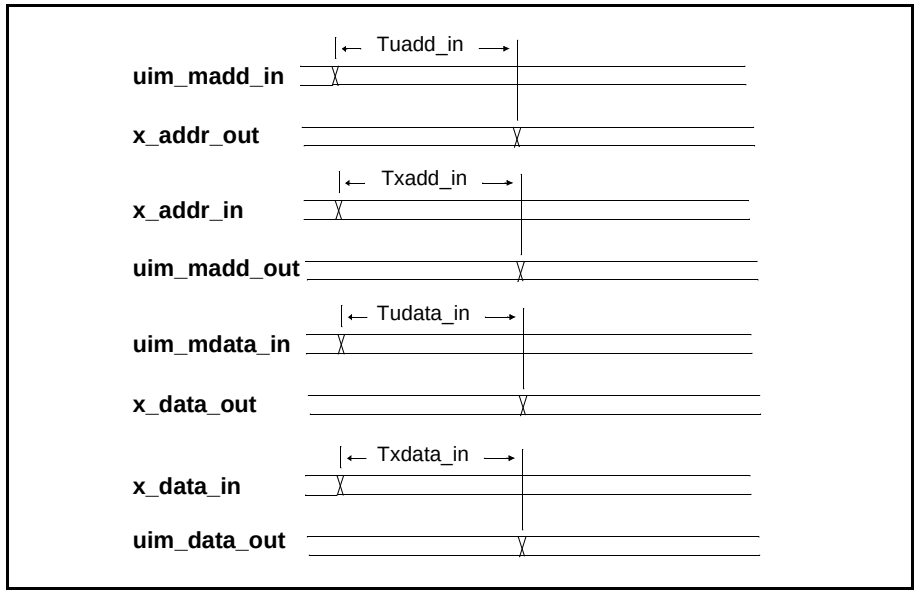


Figure 56 - Direct Through UIM Timings

9.19.2 Timing Parameters

(MAX @85^{o}C, Slow process, 3.0 Volt, output load 180 fF, input rise/fall 500 ps)

Parameters	MIN	MAX	Units	Notes
Tuadd_in	-	1.3	ns	**uim_madd_in** to **x_addr_out**
Txadd_in	-	2.1	ns	**x_addr_in** to **uim_madd_out**
Tudata_in	-	1.6	ns	**uim_mdata_in** to **x_data_out**
Txdata_in	-	2.6	ns	**x_data_in** to **uim_data_out**

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

9.20 JTAG Interface

9.20.1 Timing Diagram

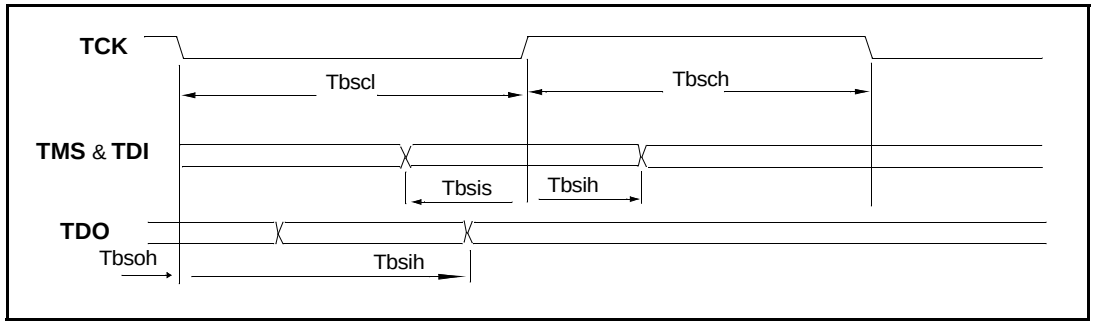


Figure 57 - JTAG Timings

9.20.2 Timing Parameters

(Typical @25^{o}C, Typical process, 3.3 Volt, output load 180 fF)

Parameters	3.3 Volt	Units	Notes
Tbscl	7.7	ns	**TCK** low period
Tbsch	7.7	ns	**TCK** high period
Tbsis	0	ns	**TDI**,**TMS** setup to [TCr]
Tbsih	1.7	ns	**TDI**,**TMS** hold from [TCr]
Tbsoh	4.4	ns	**TDO** hold time
Tbsod	8.4	ns	TCr to **TDO** valid
Tbsr	12.6	ns	Reset period

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. Accurate delays are calculated by Zarlink design kits on approved simulators.

Note. For correct data latching, the I/O signals (from the core and the pads) must be setup and held with respect to the rising edge of **TCK** in the CAPTURE-DR state of the INTEST and EXTEST instructions.

10.0 Appendix B: Modular Bus Operation

10.1 Introduction

Although the designer need not know details of the internal operation of the bus inside the embedded microcontroller core for most applications, some implementation details are included here for information purposes.

This chapter contains an overview of the protocols associated with bus arbitration and bus transactions. This represents sufficient information to give a working knowledge of the implementation of the internal bus within the microcontroller core. The bus architecture is optimized for efficient on-chip embedded systems. It is primarily designed to support ARM® CPUs and support modules, but is extensible to other processors and logic.

The ARM7TDMI™ CPU is a small and powerful 32-bit RISC CPU module suitable for integration in larger ASIC designs. However, when designing such embedded systems, the following problems must be addressed:

- Ad-hoc system design from scratch for each deeply embedded design significantly extends time to market and/or project development resource requirements.
- Manufacturing test requires fast efficient access to all component modules and I/Os, together with a rapid route for test pattern generation.
- The development of microprocessor based systems requires the use of Debug techniques such as logic analysis and ARM® EmbeddedICE™. These techniques can require special chip bond-outs to gain access to the embedded CPU core and to enhance system visibility.

Through the use of the bus, a modular approach is taken to solving these problems. Libraries of macro-functions (modules), complete with associated test patterns and software drivers, may be constructed and subsequently reused within different designs.

The following text describes the key aspects of the bus including the principal functional elements and protocol definitions.

10.1.1 Bus masters

The bus master is the controller of the current bus transaction. A bus master initiates bus requests (which are automatically managed by the microcontroller's internal Bus Arbiter), generates addresses and controls data transfers while it has bus access, by reading or writing data over the data bus.

Bus masters on a microcontroller include:

- The ARM7TDMI™ processor
- Direct Memory Access (multi-channel) controller(s)
- The System Services Module (for external test and debug)

10.1.2 Bus Slaves

A bus slave responds to addresses present on the internal bus that are in its allocated range within the address map. It will supply or receive data during read or write cycles on demand. A slave may set a wait signal to delay access, using the synchronous bus transfer protocol.

Example slave devices are:

- UARTs,
- The MPC, and
- Timers.

10.1.3 System Arbitration And Multiple Bus Master Support

The bus specification supports multiple bus masters. Bus masters must request the bus using a centralized arbiter. For example, DMA controllers may share the bus with one or more CPUs with the priorities resolved by the arbitration logic. The arbitration is pipelined by one clock cycle to avoid the need for redundant bus cycles during bus mastership changes.

10.1.3.1 Bus Masters

All bus masters are able to arbitrate for the bus using a request and grant protocol. When the bus is not granted, the master isolates itself from the bus and does not drive or respond to the bus control signals. A bus master will drive the full 32-bit address bus during bus ownership and drive the full 32-bit data bus during bus writes. The bus master also samples and responds to the bus control signals sent by slave devices to indicate their status.

10.1.3.2 Bus Slaves

Bus slaves, both internal memory and I/O devices, are selected by the decoding of the master's broadcast address. When selected during the address decode phase, the slave drives the bus status signals to indicate the condition of the current transaction. Invalid or illegal transaction types are indicated either directly by the slave or by an associated protection mechanism. The slave drives the full 32-bit data bus during read transactions. The data transaction phase may be extended if necessary for multiple bus cycles to allow for slaves with long transaction latencies.

11.0 Appendix C: The Firefly MF1 Developer Chip (MVT905001)

11.1 Introduction

In order to aid a designer in the development of application specific logic, a pre-bonded version of the Firefly MF1 core has been produced.

This device is called the Firefly MF1 Developer Chip or MVT905001. It has the same mix of macrofunctions as the Firefly MF1, but package pinout limitations mean that some signals are unavailable, and others are multiplexed.

This Appendix gives physical details of the MVT905001 device, with information on how to use the multiplexed diagnostic pins. The following table lists all the signals available on the 100 pin package.

Pin Name	Type	Function
Dreq1,2	I	DMA request
Dack1,2	O	DMA acknowledge
Sclk	I	System clock input
nSreset	I	System reset input, has 100K pull up resistor for use with external capacitor
Sdata<31:0>	IO	Data inputs/outputs
Sadd<17:0>	IO	Address Outputs
nScs<3:0>	O	Chip selects for external memory system
nSwe<3:0>	O	Write enables for external memory system
nSoe	O	Output enable for external memory system
Swait	I	Causes wait states within the MPC
Utxd	O	UART transmit data
Urxd	I	UART receive data
Iextint1,2	I	External interrupt inputs
nICE	I	Enable for JTAG pins, when low (has 100K pull up resistor)
[1]Bdiag<3:0>	O	Encoded bus state for logic analyser support
[1]Xburst	I	External master data burst
[1]Xwrite	I	External master write control
[1]Xreq	I	External master request for bus access
[1]Xcon	I	External master control input
[1]nTRST	I	Boundary scan reset.and Bdiag/Test mux control (has 100K pull up resistor)
[1]Tck	I	Test Clock
[1]Tdi	I	Test Data Input
[1]Tdo	O	Output from the boundary scan logic

Table 70 - MVT905001 Signal List

Pin Name	Type	Function
[1]Tms	I	Boundary scan test Mode Select.
VDD	-	Power
GND	-	Ground

Table 70 - MVT905001 Signal List (continued)

[1] Note that the Bdiag, X and JTAG scan test pins are shared, see below.

11.2 Test / Diagnostic Pins

There are three test/diagnostic requirements:

1. Direct access to the ARM7TDMI™ for manufacturing test of macrocells (4 needed)
2. Access to the scan pins of the ARM7TDMI for ICE support (5 needed)
3. Access to broadcast diagnostic pins for logic analyzer support (4 needed)

6 pins are allocated to test and diagnostics. They are multiplexed as below:

nICE=0	nICE=1	
Tdo	Bdiag<0>	Xreq
Tck	Bdiag<1>	Xwrite
Tdi	Bdiag<2>	Xburst
Tms	Bdiag<3>	Xcon
nTrst	1	0

Table 71 - Test Pin Functions

For normal operation, both control pins nICE and nTRST should be tied high via a resistor to allow them to be over-ridden. This means that the default mode is Broadcase Diagnostics. However, the outputs will only start toggling if Broadcast Diagnostics is internally enabled. The outputs are still driven, since leaving them floating may cause static current.

The Bdiag mode is default, rather than JTAG, because the nICE pin is also used to control DBGEN and DBGREQ on the Thumb® core. This means that the core always enters debug on reset, when nICE is low. This is not desirable for normal operation.

The use of bidirectional pins for ICE scan access means that there is a risk of contention if nICE is high when an external ICE controller is connected. To guard against this, resistors may be added between the ICE connector and the three pins which are vulnerable to this contention - Tck, Tdi and Tms. A value near 300 ohms will provide reasonable protection, while not excessively slowing input edges.

An additional safeguard is to switch nICE automatically when the ICEbreaker is connected. This can be accomplished by connecting nICE to a pin on the ICE connector which is grounded when the connector is plugged in.

11.2.1 To Use ICE

nICE <= 0

Connect the Embedded Ice Interface. Power up, nSreset <= 0 to initialize the system, and enable debug. On deasserting nSreset, the Ice interface will be active, and the core will be in debug mode.

11.2.2 To Use Broadcast Diagnostics

Disconnect Embedded Ice Interface.

nTRST <= 1

nICE <= 1

The Bdiag outputs will then need to be activated by setting the appropriate enable bit through software. The switching order is only important if moving from ICE mode to Bdiag mode without resetting the system, in which case nTRST should be set first, to ensure that test mode is not entered inadvertently.

11.2.3 To Use System Test Mode

nTRST <= 0

nICE <= 1

For information only - this is used for manufacturing test of the device.

11.3 Internal Tie-offs

The following ports of the Firefly MF1 Core (see Chapter 9.0 Appendix A: Firefly MF1 Core Datasheet) have been tied off in the MVT905001 device:

11.3.1 Timer 1

- **ten1a** and **ten1b** enables both tied high (i.e. enabled)

11.3.2 Timer 2

- **ten2a** and **ten2b** enables both tied high (i.e. enabled)

11.3.3 MPC

- **reset_size[1:0]** Both signals tied low (i.e. default to 8 bit)

11.3.4 UART

- **ueclk** signal tied low
- **ucts** signal tied high
- **udc**d signal tied high
- **udsr** signal tied low
- **ri** signal tied low.

11.3.5 DMA Controller

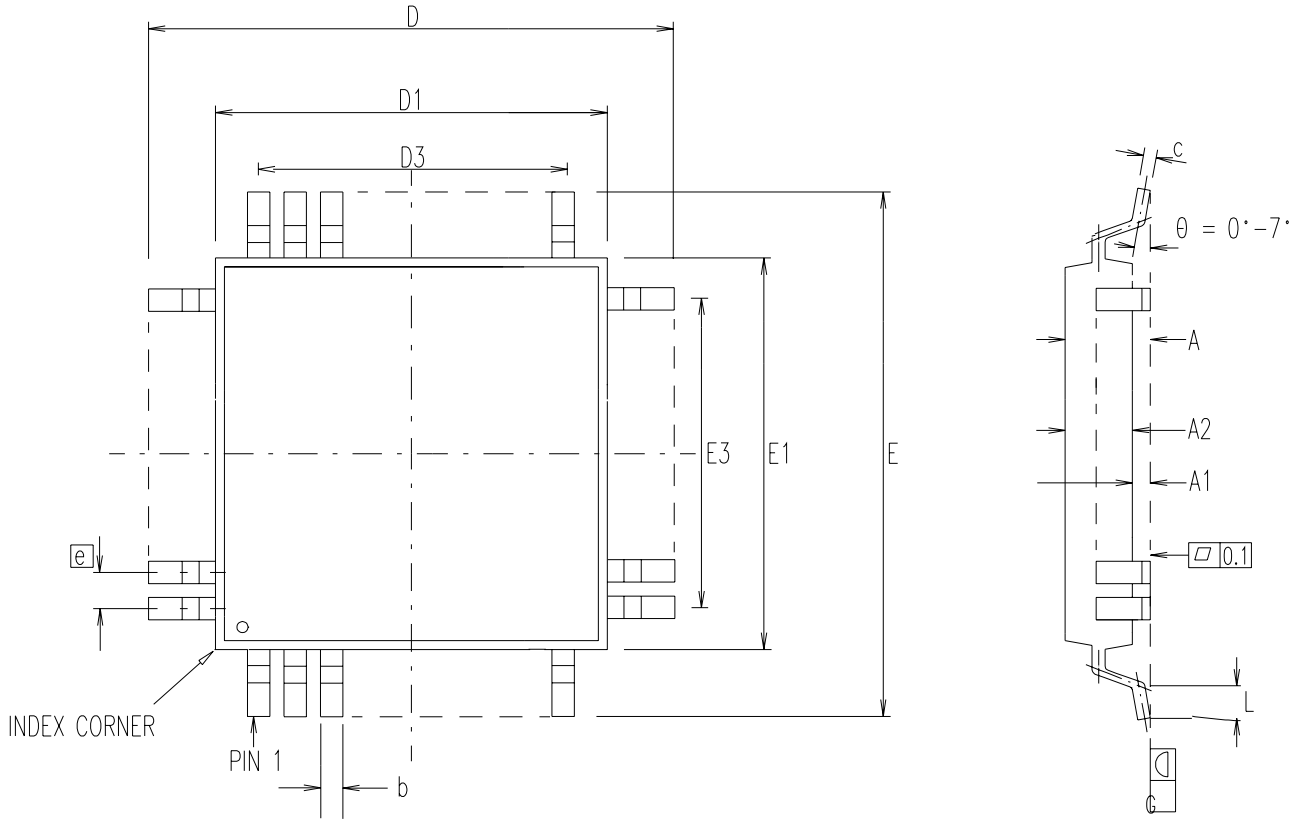
When the DMA controller is configured in **Butterfly compatible mode**, the channel 1 trigger sources **dreq1_src(0:3)** (see also section 2.7.1.3, “DMA Triggers”) are all GROUNDED.

11.4 MVT905001 Pin-out

1	Sadd<11>	31	Sadd<4>	51	Dack1	81	Sadd<17 >
	VDD		Sdata<3>		VDD		Sadd<16>
	GND		Sdata<19>		GND		Sdata<15>
	Sdata<10>		Sadd<3>		nScs<3>		Sdata<31>
5	Sdata<26>	35	Sdata<2>	55	nScs<2>	85	Sdata<30>
	Sadd<10>		GND		nScs<1>		GND
	Sdata<9>		VDD		nScs<0>		VDD
	Sdata<25>		Sdata<18>		nSwe<3>		Sadd<15>
	Sadd<9>		Sadd<2>		nSwe<2>		Sdata<14>
10	Sdata<8>	40	Sdata<1>	60	nSwe<1>	90	Sadd<14>
	GND		Sdata<17>		nSwe<0>		Sdata<13>
	VDD		Sadd<1>		GND		Sdata<29>
	Sdata<24>		Sdata<0>		VDD		Sadd<13>
	Sadd<8>		VDD		Sclk		VDD
15	Sdata<7>	45	GND	65	nSreset	95	GND
	Sdata<23>		Sdata<16>		nICE		Sdata<12>
	Sadd<7>		Sadd<0>		Urxd		Sdata<28>
	Swait		Dreq2		Utxd		Sadd<12>
	VDD		Dack2		VDD		Sdata<11>
20	GND	50	Dreq1	70	GND	100	Sdata<27>
	Sdata<22>				Tdo		
	Sdata<21>				Tck		
	Sdata<6>				Tdi		
	Sadd<6>				Tms		
25	Sdata<5>			75	nTrst		
	Sadd<5>				Iextint2		
	Sdata<4>				Iextint1		
	GND				GND		
	VDD				VDD		
30	Sdata<20>			80	nSoe		

Table 72 - MVT905001 (100 MQFP) Pin-out

11.5 Package Outline Drawing



Note 1: Pin 1 indicator may be a corner chamfer, dot or both.

Note 2: Controlling dimensions are in millimeters.

Note 3: The top package body size may be smaller than the bottom package body size by a max. of 0.15 mm.

Note 4: Dimension D1 and E1 do not include mould protusion. Allowable protusion is 0.25 mm per side. D1 and E1 are maximum plastic body size dimensions including mould mismatch.

Note 5: Dimension b does not include dambar protusion. Allowable dambar protusion shall not cause the lead width to exceed the maximum b dimension by more than 0.08 mm.

Coplanarity, measured at seating plane G, to be 0.10 mm max.

Figure 58 - Package Outline Drawing (100 MQFP)

Symbol	Dimensions in millimetres				Dimensions in inches		
	MIN	Nom	MAX		MIN	Nom	MAX
A	2.75		3.40		0.108		0.134
A1	0.25		0.50		0.010		0.020
A2	2.50		2.90		0.098		0.114
D	22.95		23.45		0.904		0.923
D1	19.80		20.20		0.780		0.795
D3	18.85 Ref				0.742 Ref		
E	16.95		17.45		0.667		0.687
E1	13.80		14.20		0.543		0.559
E3	12.25 Ref				0.486 Ref		
L	0.73		1.03		0.029		0.041
e	0.62 BCS				0.026 BSC		
b	0.22		0.38		0.009		0.915
	Zarlink drawing number: GPD00305						

Table 73 - Dimensions Of 100 Lead MQFP Package

11.6 MVT905001 Electrical Specification

The following data supplied should be referenced in conjunction with the **Firefly MF1 Core Design Manual**.

11.6.1 DC Parameters

Symbol	Parameter	Min	Max	Units
VDD	Supply Voltage	-0.5	7.0	V
Vip	Voltage applied to any pin	VSS-0.5	VDD+0.5	V
Osct	Output short circuit time		1	second
ESD	HBM model ESD		4	KV
Ts	Storage temperature	-55	150	deg C

Table 74 - Absolute Maximum Ratings

Note: These are stress ratings only and exceeding the absolute maximum ratings may permanently damage the device. Operating the device at the absolute maximum ratings for extended periods may also effect the device reliability.

Symbol	Parameter	5 Volt		Units
		Min	Max	
VDD	Supply Voltage	4.5	5.5	V
Ta	Ambient operating temperature	0	70	deg.C
I_{lu}	DC latch-up current	>500		mA

Table 75 - DC Operating Conditions

Note: MVT905001 will also operate with a supply voltage of 3.3V +/- 10% and over the Industrial Temperature range of -40 to +85° C. Electrical Characteristics For Device Input/Output.

All inputs, outputs and bidirectional signals (with the exception of Sclk) are CMOS/TTL compatible, with the characteristics shown below.

Parameter	Description	Min	typical	Max	Condition
V_{OH}	Output High Voltage	$0.8V_{DD}$	$0.9V_{DD}$		5V I_{OH}=6mA
V_{OL}	Output Low Voltage		0.2V	0.4V	5V I_{OL}=6mA
O_{SC5}	Output short circuit current 5V supply	30		160mA	V_{DD}=5.5V

Table 77 - Device Output Pin Characteristics

Parameter	Description	Min	Max	Units
V_{IH}	Input High Voltage	2.0		V
V_{IL}	Input Low Voltage		0.8	V
I_L	Input leakage current	-1	1	uA
C_{IN}	Input (and I/O) Capacitance		4	pF

Table 78 - Device Input Pin Characteristics

Note: The **nSreset, nICE & nTRST** input pins have an internal pull-up resistor of typically 100k Ohms tied to them.

Parameter	Description	Min	Max	Units
V_{IHK}	Sclk Input High Voltage	$0.7V_{DD}$		V
V_{ILK}	Sclk Input Low Voltage		$0.2V_{DD}$	V
I_{LK}	Sclk Input leakage current	-1	1	uA
C_{INK}	Sclk Input (and I/O) Capacitance		4	pF

Table 79 - Sclk Input Pin Characteristics

11.6.2 Loading Effects On Output Timing

Each timing parameter for external outputs within this document assume a 40pF load capacitance. This table can be used to calculate timings for alternative loads.

Parameter	5 Volt value	Units	Description and notes
T_{RFF}	42	pS/pF	Maximum output timing adjustment for Variance from 40pF loads (falling edges)
T_{RFR}	90	pS/pF	Maximum output timing adjustment for variance from 40pF loads (rising edges)

Table 80 - Variances To Output Timing

11.6.3 Power Consumption

Typical active power is heavily dependent upon the application environment. Loading of output pins, system clock speed and activity, core activity and on-chip peripheral usage over time will all significantly impact this figure. The values given below assume constant operation of the ARM® core plus typical activity expected using peripheral macro-functions, with **Sclk** @ 28.7MHz 5v nominal.

Symbol	Condition	5 volt nominal	Units
PD	Typical active power consumption	350	mW
PD	Static power consumption	15	uA

Table 81 - Device Power Consumption

11.6.4 AC Performance

The timing parameters on the following pages assume a logic switching point of 50% of VDD:

All inputs assume rise and fall times of nominally 2 ns. Minimum (min) and maximum (max) figures are referenced at extremes of Voltage and Temperature.

Important Notes:

All parameters will scale from their MIN to MAX value as temperature RISES or voltage FALLS. Their relationship, however, will always remain the same. Therefore if conditions give rise to a maximum propagation delay, such as **Taddr** (max), any corresponding hold times will also be maximum e.g. **Tnweh** (max).

All timings in this document are preliminary figures based on simulation results under worst/best case conditions where appropriate.

11.6.5 MPC Timing Diagrams: On-chip Wait-state Control

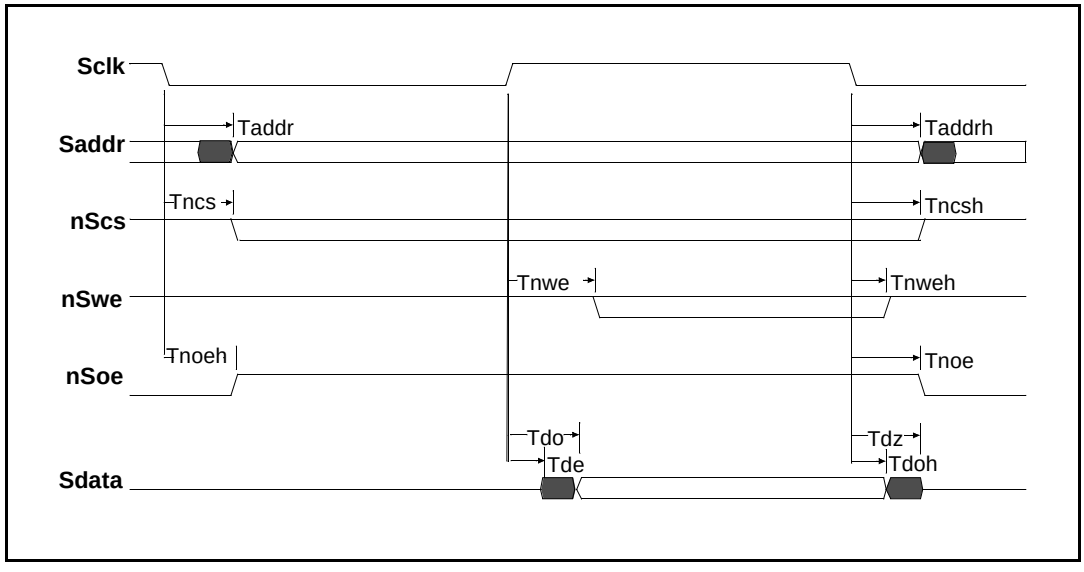


Figure 59 - MPC External Memory Write Cycle

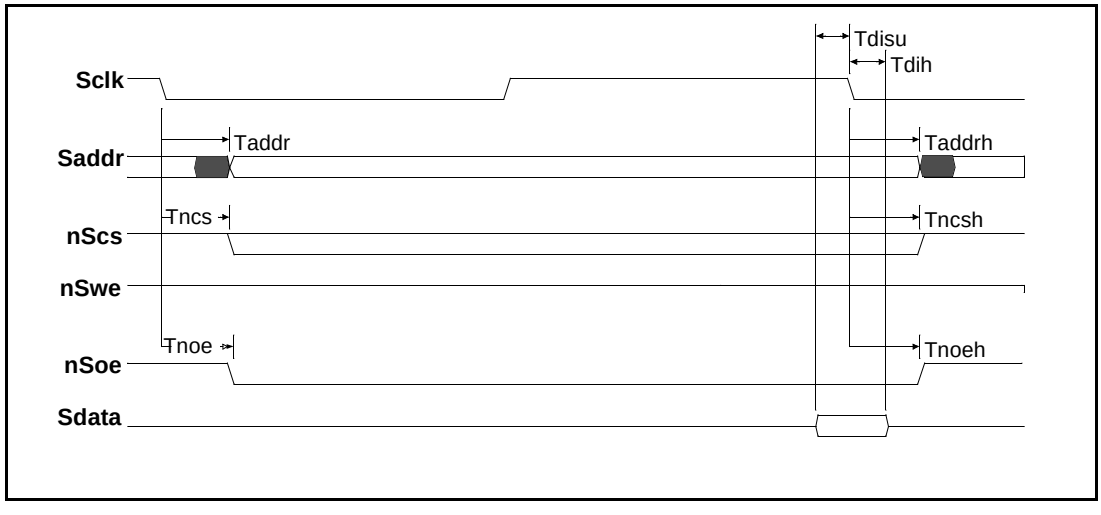


Figure 60 - MPC External Memory Read Cycle

Note: These transactions are the same whether the ARM® core OR the DMA Controller is the current bus master

Parameter	Min	Max	Unit	Description and notes
Tsclkl	15.6		ns	Sclk low period
Tsclkh	15.6		ns	Sclk high period
Taddr	5.1	17.0	ns	Sclk to Address Valid
Taddrh[1]	4.4	14.8	ns	Address hold after Sclk
Tncs	5.1	17.5	ns	Sclk to chip select valid
Tncsh[1]	4.5	15.2	ns	Chip select hold after Sclk
Tnoe	6.0	21.0	ns	Sclk to output enable active
Tnoeh	5.2	17.6	ns	Output enable hold after Sclk
Tnwe	3.4	11.3	ns	Sclk to Write enable
Tnweh[1]	3.3	11.0	ns	Write enable hold after Sclk
Tdisu	4.0	-	ns	Data setup before Sclk
Tdih	0	-	ns	Data input hold time
Tde	4.0	13.5	ns	Output enable time after Sclk
Tdo	5.5	18.6	ns	Data valid time after Sclk
Tdz	4.0	14.5	ns	Data disable time after Sclk
Tdoh	4.0	13.5	ns	Data out hold time after Sclk

Table 82 - MPC Timing: Main Signals

1. **Tnweh** (**nSwe** rising) will ALWAYS precede **Taddrh** and **Tncsh**.

11.6.6 MPC Timing Diagram Showing Off-Chip Wait-State Control

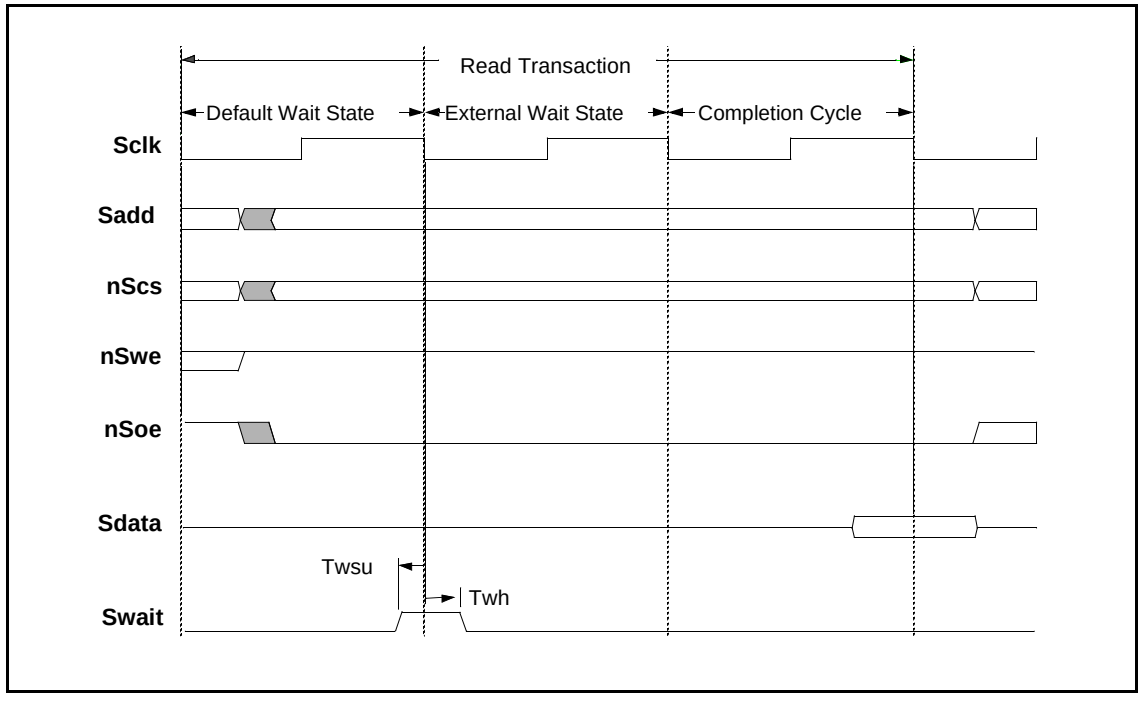


Figure 61 - MPC Timing: Swait

Parameter	Min	Max	Units	Description and notes
Twsu	5.0	-	ns	Swait setup time before **Sclk**
Twh	5.0	-	ns	Swait hold time after **Sclk**

Table 84 - MPC Timing: Swait

11.6.7 DMA Timing: Single Address Transfer

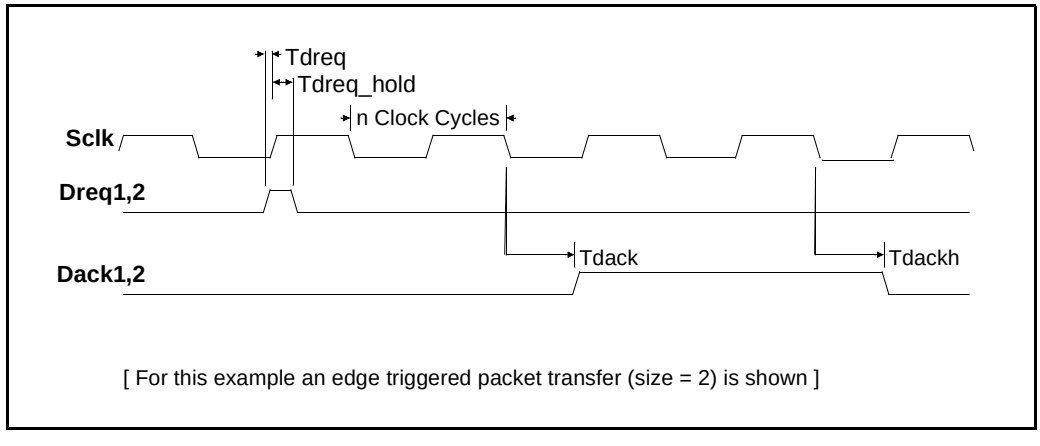


Figure 62 - DMA Timing: Single Address Transfer

NOTE: When performing a DMA transfer, memory signals are as per the MPC timing information.

Parameter	Min	Max	Unit	Description and notes
Tdreq	5.0	-	ns	**Dreq** setup before **Sclk** (Dreq1 & 2)
Tdreq_hold	5.0	-	ns	**Dreq** hold time after **Sclk** (Dreq1 & 2)
Tdack	5.0	16.6	ns	**Sclk** to **Dack** active (Dack1 & 2)
Tdackh	4.5	15.3	ns	**Dack** hold after **Sclk** (Dack1 & 2)

Table 85 - DMA Through MPC

11.6.8 External Interrupt Inputs: Timing For Edge-sensitive Mode

Notes:

Interrupts are asynchronous and therefore unaffected by Sclk.

Interrupts are programmable and timings apply for both rising and falling edges.

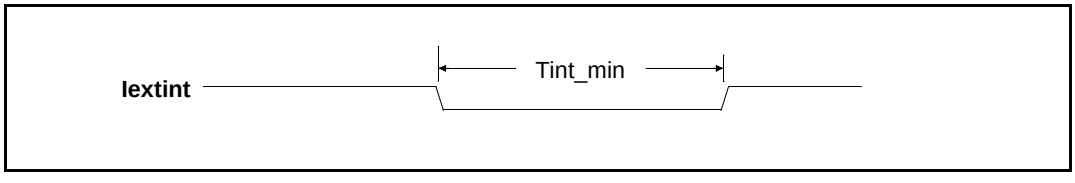


Figure 63 - External Interrupt Timing - Edge-Sensitive Mode

Parameter	Min	Max	Unit	Description and notes
Tint_min	2 clock cycles	-	ns	Minimum external interrupt width

Table 87 - Edge Triggered Interrupts: Extint (1 or 2)

11.6.9 External Interrupt Inputs: Timing For Level-sensitive Mode

When Level-sensitive interrupts are programmed, they must remain active until a suitable response is generated (i.e., until they have been serviced).

11.6.10 Broadcast Diagnostic Timing Diagrams

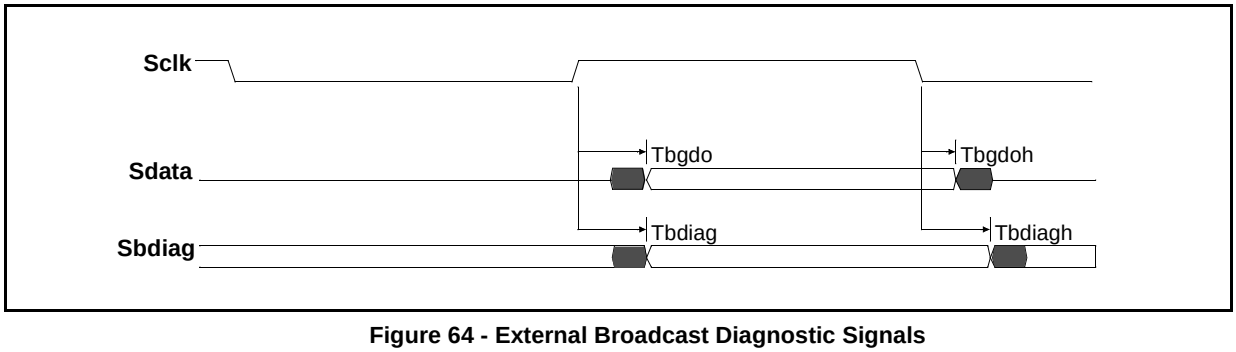


Figure 64 - External Broadcast Diagnostic Signals

Parameter	Min	Max	Units	Description and notes
Tbgdo	-	20.0	ns	**Sdata** valid after **Sclk** with broadcast diagnostics enabled
Tbgdoh	2.0	-	ns	**Sdata** hold time after **Sclk** with broadcast diagnostics enabled
Tbdiag	-	20.0	ns	**Bdiag** data valid after **Sclk**
Tbdiagh	5.0	-	ns	**Bdiag** output data hold time after **Sclk**

Table 89 - Broadcast Diagnostic Timing

11.6.11 JTAG Interface Timing Diagrams

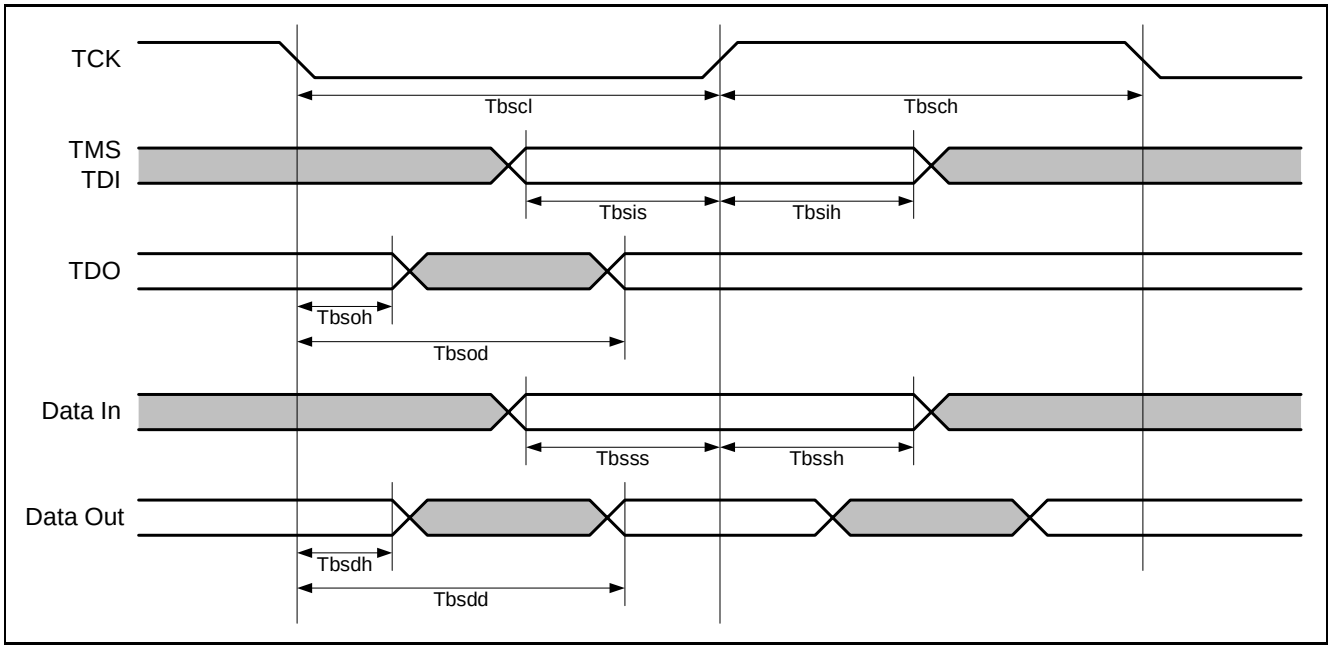


Figure 65 - JTAG Interface Timing

Note: These timings are taken from the ARM7TDMI™ datasheet section 8.14

Parameter	Min	Max	Unit	Description	Notes
Tbscl	15.6	-	ns	**TCK** low period	
Tbsch	15.6	-	ns	**TCK** high period	
Tbsis	5.0	-	ns	**TDI**,**TMS** setup to TCr	
Tbsih	5.0	-	ns	**TDI**,**TMS** hold from TCr	
Tbsoh	2.4	-	ns	**TDO** hold time	2
Tbsod		18	ns	**TCr** to **TDO** valid	2
Tbsss	3.6		ns	I/O signal setup to [TCr]	1
Tbssh	7.6		ns	I/O signal hold from [TCr]	1
Tbsdh	2.4		ns	data output hold time	2
Tbsdd		17.1	ns	**TCf** to data output valid	2
Tbsr	25	-	ns	Reset period	
Tbse		16.4	ns	Output Enable time	2
Tbsz		14.7	ns	Output Disable time	2

Table 90 - JTAG Interface Timings

Note 1: For correct data latching, the I/O signals (from the core and the pads) must be setup and held with respect to the rising edge of TCK in the CAPTURE-DR state of the INTEST and EXTEST instructions.

Note 2: Assumes that the data outputs are loaded with the AC test loads (see AC parameter specification).

Parameter	Min	Max	Unit	Description and notes
Tbscl	15.6	-	ns	**TCK** low period
Tbsch	15.6	-	ns	**TCK** high period
Tbsis	5.0	-	ns	**TDI**,**TMS** setup to TCr
Tbsih	5.0	-	ns	**TDI**,**TMS** hold from TCr
Tbsoh	2.4	-	ns	**TDO** hold time
Tbsod	-	25	ns	TCr to **TDO** valid
Tbsr	25	-	ns	Reset period

Table 91 - JTAG Interface Timings

Note 1: For correct data latching, the I/O signals (from the core and the pads) must be setup and held with respect to the rising edge of TCK in the CAPTURE-DR state of the INTEST and EXTEST instructions.

Note 2: Assumes that the data outputs are loaded with the AC test loads (see AC parameter specification).

11.7 Device I/O Summary

The following table gives a summary of all the pins on the Firefly MF1 Developer Chip together with a reference to the relevant Databook Chapter and the appropriate section (if applicable) within the AC performance section.

Mnemonic	Type	Function	Notes/AC performance reference	Handbook Chapter(s)
Dreq1,2	I	DMA Request	See DMA Timing & System Configuration register	2.7.1.3 6.0
Dack1,2	O	DMA Acknowledge	See DMA Timing	6.0
Sclk	I	System clock	See MPC Timing	3.0
nSreset	I	0 = System Reset	Includes 100k pullup	
Sdata31:0	I/O	Data I/O	See MPC Timing	3.0
Sadd17:0	O	Address Outputs for external memory		3.0
nScs3:0	O	0 = Chip select for external memory	See MPC Timing	3.0
nSwe3:0	O	0 = Write enable for external memory	See MPC Timing	3.0
nSoe	O	0 = Output enable for external memory	See MPC Timing	3.0
Swait	I	1 = Wait state requested	See Swait Timing	3.0
Utxd	O	UART Transmit Data		4.0
Urxd	I	UART Receive Data		4.0
nUrts	O	0 = UART ready to send.	NOTE: Not bonded out	4.0
nUcts	I	0 = UART is clear to send	NOTE: Not bonded out (100K pull down fitted)	4.0
Iextint1,2	I	External interrupts - Fully programmable.	See Ext. Interrupt Timing & Interrupt Source Channels	2.7.2 5.0
Bdiag3:0	O	Encoded Bus States. Note: Shared with JTAG	See Broadcast Diagnostic Timing	8.0
Tck	I	JTAG: Test clock (Shared with Bdiag1)	See ARM7TDMI Datasheet (Section 8.14)	8.0
Tdi	I	JTAG: Test data input (Shared with Bdiag2)	See ARM7TDMI Datasheet (Section 8.14)	8.0
Tms	I	JTAG: Boundary scan test Mode Select (Shared with Bdiag3)	See ARM7TDMI Datasheet (Section 8.14)	8.0

Table 92 - I/O Summary

Mnemonic	Type	Function	Notes/AC performance reference	Handbook Chapter(s)

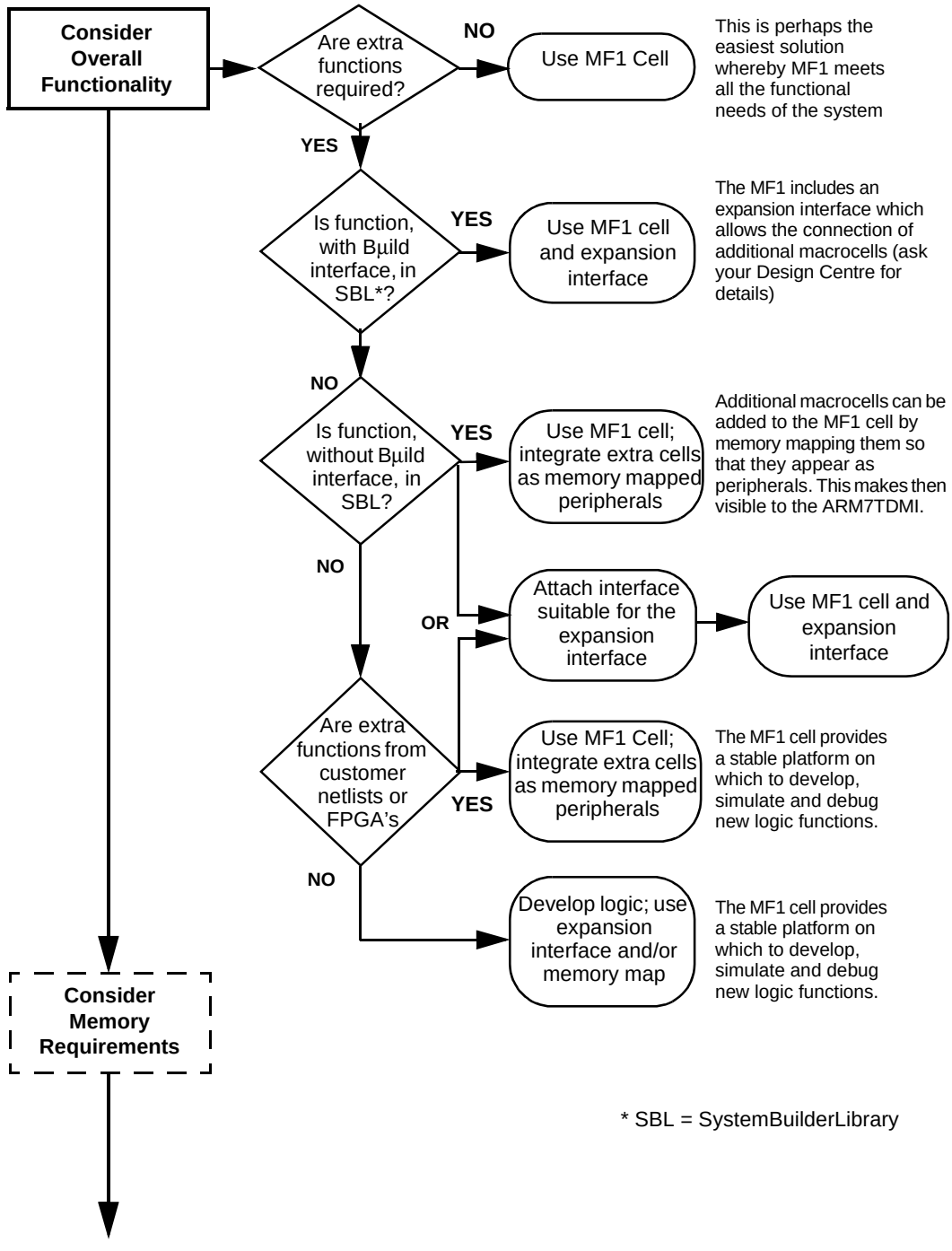
nTRST	I	Depends upon nICE polarity:
		If nICES = 0 then
		- JTAG: Boundary scan reset
		If nICES = 1 then
		1 = enables Bdiag outputs
		0 = enables System test modes
			See ARM7TDMI Datasheet (Section 8.14) and Test Diagnostic pin section	8.0

Tdo	O	JTAG: Boundary scan data output (Shared with Bdiag0)	See ARM7TDMI Datasheet (Section 8.14)	8.0
nICE	I	JTAG/Boundary scan mux control 0 = JTAG 1= Enable Bdiag or Test mode (see nTRST for details)	See Test Diagnostic pin section	8.0

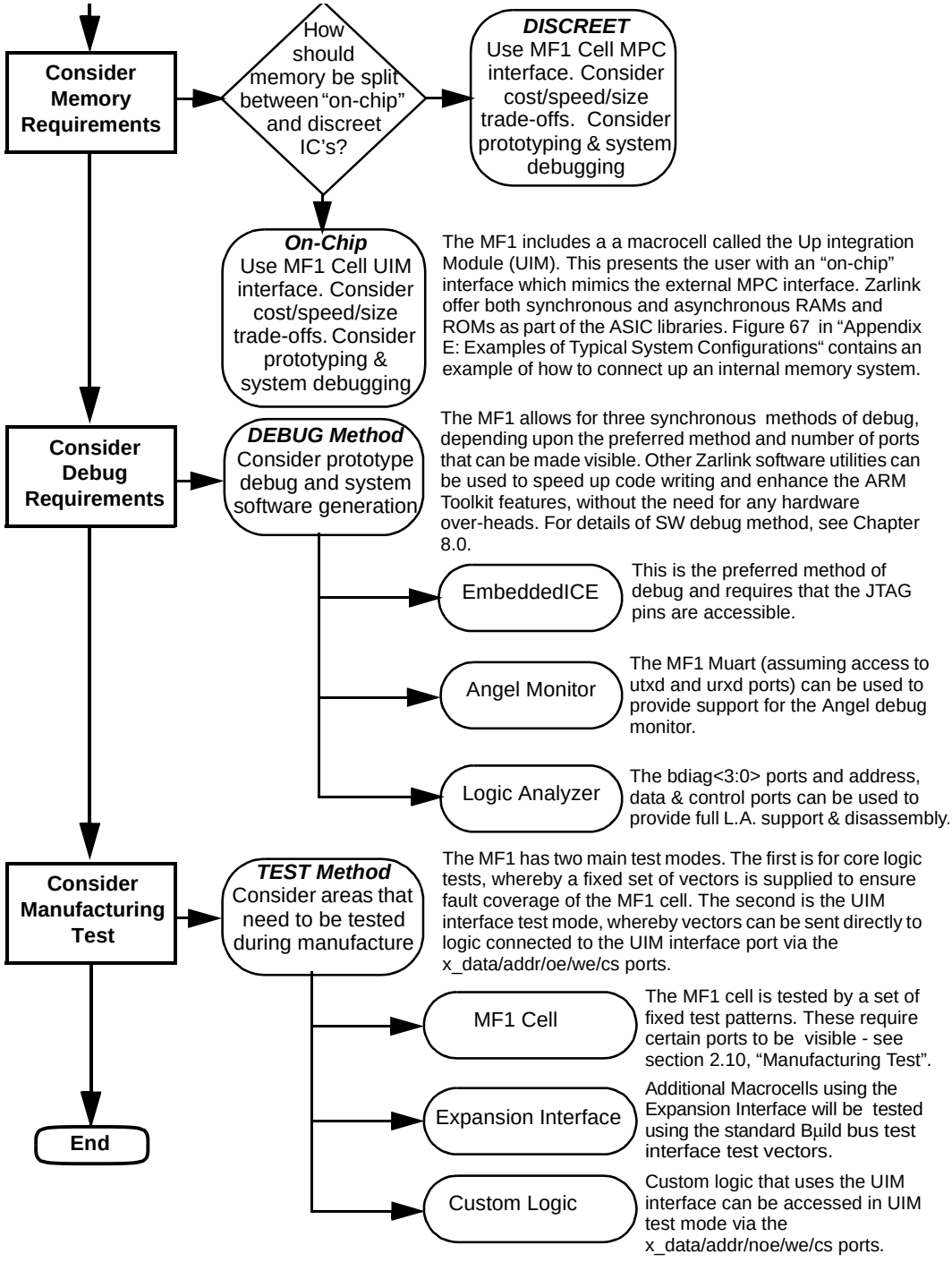
Table 92 - I/O Summary (continued)

12.0 Appendix D: System Decision Guide

The following chart is intended as a aid to System designers. It reflects the typical decisions taken while deciding upon System partitioning and indicates how the MF1 cell can provide a solution.



* SBL = SystemBuilderLibrary


Consider Memory Requirements
How should memory be split between "on-chip" and discreet IC's?
DISCREET
Use MF1 Cell MPC interface. Consider cost/speed/size trade-offs. Consider prototyping & system debugging
On-Chip
Use MF1 Cell UIM interface. Consider cost/speed/size trade-offs. Consider prototyping & system debugging
The MF1 includes a a macrocell called the Up integration Module (UIM). This presents the user with an "on-chip" interface which mimics the external MPC interface. Zarlink offer both synchronous and asynchronous RAMs and ROMs as part of the ASIC libraries. Figure 67 in "Appendix E: Examples of Typical System Configurations" contains an example of how to connect up an internal memory system.
Consider Debug Requirements
DEBUG Method
Consider prototype debug and system software generation
The MF1 allows for three synchronous methods of debug, depending upon the preferred method and number of ports that can be made visible. Other Zarlink software utilities can be used to speed up code writing and enhance the ARM Toolkit features, without the need for any hardware over-heads. For details of SW debug method, see Chapter 8.0.
EmbeddedICE
This is the preferred method of debug and requires that the JTAG pins are accessible.
Angel Monitor
The MF1 Muart (assuming access to utxd and urxd ports) can be used to provide support for the Angel debug monitor.
Logic Analyzer
The bdiag<3:0> ports and address, data & control ports can be used to provide full L.A. support & disassembly.
Consider Manufacturing Test
TEST Method
Consider areas that need to be tested during manufacture
The MF1 has two main test modes. The first is for core logic tests, whereby a fixed set of vectors is supplied to ensure fault coverage of the MF1 cell. The second is the UIM interface test mode, whereby vectors can be sent directly to logic connected to the UIM interface port via the x_data/addr/oe/we/cs ports.
MF1 Cell
The MF1 cell is tested by a set of fixed test patterns. These require certain ports to be visible - see section 2.10, "Manufacturing Test".
Expansion Interface
Additional Macrocells using the Expansion Interface will be tested using the standard Bµild bus test interface test vectors.
Custom Logic
Custom logic that uses the UIM interface can be accessed in UIM test mode via the x_data/addr/noe/we/cs ports.
End

13.0 Appendix E: Examples of Typical System Configurations

The following figures show an example of how a typical system should be wired up. All pins used in the example are detailed in the tables which follow and reference should be made to them when designing new systems.

13.1 Example External Memory System Configuration

An example memory system configuration is shown in Figure 66. This shows a system that contains the following:

- An 8 bit wide EPROM in memory area 1. This is where the boot code for the Micro Controller will reside. Note that the **reset_size[1:0]** bits have both been tied to ground, to indicate that area 1 will contain 8 bit wide memory.
- Area 2 contains 32 bit wide fast SRAM, made up of four 8 bit wide devices. This will allow writes to individual bytes.
- Area 3 contains a 16 bit peripheral.
- Area 4 is not used externally.
- A peripheral which is capable of performing DMA flyby transfers is also connected. It uses DMA channel 1 and hence connects to Dreq1 and Dack1. It also generates an interrupt which has been wired to spare external interrupt ext_int(3).

External I/O cells have been added to create bidirectional external buses. Note that even though they are not used by the memory configuration, some of the input paths on the control and address signals are required for manufacturing test.

Important points to note are:

Areas 1,2 and 3 are not going to be used internally hence there is no need to provide a uim test path for x_ncs_in<2:0>, so these are tied off.

As Area 4 is going to be used internally and testing will be via uim test mode, then x_ncs_in(4) needs to be bonded. Note that this path is only needed in uim test mode hence it could be multiplexed with another signal to save on pins.

All memory areas will have wait states generated automatically and hence x_wait is tied off.

No 16 bit memories capable of upper and lower byte selection have been used (either externally or internally), hence the x_nub_out pin is left unconnected and x_nub_in is tied off

Manufacturing test needs the lower 16 data and address lines to be bidirectional. As the data bus is 32 bits wide part of this has already been achieved. The address bus however requires to have the lower 16 bits connected via bidirectional I/O’s.

The Sdata bus requires tristatable inputs as well as outputs. These can be created by using standard I/O’s and placing tristatable buffers in the ’input’ data path.

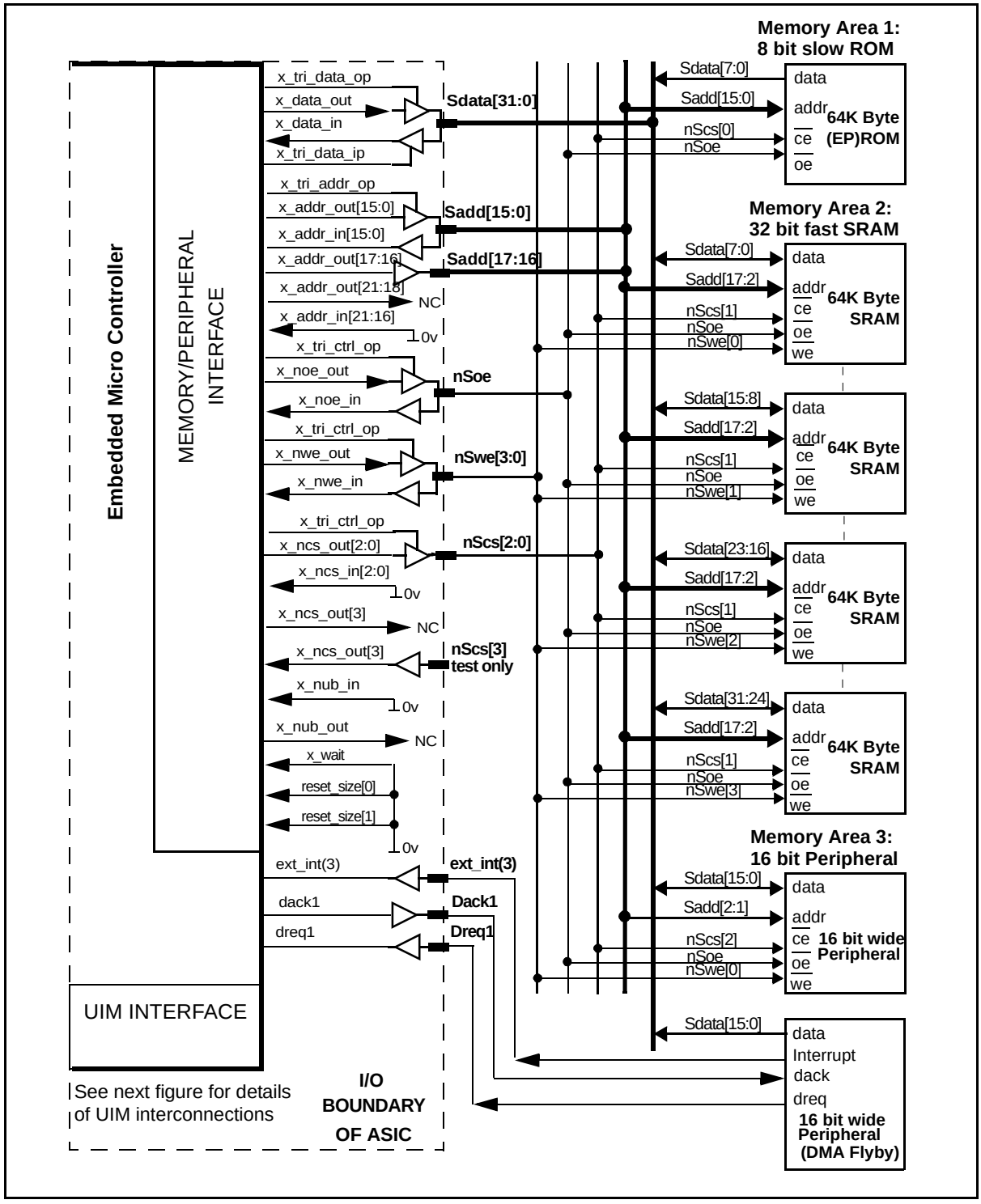


Figure 66 - External memory system interconnections

13.2 Example Of Connections To The UIM Interface

An example UIM interface system configuration is shown in Figure 67. As well as the external memory connections shown previously, the system also contains the following:

- Areas 1,2 and 3 are not used internally.
- Area 4 contains 32 bit wide fast internal SRAM, made up of four 8 bit wide devices. This will allow writes to individual bytes.
- A peripheral which is capable of performing DMA flyby transfers is connected. It uses DMA channel 2 and hence connects to Dreq2 and Dack2. It also generates an interrupt which has been wired to spare external interrupt ext_int(4).

Important points to note are:

- Areas 1,2 and 3 are not going to be used internally hence uim_mints[2:0] have been tied off low and uim_nmcs[2:0] have been left unconnected.
- As Area 4 is going to be used internally hence uim_mints[3] has been tied off high and uim_nmcs[3] has been connected to the internal RAM's
- Simple RAM interfaces may be needed if using our internal RAM's, suggested examples of these are in section 10 of this document.
- The internal memory does not use upper byte select so uim_nmub_out is not connected.
- An external Bus master has not been connected to the uim interface, hence uim_master is tied off low.
- The DMA flyby peripheral has been connected to DMA channel 2 and uses spare interrupt ext_int[4]. As the peripheral is internal uim_mdint[2] has been tied off high, the flyby peripheral on DMA channel 1 is external so uim_mdints[1] has been tied low.
- Spare interrupt lines, ext_int[5:23] have been tied off low to avoid floating inputs.
- As the data outputs from the RAM's and the flyby peripheral are tristatable, an internal bidirectional data bus has been created by including tristatable buffers in the uim_mdata_out path. In order to stop this bus floating when no internal devices are being selected, bus hold cells should be used. An alternative would be to keep the buses unidirectional and use a multiplexer, with a default condition, to route the data. However, the outputs of the RAM's would have to be constantly enabled, else hold cells would be needed on the input to the mux.

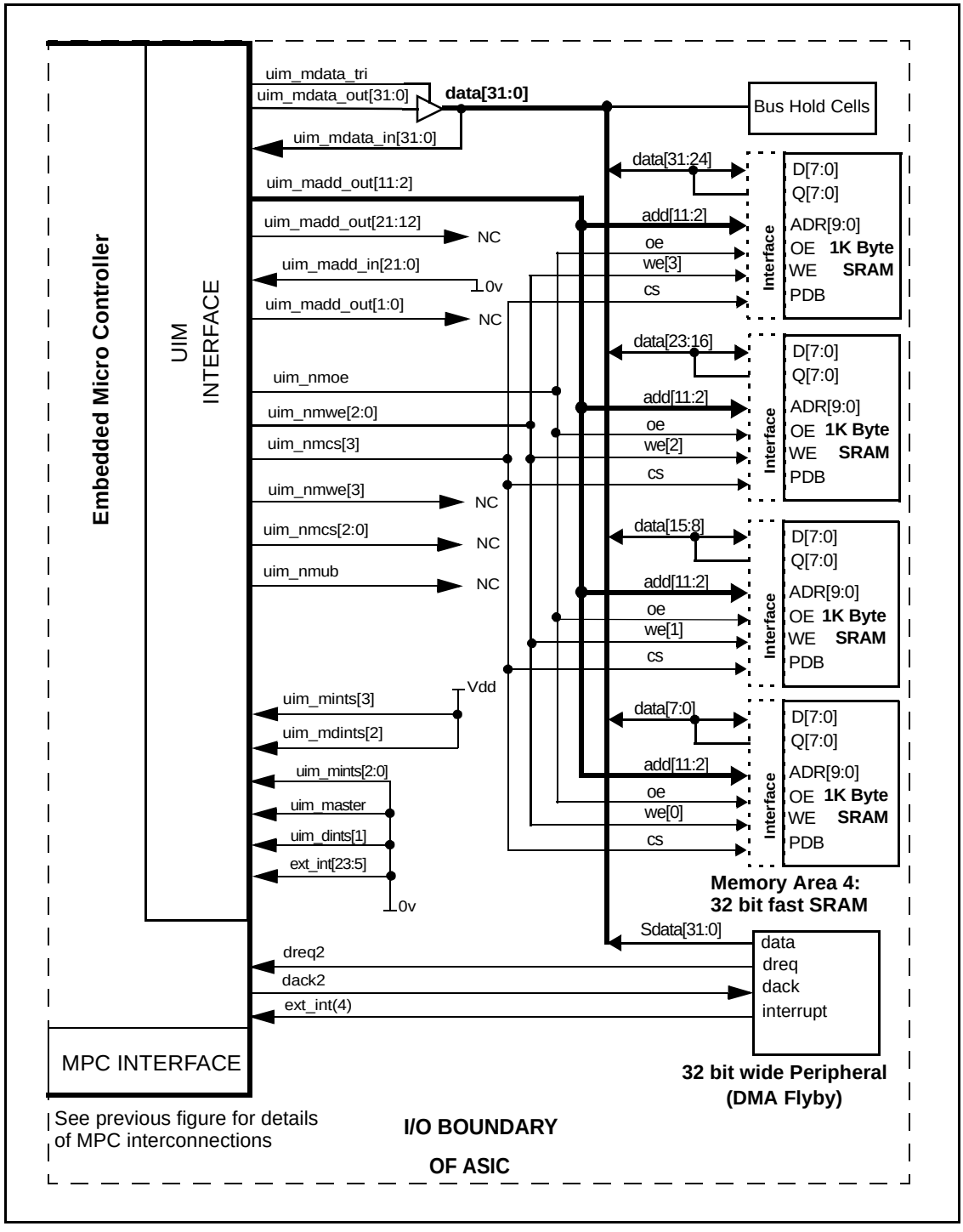


Figure 67 - Example UIM interface interconnections

13.2.1 Example of connecting an External System Master to the Firefly UIM port

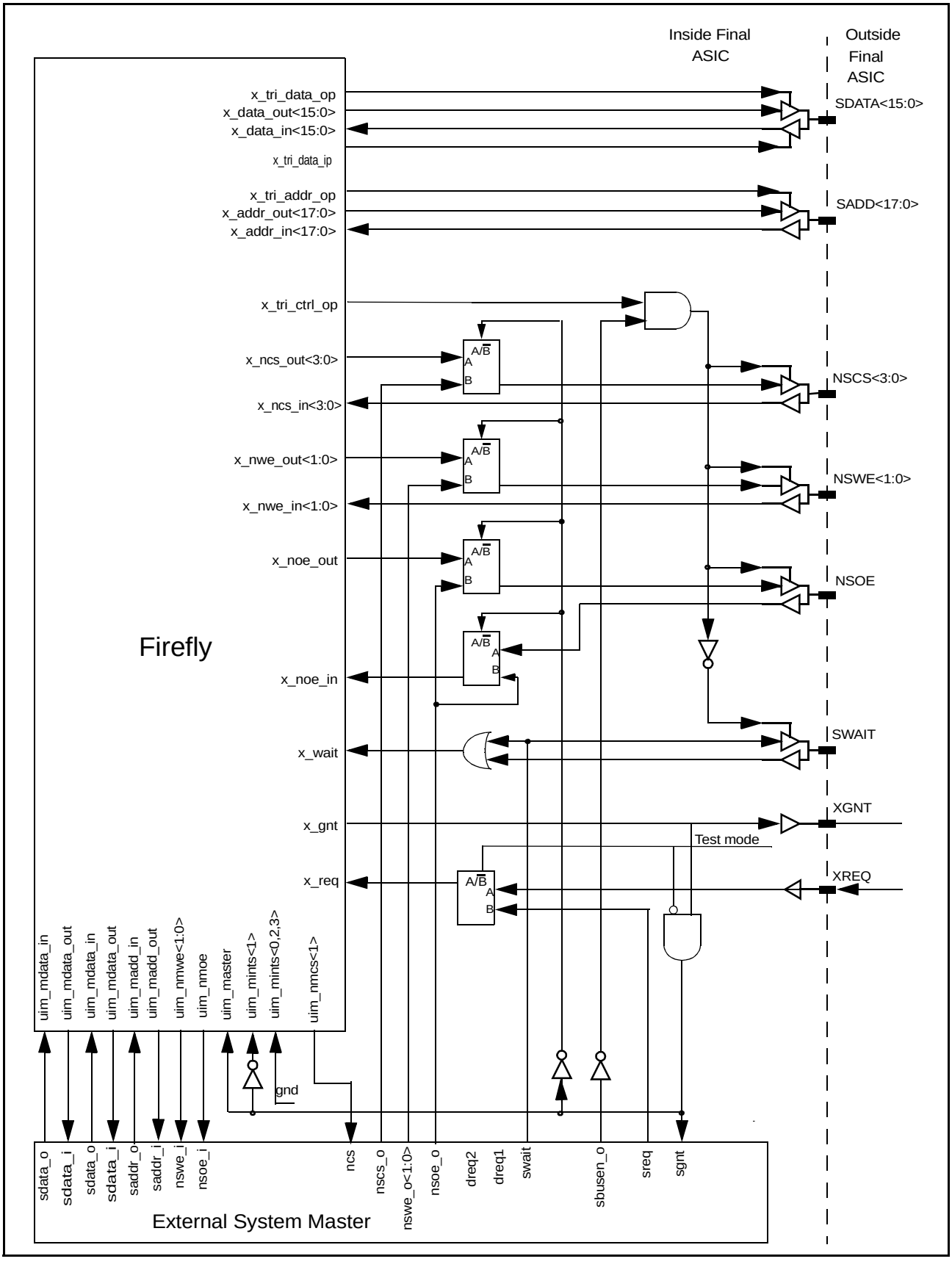
The UIM (Up Integration Module) in MF1 allows for an External 'System' Bus Master to be up integrated and connected internally to the UIM port.

The external master must indicate that it requires Firefly to stop acting as the 'System' bus master and relinquish control of the external memory bus. This is achieved by asserting X_req and waiting for X_gnt as comfirmation.

The UIM port needs to know whether the Bus Master has been connected internally to the UIM port or resides outside the device on the External System bus. This is done via the uim_master pin which is driven from the Sgnt signal supplied when the External master has been granted the bus.

The UIM does not allow the memory control signals (eg nwe, ncs & noe) to be input into the UIM port and hence the Memory control signals from the External Master must be multiplexed with the 'normal' outputs (x_nwe_out, x_ncs_out & x_noe_out). The reason for not multiplexing these paths within the UIM is due to the crtitical timing associated with the memory control signals. However the UIM does provide the mutiplexing for the address and data from the External master (uim_add_in and uim_data_in) to the x_add_out pins and hence the Sadd pads. Control for the direction of the data is achieved via the x_noe_in pin. The nsoe_o output from the External Master is muxed back into x_noe_in.

When performing an integration of a Bus Master, consideration must be given to all test mode paths, including the mandatory connections needed for Firefly testing (e.g. Firefly requires that the x_req port is made available as it performs external master type tests). It is important that in this test mode the up integrated master does NOT respond to x_gnt and start to drive the external bus signals.


Inside Final ASIC
Outside Final ASIC
x_tri_data_op
x_data_out<15:0>
x_data_in<15:0>
x_tri_data_ip
SDATA<15:0>
x_tri_addr_op
x_addr_out<17:0>
x_addr_in<17:0>
SADD<17:0>
x_tri_ctrl_op
x_ncs_out<3:0>
x_ncs_in<3:0>
NSCS<3:0>
x_nwe_out<1:0>
x_nwe_in<1:0>
NSWE<1:0>
x_noe_out
NSOE
Firefly
x_noe_in
x_wait
SWAIT
x_gnt
XGNT
Test mode
x_req
XREQ
A/B
A
B
uim_mdata_in
uim_mdata_out
uim_mdata_in
uim_mdata_out
uim_madd_in
uim_madd_out
uim_nmwe<1:0>
uim_nmoe
uim_master
uim_mints<1>
uim_mints<0,2,3>
uim_nmcs<1>
gnd
sdata_o
sdata_i
sdata_o
sdata_i
saddr_o
saddr_i
nswe_i
nsoe_i
ncs
nscs_o
nswe_o<1:0>
nsoe_o
dreq2
dreq1
swait
sbusen_o
sreq
sgnt
External System Master

For more information about all Zarlink products
visit our Web Site at

www.zarlink.com

TECHNICAL DOCUMENTATION - NOT FOR RESALE