ARM 7500

Data Sheet

Document Number: ARM DDI 0050B

Issued: Dec 1994

Change Log

Issue	Date	By	Change
A	Oct 1994	PO/GB/EH/BJH	Created.
B	Dec 1994	PB	Edited.

Pre-release Draft - Unrestricted

Advanced RISC Machines

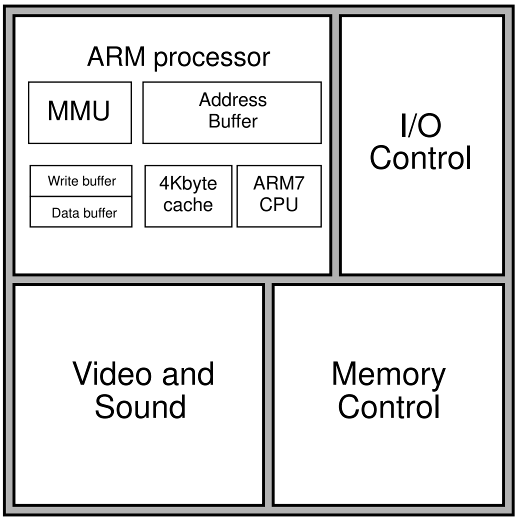
Preface

ARM7500 is a highly integrated single chip computer based around the ARM RISC microprocessor macrocell. ARM7500 contains all the functionality required to create a complete computing system with the minimum of external components.The wide range of features incorporated into ARM7500 make it an extremely flexible device, which can be programmed according to the required application to optimise for high performance or low power, or a combination of both.

Features

- Highly integrated RISC computer
- 30 Dhrystone 2.1 MIPS ARM7 core @ 33MHz
- 4 Kbyte combined instruction and data cache
- Flexible Memory Management Unit
- Supports 16 or 32 bit wide memory via internal ROM and DRAM controllers
- 3 channel DMA
- I/O controller
- 2 serial ports, 4 A/D channels
- 8 stereo sound channels
- 32-bit CD quality serial sound channel
- Video controller with up to 120MHz pixel clock
- 16 million colours from 256-entry palette
- 16-level grey scales for LCD displays
- Suspend and stop power saving modes

Block diagram of the ARM 7500



Applications

ARM7500 is ideally suited to those applications requiring a compact, low-cost, power-efficient, high-performance, RISC computing system on a single chip. These include:

Multimedia	**Interactive visual display terminals**
Portable Computing	**Handheld test instrumentation**
Games consoles	**Desktop computing**

TOC

Contents

1 Introduction

This chapter introduces the ARM7500 single chip microprocessor.

1.1 Introduction

ARM7500 is a high-performance, low-power RISC-based single-chip computer centred around the ARM processor microprocessor core. To maximise the potential of the ARM processor macrocell, ARM7500 contains memory and I/O control on-chip, to enable the direct connection of external memory devices and peripherals with the minimum of external components.

ARM7500 includes features which make it particularly suitable for low-power portable applications. Both 32 and 16-bit wide memory systems are supported, allowing a lower cost 16-bit based system to be designed. The ARM7500 will drive monochrome single or dual panel LCDs with 16 levels of greyscaling, and will also drive colour LCD panels. Power management circuitry is included with two power saving states. The high level of integration achieved allows significant PCB area saving, and results in a very cost competitive part.

ARM7500 is also particularly suited to any application requiring high-quality video, sound and general I/O requirements, such as multimedia. The video controller provides up to 16 million colours from a 256-entry palette, running at up to 120MHz pixel clock rate. The sound subsystem includes an eight channel stereo analog sound interface and a serial sound interface for CD quality 32-bit sound. Four on-chip A to D converters allow the connection of analog joysticks or similar control devices. The clocking scheme is very flexible, allowing a very cheap system to be built using a single 32MHz oscillator while also permitting asynchronous clocks to be used for the CPU, memory and I/O subsystems giving an extremely flexible system, able to take advantage of the fastest available DRAM memory.

The wide range of features incorporated into ARM7500 make it an extremely flexible device, which can be programmed according to the required application to optimise for high performance or low power, or a combination of both.

1.2 Functional block diagram

The block diagram opposite gives a more detailed view of the functionality of the ARM7500 single chip computer.

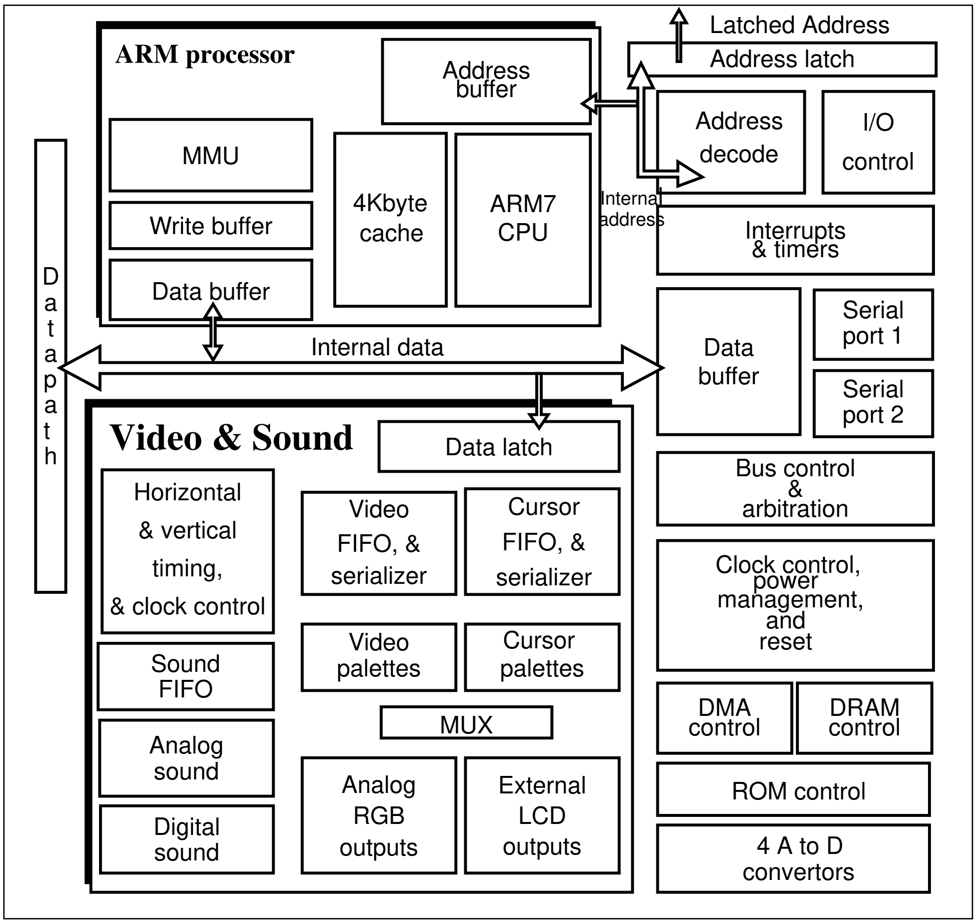


Figure 1-1: Block diagram of ARM7500

1.3 ARM processor CPU

The ARM processor contains an ARM7 core with MMU, Write Buffer, 4K cache, and is identical to the ARM710C macrocell except for the smaller cache.

1.4 Video and sound macrocell

The video and sound macrocell gives the ARM7500 the flexibility to drive high specification CRT or low power LCD displays, and features the following:

- Up to 120MHz pixel clock rate
- fully programmable display parameters
- 256-entry video palette
- Red Green and Blue 8-bit linear DACs to drive CRT
- 1,2,4,8,16,32 bits/pixel CRT modes
- Up to 16 million colours
- External bits in palette for supremacy, fading, Hi_Res
- Single or dual panel LCD driving
- 16-level grey scaler for LCD
- Power management features
- Hardware cursor for all display modes
- Sound system—8 bit analog stereo sound or serial CD digital output

1.5 Clock control and power management

The clocking strategy for ARM7500 has been designed for maximum flexibility, and includes separate clock inputs for the:

- CPU core clock
- Memory system clock
- I/O system clock (in addition to the video clock inputs).

Each of the three clock inputs has a selectable divide-by-two prescaler to generate a 50/50 mark-space ratio if required. Throughout this datasheet, all timing diagrams assume that **CPUCLK**, **MEMCLK**, and **I_OCLK** are divided by one.

There are two levels of power management included.

SUSPEND mode	The clock to the CPU is stopped, but the display continues to work normally.
STOP mode	All clocks are stopped. Two asynchronous wake-up event pins are provided to terminate stop mode. Circuitry is included on chip to stop external oscillators and restart them cleanly when required.

1.6 Memory system

The memory system interface control logic is completely asynchronous in operation to the I/O control logic. This means that the clock to the memory controller can be increased in frequency to allow faster memory to be used. This implementation gives maximum system flexibility.

ARM7500 can control a 32 or 16-bit wide memory system. The width of each bank of ROM or DRAM is selectable by programming appropriate register bits.

A DRAM controller is included which can directly drive up to 4 banks of DRAM. Four NRAS strobes individually select one of the four banks, and four NCAS strobes provide individual byte selection. The DRAM address multiplexing option provided allows a wide variety of DRAM sizes from 256K to beyond 16MB to be used. Up to 256 page mode transfers may occur in one sequential burst.

When configured for operation with a 16-bit DRAM system, the DRAM controller will convert the access into two DRAM cycles to access the two halves of the 32-bit word. Byte transfers will only take one DRAM access cycle, even in 16 bit mode.

A programmable register allows one of four DRAM refresh rates to be selected. In addition, a register is provided to enable direct software control of the NCAS and NRAS lines for setting DRAM into a self-refresh state.

A ROM controller supports two 16MB banks of ROM with individually programmable read cycle timings. Support is provided for burst mode reads. Each ROM bank can be programmed to operate in 16-bit wide mode, and like the DRAM controller will convert accesses into two ROM cycles for the two halves of the 32 bit word.

1.6.1 DMA

Three fully programmable DMA channels are included, for video, cursor and sound data. The DMA controller includes additional support for dual panel LCDs.

1.6.2 I/O control

The I/O bus of ARM7500 is 16-bits wide but for some types of access can be expanded to 32 bits by the use of external transceivers. The input clock I_OCLK provides a reference for the I/O subsystem which is nominally 32MHz.

The I/O features of this device can be separated into 3 distinct cycle types:

- Simple I/O with fixed 8MHz timings
- Module I/O with variable length 8MHz timings
- PC bus style I/O with fixed 16MHz timings and support for 32-bit data

Simple I/O

The Simple I/O type of access is 16-bit only and has a selection of 4 different cycle speeds selectable by address. When writing, the upper half-word of the ARM data bus is written out on the I/O bus. When reading, the I/O bus data is read back onto the lower half-word of the ARM data bus.

During these accesses, a chip select is asserted with the appropriate **nIOR**/**nIOW** read or write strobe, based on the 8MHz clock **CLK8**.

Module I/O

The Module I/O type of access is 16 bit only and its timing is controlled by a handshake mechanism with the external hardware. The signals NIORQ (output) and NIOGT (input) are used for this handshaking and are referenced to REF8M. When writing, the upper half-word of the ARM data bus is written out on the I/O bus. When reading, the I/O bus data is read back onto the lower half-word of the ARM data bus.

During these accesses, a chip select is asserted but the **nIOR**/**nIOW** read and write strobes are not used, although the **IORNW** signal is active.

PC bus style I/O

The PC bus style I/O type of access routes the lower half-word of the ARM bus through the device providing a direct 16-bit interface. Signals are generated to support the addition of external latches/drivers to extend the I/O data by 16 bits. The upper half-word of the ARM data bus is routed through these external devices if present.

There are 5 different address areas generating 5 different chip selects using the same type of access. There are 4 fixed cycle types based on the 16MHz clock, although the largest area only supports two of these cycle types. Any access may be held up by external circuitry removing the **READY** signal before the end of the cycle.

During these accesses, the relevant chip select is asserted as well as read or write strobes as appropriate.

Two special inputs are provided to allow external circuitry to route the full 32 bits through the 16-bit I/O bus using multiplexing. This would allow, for example, the execution of code from a 16-bit PCMCIA card with suitable external controller. On a read I/O, if this latching signal is used, the data read back onto the ARM data bus comes from the I/O bus instead of the external extension latches.

1.7 Other features

ARM7500 includes four analog comparators, which can be used to create four A to D converter channels, and two serial keyboard/mouse ports. There are 8 general-purpose open-drain I/O lines which can be used as inputs or open drain outputs and as interrupt sources if required. An interrupt handler processes a variety of internal and external interrupt sources to generate the IRQ and FIQ interrupts for the ARM processor.

1.8 Test modes

ARM7500 has an **nTEST** pin which is used to invoke various test modes. When **nTEST** is set LOW, the functionality of many of the pins will change depending on the values applied to the **nINT3**, **nINT6** and **nINT8** pins. These test modes are intended purely for production test. The **nTEST** pin includes an on-chip pull-up, but it is recommended that the pin be pulled up to VDD externally too.

Note: *The **nTEST** pin should never be forced LOW.*

1.9 Structure of ARM7500

ARM7500 is based around two modified ARM macrocells, the ARM processor and video and sound macrocells. These macrocells are self-contained and the relevant control registers are contained within them. This has the effect that there are three sets of programmable registers within the ARM7500, which are accessed in different ways depending on their location.

The ARM processor register programming is described in *Chapter 4: ARM Processor Programmer's Model* . The video and sound macrocell's registers are programmed using only the internal ARM7500 data bus (the address bus is not passed to the macrocell). The address 0x03400000 is decoded to provide a write strobe for the video macrocell registers, and the addressing of registers within the macrocell is decoded from the upper four or eight bits of the data word. This system is described more fully in *Chapter 9: Video and Sound Programmer's Model* .

The remaining ARM7500 registers, associated with Memory, I/O and general miscellaneous control, form a separate group and are programmed between addresses 0x03200000 and 0x032001F8. The majority of the registers are only eight bits wide, although all register addresses are word aligned. These registers are described in *Chapter 13: Memory and I/O Programmer's Model* .

Interaction between the macrocells occurs mainly across the ARM7500's internal 32-bit data bus, which is routed to the two main macrocells, and most of the other memory and I/O control logic. The ARM processor's address bus is routed to an internal address decoder where memory space is decoded to determine required cycle types and register addresses. The same address bus is latched and exported from the chip as the **LA[28:0]** bus. Only these 29 bits of the address bus are available externally.

1.10 Resetting ARM7500 systems

The ARM7500 is designed to operate with both 16 and 32-bit wide ROM, which means that it must be capable of booting from either. To achieve this, the chip is always reset into 16-bit mode, which might be expected to cause difficulty when the chip is being booted up from 32-bit ROM. However, *Appendix A.1: Initialisation and Boot Sequence* describes a simple code sequence which will allow the chip to be started up without difficulty under these circumstances.

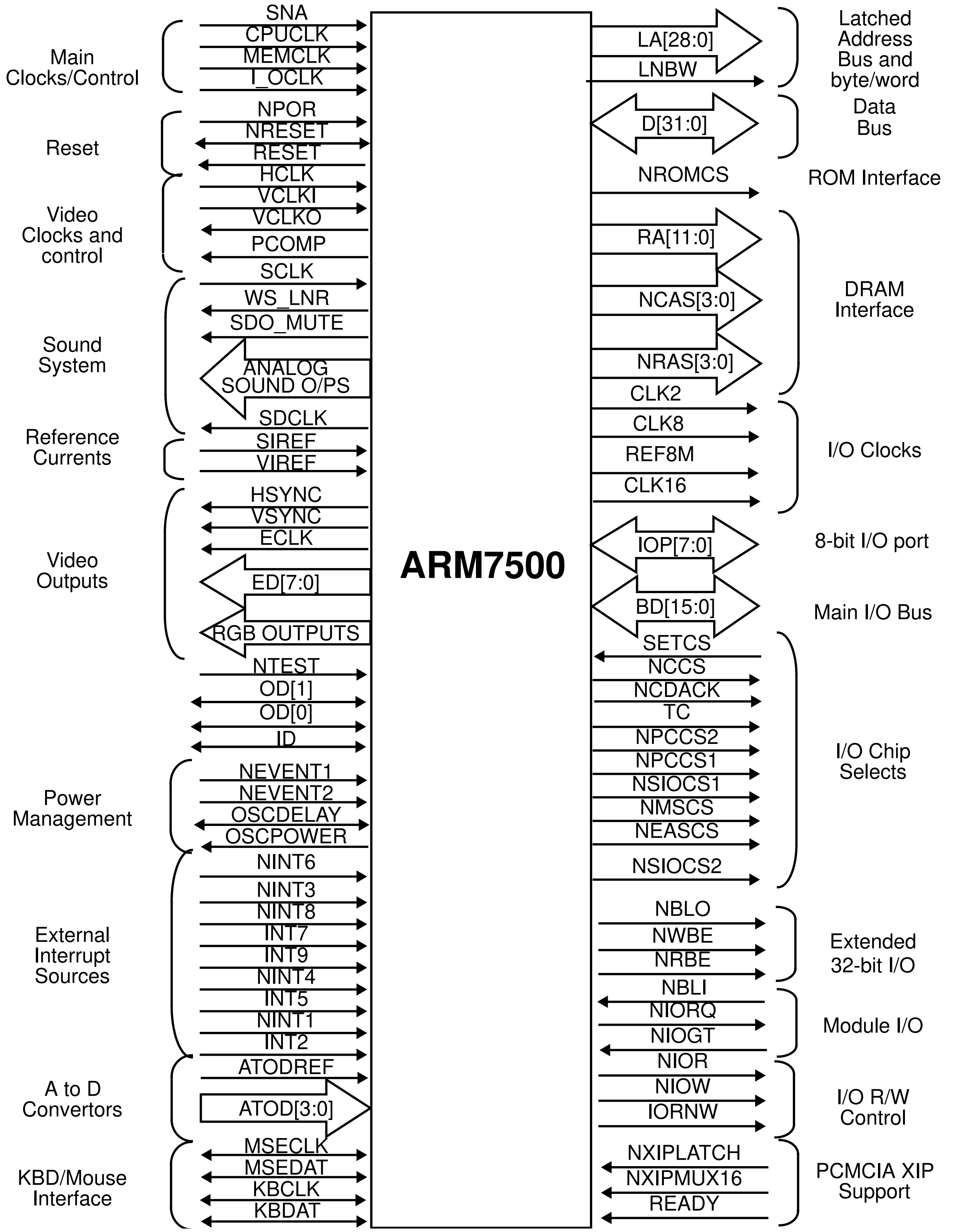
1.11 Datasheet Notation

0x	marks a Hexadecimal quantity
BOLD	external signals are shown in bold capital letters
binary	where it is not clear that a quantity is binary it is followed by the word binary

2 Signal Description

This chapter gives the name, type, and relevant details of each of the ARM7500 signals.

Signal Description



Pre-release Draft - Unrestricted

2.1 Signal description for ARM7500

Note: *When output signals are placed in the high impedance state for long periods, care must be taken to ensure that they do not float to an undefined logic level.*

Key to signal types:

IC	Input, CMOS threshold
OCZ	Output, CMOS levels, tri-stateable
IT	TTL threshold input
ICS	CMOS Schmitt input
IA	Analog input
OA	Analog output
BTZ	Bidirectional with CMOS output and TTL threshold input level
TOD	Open drain with TTL input
CSOD	Open drain with CMOS schmitt input
IAOD	Analog input with programmable internal pull-down transistor

Name	Type	Description
LA[28:0]	OCZ	Latched address bus. This bus is the latched version of the ARM address for memory accesses, changing on the falling edge of the internal MCLK signal.
D[31:0]	BTZ	The main data bus for the ARM7500. All external data transfers happen via this bus. When the ARM7500 is configured for operation in 16-bit mode, only the lower 16 bits are used.
SnA	IC	Synchronous/not Asynchronous. This pin is set according to the relationship required between the internal clock signals MCLK and FCLK for the ARM. If this pin is set HIGH, both the memory system and the CPU are driven from the **MEMCLK** pin, and the required synchronous timing relationship between the ARM processor clocks is generated automatically on-chip. If different clocks are to be used, for the **MEMCLK** and **CPUCLK** inputs, the **SnA** pin must be set LOW.
BOUT	AO	Blue Analog Output. The video signal analog outputs are designed to drive doubly-terminated 75 lines.
ECLK	OCZ	External Clock. When enabled, this clock validates the data on ED[7:0]. In normal video mode, it runs at the pixel rate, but when LCD data is being produced, it runs at a quarter of the pixel rate.
ED[7:0]	OCZ	External Data. This is the digital video output port of the ARM7500. From this, the digital equivalent of the analog output may be produced in any colour, or data from the external palette may be produced. This may be used for a variety of purposes such as fading or supremacy. Also, data for driving LCD panels is output from this port. Data produced is validated by **ECLK**.

Table 2-1: ARM7500 signal description

Name	Type	Description
GOUT	AO	Green Analog Output. The video signal analog outputs are designed to drive doubly-terminated 75Ω lines.
HCLK	IT	High speed Clock for use with video subsystem.
HSYNC	OCZ	Horizontal Synchronisation. There are two synchronisation outputs on ARM7500, **HSYNC** and **VSYNC**. Dependent on the state of bits 17 and 16 in the video External register, either a horizontal or a composite (NOR) sync may be output on this pin, in either polarity. The width of the **HSYNC** pulse is definable in units of 2 pixels.
LM	AO	Left Minus Analog sound. Negative analog sound output for the left channel.
LP	AO	Left Plus Analog sound. Positive analog sound output for the left channel.
PCOMP	OCZ	Phase Comparator Output for use with VCLK pins.
ROUT	AO	Red Analog Output. The video signal analog outputs are designed to drive doubly-terminated 75Ω lines.
RM	AO	Right Minus Analog Sound. Negative analog right hand stereo channel sound output.
RP	AO	Right Plus Analog Sound. Positive analog right stereo channel sound output.
SCLK	IT	Sound Clock. This signal can be used to clock the sound system, when a clock asynchronous to the internal video reference clock is required.
SDCLK	OCZ	Serial Data Clock. When the sound system is in serial sound mode, this clock is output and validates serial data on its rising edge.
SDO_MUTE	OCZ	Serial Data Out / Mute. This pin has two functions depending on whether the sound mode is either analog or digital serial sound. In digital mode, serial sound data is output from this pin. In analog mode, this signal goes HIGH between samples to allow for DAC settling.
SYNC	IT	External SYNC. This signal is used to synchronise ARM7500 with another video system.
SIREF	IA	Sound Reference Current. A reference current must be fed into this pin in order to calibrate the sound DAC outputs. For most applications, a resistor to **VDD** is sufficient, although a constant current source is recommended.
VCLKI	IC	Phase Comparator Clock In (for video subsystem).
VCLKO	OCZ	Phase Comparator Clock Out (for video subsystem).
VDD_Analog		Positive (+5V) supply for analog video system.
VDD_Sound		Positive (+5V) supply for analog sound system.

Table 2-1: ARM7500 signal description (Continued)

Name	Type	Description
VIREF	IA	Video Reference Current. The video DACs need a reference current in order to calibrate them. A constant current source is recommended, although a resistor up to **VDD** is sufficient for many applications. This current also generates the constant source for the A to D comparators.
VSS_Analog		Supply ground for analog video system.
VSS_Sound		Supply ground for analog sound system.
VSYNC	OCZ	Vertical Synchronisation. Dependent on the state of bits 19 and 18 in the external register, either a vertical or a composite (XNOR) sync may be output on this pin, in either polarity. The width of the **VSYNC** pulse may be defined in units of a raster.
WS_LnR	OCZ	Word Select / Left NOT Right. Again, this pin has two functions depending on whether the sound mode is either analog or digital. In digital mode, this signal denotes whether the output serial data is for the left hand stereo channel or the right hand channel. In analog mode this signal gives the same stereo direction information.
nTEST	IT	Test mode input. This pin should be held permanently HIGH. It is only intended to be used during production test of the ARM7500. An on-chip pull-up is included, but it is advisable to fit an external pull-up resistor to this pin.
RA[11:0]	OCZ	DRAM row/column multiplexed address bus. Addresses for this bus are decoded from the ARM processor address for normal memory accesses, and are generated by the DMA controller for DMA.
nRAS[3:0]	OCZ	DRAM row address strobes. Each of these selects one of the four banks of DRAM available.
nCAS[3:0]	OCZ	DRAM column address strobes. These select the byte within the word for DRAM accesses.
VDD_ATOD	power	Positive 5V supply for the A to D converter comparators
VSS_ATOD	power	Analog ground for the A to D converter comparators
ATOD[3:0]	IAOD	Four A to D channel input voltages.
ATODREF	IA	Reference voltage for the A to D converter comparators..
OSCPOWER	OCZ	Enable signal for the system oscillator(s). When LOW, this signal can be used to disable the external oscillator(s).
OSCDELAY	CSOD	Requires an RC network to generate a fixed delay when restarting the system oscillator(s) on exit from STOP mode.
RESET	OCZ	Reset output, synchronised version of internal system reset signal.
nRESET	CSOD	Open drain output and ‘soft’ reset input. This pin is sampled every 1µs for reset events, so to guarantee a successful reset, a reset pulse applied to this pin must be longer than 1µs.

Table 2-1: ARM7500 signal description (Continued)

Name	Type	Description
nROMCS	OCZ	ROM Chip select. Goes LOW to indicate a ROM access.
I_OCLK	IC	I/O system clock. This clock input should always be 32MHz when in divide by 1 mode, and 64MHz in divide by 2 mode.
MEMCLK	IC	Memory system clock. In synchronous mode, both the memory system and the ARM processor FCLK are driven from this clock.
CPUCLK	IC	Clock used to create FCLK for the ARM CPU in asynchronous mode. When SnA is HIGH this should be tied HIGH or LOW permanently.
BD[15:0]	BTZ	The main external 16-bit I/O bus.
MSCLK	TOD	Mouse clock. An open drain pin for the mouse PS/2 interface.
MSDATA	TOD	Mouse data. An open drain pin for the mouse PS/2 interface.
KBCLK	TOD	Keyboard clock. An open drain pin for the keyboard PS/2 interface.
KBDATA	TOD	Keyboard data. An open drain pin for the keyboard PS/2 interface.
nPOR	ICS	Power on reset. Any LOW transitions on this pin are detected and stretched to ensure full reset.
IOP[7:0]	TOD	8 bit wide I/O port. Each bit is directly controllable via an ARM7500 register, and can be used as an interrupt source if required.
ID	TOD	The ID pin can be used to activate a system ID chip. It is forced LOW during the power on reset sequence.
OD[1:0]	TOD	Two open drain pins which (unlike the **IOP[7:0]** bus) cannot be used to generate interrupts, but can be used as general purpose I/O pins, for example to communicate with a real time clock chip.
SETCS	IC	**SETCS** selects between two address decoding options for the three main I/O chip selects. It affects the outputs **nEASCS**, **nMSCS** and **nSIOCS2**.
nINT1	IT	Falling edge triggered interrupt pin. This pin also has the feature that its value can be read directly in the IOCR I/O control register.
INT2	IT	Rising edge triggered interrupt pin. Can generate an IRQ interrupt.
nINT3	IT	Active LOW interrupt pin. ARM7500 can be programmed to generate an IRQ from this pin.
nINT4	IT	Active LOW interrupt pin, which can be used to generate an IRQ interrupt.
INT5	IT	Active HIGH interrupt pin. Can be used to generate either an IRQ or a FIQ interrupt, depending on the status of the relevant mask register bits.
nINT6	IT	Active LOW interrupt pin. Can generate either an IRQ or a FIQ depending on the programming of the mask registers.
INT7	IT	Active HIGH interrupt pin. Can be used to generate an IRQ interrupt.

Table 2-1: ARM7500 signal description (Continued)

Name	Type	Description
nINT8	IT	Active LOW interrupt pin. Can be used to generate either a FIQ or an IRQ interrupt.
INT9	IT	Active HIGH interrupt pin, which can only be used to generate a FIQ (highest priority) interrupt.
nEVENT1	IT	Active LOW asynchronous event pin 1. A falling edge is used to terminate STOP or SUSPEND power saving modes.
nEVENT2	IT	Active LOW asynchronous event pin 2. A falling edge is used to terminate STOP or SUSPEND power saving modes.
READY	IT	Can be used to stretch I/O accesses when set LOW during a 16MHz PC-style I/O cycle.
nIORQ	OCZ	I/O request signal used for Module type I/O for handshaking, together with **nIOGT**.
nIOGT	IT	I/O grant signal used for Module type I/O for handshaking, together with **nIORQ**.
nBLI	IT	Input used during Module-style I/O reads to cause the latching of data from the BD port.
nBLO	OCZ	Latching signal for use with external latches on the upper 16 bits of the external datapath to create a 32-bit wide I/O bus.
nRBE	OCZ	Active LOW Read enable for an external transceiver attached to the upper 16 bits of the I/O bus, to create a 32-bit wide I/O bus.
nWBE	OCZ	Active LOW Write enable for an external transceiver attached to the upper 16 bits of the I/O bus, to create a 32-bit wide I/O bus.
nXIPMUX16	IT	For Execute in place (XIP) support. This signal multiplexes 16 bits of data from the upper or lower halfword of the ARM7500 internal data bus to the 16-bit I/O bus, depending on its state during writes.
nXIPLATCH	IT	For XIP support. Latches the upper 16 bits of data from the I/O bus while the lower 16 bits are being read. Used in conjunction with **nXIPMUX16** to enable XIP from, for example, a 16-bit PCMCIA card.
nSIOCS1	OCZ	Active LOW chip select for simple I/O.
nSIOCS2	OCZ	Active LOW chip select for simple I/O, with address decode modified according to the state of **SETCS**.
nMSCS	OCZ	Active LOW chip select for module type I/O, with address decode modified according to the state of **SETCS**.
nEASCS	OCZ	Active LOW chip select for extended 16Mhz PC-style I/O, with address decode modified according to the state of **SETCS**.
nCCS	OCZ	Not Combo Chip Select. Chip select signal for a PC Combo chip.

Table 2-1: ARM7500 signal description (Continued)

Name	Type	Description
nCDACK	OCZ	Not Combo Dack. Chip select and Dack signal for PC Combo chip.
TC	OCZ	Active HIGH terminal count. Used in conjunction with the **nCDACK** signal for pseudo DMA to a Combo chip.
nPCCS1	OCZ	Active LOW chip select for an area of 16Mhz PC-style I/O space.
nPCCS2	OCZ	Active LOW chip select for an area of 16Mhz PC-style I/O space.
LNBW	OCZ	Latched Not Byte word signal. This is a latched version of the internal NBW signal from the ARM processor, changing on the falling edge of the internal MCLK signal.
IORNW	OCZ	I/O read/not write, HIGH during an I/O read, and LOW during an I/O write.
NIOR	OCZ	Not I/O read. Is LOW during simple and PC-style I/O reads. Not used for Module type I/O.
NIOW	OCZ	Not I/O write. Is LOW during simple and PC-style I/O reads. Not used for Module type I/O.
CLK2	OCZ	2MHz I/O clock output.
CLK8	OCZ	8MHz I/O clock output, the inverted version of REF8M.
REF8M	OCZ	8MHz I/O clock output.
CLK16	OCZ	16MHz I/O clock output, for PC-style I/O.

Table 2-1: ARM7500 signal description (Continued)

3 The ARM Processor Macrocell

This chapter introduces the ARM processor 32-bit microprocessor macrocell.

3.1 Introduction

The ARM7500 contains a 32-bit RISC ARM processor, similar to the ARM710C macrocell. It has a 4Kbyte cache, write buffer, and a Memory Management Unit (MMU). The ARM processor macrocell offers high-level RISC performance, yet its fully static design ensures minimal power consumption. This makes it ideal for incorporation into the ARM7500. The ARM7500 aims to make maximum use of the performance and flexibility offered by the ARM processor.

This part of the datasheet describes the features of the ARM processor macrocell which are available to the user in its embedded state within the ARM7500 single- chip computer. It is not intended that this should be used as a standalone datasheet for a separate ARM processor macrocell.

3.1.1 Architecture

The ARM processor architecture is based on 'Reduced Instruction Set Computer' (RISC) principles, and the instruction set and related decode mechanism are greatly simplified compared with microprogrammed 'Complex Instruction Set Computers' (CISC).

The mixed data and instruction cache together with the write buffer substantially raise the average execution speed and reduce the average amount of memory bandwidth required by the processor. This allows the ARM7500 bus structure to support Direct Memory Access (DMA) channels with minimal performance loss.

The MMU supports a conventional two-level page-table structure and a number of extensions which make it ideal for embedded control, UNIX and Object Oriented systems.

3.2 Instruction set

The instruction set comprises ten basic instruction types:

- two of these make use of the on-chip arithmetic logic unit, barrel shifter and multiplier to perform high-speed operations on the data in a bank of 31 registers, each 32 bits wide
- three classes of instruction control data transfer between memory and the registers, one optimized for flexibility of addressing, another for rapid context switching and the third for swapping data
- two instructions control the flow and privilege level of execution
- three types are dedicated to the control of external coprocessors which allow the functionality of the instruction set to be extended in an open and uniform way. However, as for the ARM710, the facility to add external coprocessors to the ARM7500 is not available, and software emulation of coprocessor activity will be required if these instructions are to perform a defined function.

The ARM instruction set is a good target for compilers of many different high-level languages. Where required for critical code segments, assembly code programming is also straightforward, unlike some RISC processors which depend on sophisticated compiler technology to manage complicated instruction interdependencies.

3.3 Memory interface

The memory interface has been designed to allow the performance potential to be realised without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals permit the ARM7500 to exploit the paged mode access offered by industry-standard DRAMs.

3.4 Clocks and Synchronous/Asynchronous modes

The ARM processor uses two independent clock sources, MCLK and FCLK. Both are generated internally to ARM7500 from MEMCLK and CPUCLK. The ARM7 core CPU switches between MCLK and FCLK according to the operation being carried out. For example, if the ARM7 core CPU is reading data from the cache it will be clocked by FCLK, whereas if the core CPU is reading data from uncached memory then it will be clocked by MCLK. The ARM processor’s control logic ensures that the correct clock is used internally and switches between the two clocks automatically.

When SnA is tied high MEMCLK creates both FCLK and MCLK, with MCLK having half the frequency of FCLK. This synchronous mode ensures that there are no synchronisation penalties whenever the ARM 7 core is switched between FCLK and MCLK.

When SnA is tied low, MEMCLK creates MCLK and CPUCLK must be driven to supply FCLK. MEMCLK and CPUCLK can be of unrelated frequency. There is a synchronisation penalty whenever the ARM7 core clock switches between MCLK and FCLK. This penalty is symmetric, and varies between nothing and a whole period of the clock to which the core is resynchronising. Thus when changing there is an average resynchronisation penalty of half a clock period, MCLK or FCLK as appropriate.

3.5 ARM processor Block diagram

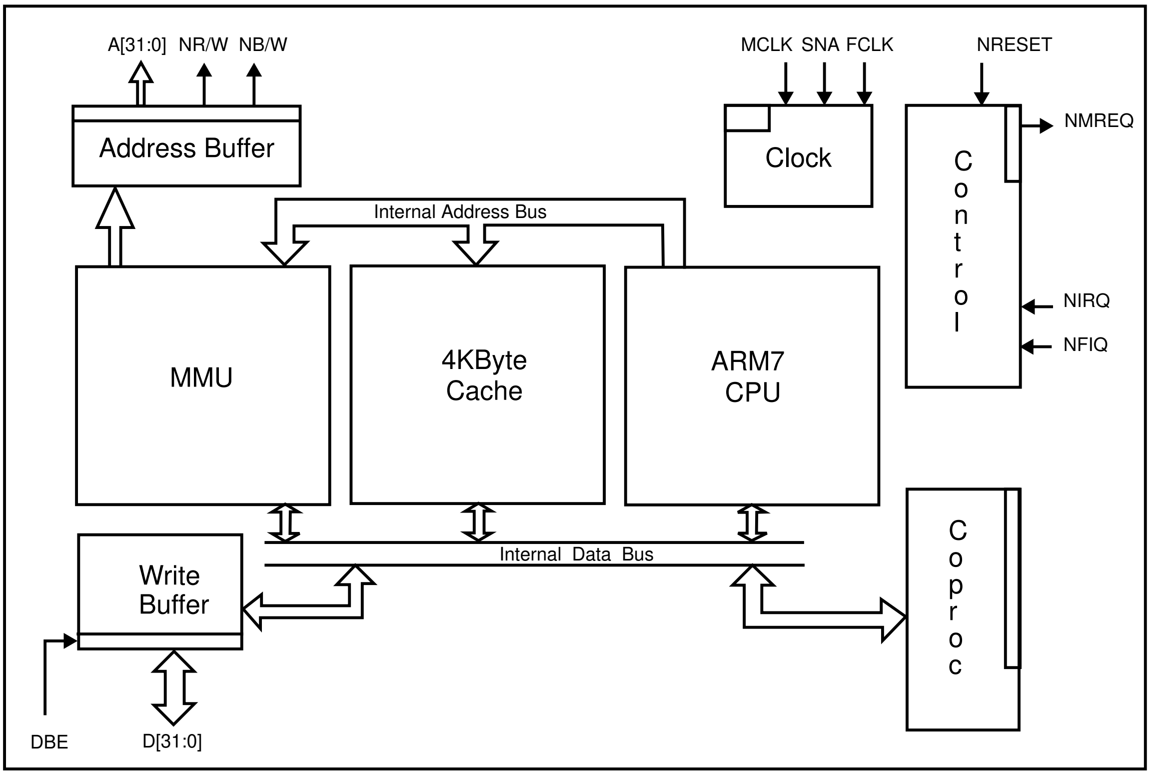


Figure 3-1: ARM processor block diagram

4 ARM Processor Programmer's Model

This chapter details the ARM Processor programmable registers.

4.1 Introduction

The ARM Processor supports a variety of operating configurations. Some are controlled by register bits and are known as the *configurations*. Others may be controlled by software and these are known as *operating modes*.

4.2 Register configuration

The ARM Processor provides 3 register configuration settings which may be changed while the processor is running and which are discussed below.

4.2.1 Big and Little Endian (the bigend bit)

The bigend bit, in the Control Register, sets whether the ARM7500 treats words in memory as being stored in Big Endian or Little Endian format.

Memory is viewed as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on.

Little Endian

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 (**D[7:0]**) 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
Lower Address	**3**	**2**	**1**	**0**	0

- Least significant byte is at lowest address
- Word is addressed by byte address of least significant byte

Figure 4-1: Little Endian addresses of bytes within words

Big Endian

The most significant byte of a word is stored at the lowest numbered byte and the least significant byte is stored at the highest numbered byte. Byte 0 of the memory system should therefore be connected to data lines 31 through 24 (**D[31:24]**). Load and store are the only instructions affected by the endianism.

Big Endian

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

- Most significant byte is at lowest address
- Word is addressed by byte address of most significant byte

Figure 4-2: Big Endian addresses of bytes within words

4.2.2 Configuration bits for backward compatibility

Two register bits, PROG32 and DATA32, allow one of three processor configurations to be selected:

1. **26-bit program and data space**

   (PROG32 LOW, DATA32 LOW). This configuration forces ARM Processor to operate like the earlier ARM processors with 26-bit address space. The programmer's model for these processors applies, but the new instructions to access the CPSR and SPSR registers operate as detailed in ➲*5.5 PSR transfer (MRS, MSR)* on page 5-15. In this configuration it is impossible to select a 32-bit operating mode, and all exceptions (including address exceptions) enter the exception handler in the appropriate 26-bit mode.

2. **26-bit program space and 32-bit data space**

   (PROG32 LOW, DATA32 HIGH). This is the same as the 26-bit program and data space configuration, but with address exceptions disabled to allow data transfer operations to access the full 32-bit address space.

3. **32-bit program and data space**

   (PROG32 HIGH, DATA32 HIGH). This configuration extends the address space to 32 bits, introduces major changes in the programmer's model and provides support for running existing 26-bit programs in the 32-bit environment.

(The fourth processor configuration (26-bit data space and 32-bit program space) should not be selected.)

26-bit program space

When configured for 26-bit program space, ARM7500 is limited to operating in one of four modes known as the 26-bit modes. These modes correspond to the modes of the earlier ARM processors and are known as:

- User26
- FIQ26
- IRQ26 and
- Supervisor26

Note: The PROG32 and DATA32 bits are used only for backward compatibility with earlier ARM processors and should normally be set to 1. The 32-bit mode is recommended for compatibility with future ARM processors and all new code should be written to use only the 32-bit operating modes.

Because the original ARM instruction set has been modified to accommodate 32-bit operation there are certain additional restrictions which programmers must be aware of. Reference should also be made to the *ARM Application Notes "Rules for ARM Code Writers"* and *"Notes for ARM Code Writers"* available from your supplier.

4.3 Operating mode selection

ARM Processor has a 32-bit data bus and a 32-bit address bus. However, only 29 of the address bits are available at the ARM7500 pins. The data types the processor supports are Bytes (8-bits) and Words (32-bits), where words must be aligned to four byte boundaries. Instructions are exactly one word, and data operations (e.g. ADD) are only performed on word quantities. Load and store operations can transfer either bytes or words.

ARM Processor supports six modes of operation:

User mode (usr)	The normal program execution state.
FIQ mode (fiq)	Designed to support a data transfer or channel process.
IRQ mode (irq)	Used for general purpose interrupt handling.
Supervisor mode (svc)	A protected mode for the operating system.
Abort mode (abt)	Entered after a data or instruction prefetch abort.
Undefined mode (und)	Entered when an undefined instruction is executed.

Mode changes may be made under software control or may be brought about by external interrupts or exception processing. Most application programs execute in User mode. The other modes, known as *privileged modes*, are entered to service interrupts or exceptions, or to access protected resources.

4.4 Registers

The processor macrocell has a total of 37 registers made up of 31 general 32-bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode, and the other registers (the *banked registers*) are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing. The register bank organisation is shown in *Figure 4-3: Register organisation* on page 4-5. The banked registers are shaded in the diagram.

General Registers and Program Counter Modes

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

Figure 4-3: Register organisation

In all modes, 16 registers (R0 to R15) are directly accessible. All registers except R15 are general purpose and may be used to hold data or address values. Register R15 holds the Program Counter (PC). When R15 is read, bits [1:0] are zero and bits [31:2] contain the PC. A seventeenth register (the CPSR - Current Program Status Register) is also accessible. It contains condition code flags and the current mode bits and may be thought of as an extension to the PC.

R14 is used as the subroutine link register and receives a copy of R15 when a Branch and Link instruction is executed. It may be treated as a general purpose register at all other times. R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are used similarly to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). Many FIQ programs will not need to save any registers.

User mode, IRQ mode, Supervisor mode, Abort mode and Undefined mode each have two banked registers mapped to R13 and R14. The two banked registers allow these modes to each have a private stack pointer and link register.

Supervisor, IRQ, Abort and Undefined mode programs which require more than these two banked registers are expected to save some or all of the caller's registers (R0 to R12) on their respective stacks. They are then free to use these registers which they will restore before returning to the caller.

In addition there are also five SPSRs (Saved Program Status Registers) which are loaded with the CPSR when an exception occurs. There is one SPSR for each privileged mode.

4.4.1 Program status registers

The format of the Program Status Registers is shown in *Figure 4-4: Format of the Program Status Registers (PSRs).*

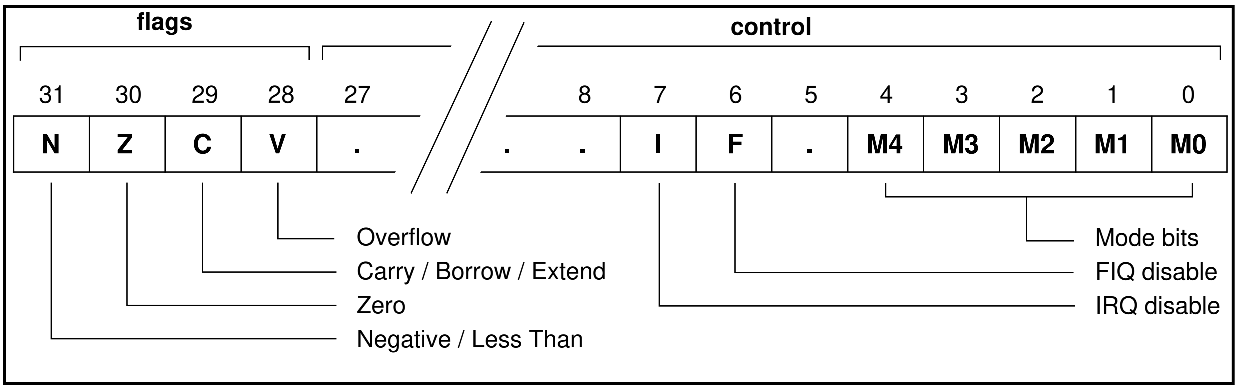


Figure 4-4: Format of the Program Status Registers (PSRs)

Condition code flags

The N, Z, C and V bits are the *condition code flags*. The condition code flags in the CPSR may be changed as a result of arithmetic and logical operations in the processor and may be tested by all instructions to determine if the instruction is to be executed.

Interrupt disable bits

The I and F bits are the *interrupt disable bits.* The I bit disables IRQ interrupts when it is set and the F bit disables FIQ interrupts when it is set.

Mode bits

The M0, M1, M2, M3 and M4 bits (M[4:0]) are the *mode bits*, and these determine the mode in which the processor operates. The interpretation of the mode bits is shown in the following table. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used.

M[4:0]	Mode	Accessible register set	
10000	User	PC, R14..R0	CPSR
10001	FIQ	PC, R14_fiq..R8_fiq, R7..R0	CPSR, SPSR_fiq
10010	IRQ	PC, R14_irq..R13_irq, R12..R0	CPSR, SPSR_irq
10011	Supervisor	PC, R14_svc..R13_svc, R12..R0	CPSR, SPSR_svc
10111	Abort	PC, R14_abt..R13_abt, R12..R0	CPSR, SPSR_abt
11011	Undefined	PC, R14_und..R13_und, R12..R0	CPSR, SPSR_und

Table 4-1: The mode bits

Control bits

The bottom 28 bits of a PSR (incorporating I, F and M[4:0]) are known collectively as the *control bits*. The control bits change when an exception arises and in addition can be manipulated by software when the processor is in a privileged mode. Unused bits in the PSRs are reserved and their state must be preserved when changing the flag or control bits. Programs must not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

4.5 Exceptions

Exceptions arise whenever there is a need for the normal flow of program execution to be broken. For example, the processor can be diverted to handle an interrupt from a peripheral. The processor state just prior to handling the exception must be preserved so that the original program can be resumed when the exception routine has completed. Many exceptions may arise at the same time.

ARM Processor handles exceptions by making use of the banked registers to save state. The old PC and CPSR contents are copied into the appropriate R14 and SPSR and the PC and mode bits in the CPSR bits are forced to a value which depends on the exception. Interrupt disable flags are set where required to prevent otherwise unmanageable nestings of exceptions. In the case of a re-entrant interrupt handler, R14 and the SPSR should be saved onto a stack in main memory before re-enabling the interrupt.

Note: When transferring the SPSR register to and from a stack, it is important to transfer the whole 32-bit value, and not just the flag or control fields.

When multiple exceptions arise simultaneously, a fixed priority determines the order in which they are handled. The priorities are listed in ➲*4.5.7 Exception priorities* on page 4-12.

4.5.1 FIQ

The FIQ (Fast Interrupt reQuest) exception is generated by the interrupt handler within the ARM7500. This input is delayed by one clock cycle for synchronisation before it can affect the processor execution flow. It is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications (thus minimising the overhead of context switching).

Note: The FIQ exception may be disabled by setting the F flag in the CPSR (but note that this is not possible from User mode).

If the F flag is clear, ARM Processor checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction. When a FIQ is detected, ARM Processor performs the following:

1. Saves the address of the next instruction to be executed plus 4 in R14_fiq; saves CPSR in SPSR_fiq.
2. Forces M[4:0]=10001 (FIQ mode) and sets the F and I bits in the CPSR.
3. Forces the PC to fetch the next instruction from address 0x1C.

Returning from FIQ

To return normally from FIQ, use SUBS PC, R14_fiq,#4 which will restore both the PC (from R14) and the CPSR (from SPSR_fiq) and resume execution of the interrupted code.

4.5.2 IRQ

The IRQ (Interrupt ReQuest) exception is a normal interrupt caused by the interrupt handler within the ARM7500. It has a lower priority than FIQ, and is masked out when a FIQ sequence is entered. Its effect may be masked out at any time by setting the I bit in the CPSR (but note that this is not possible from User mode).

If the I flag is clear, ARM Processor checks for a LOW level on the output of the IRQ synchroniser at the end of each instruction. When an IRQ is detected, ARM Processor performs the following:

1. Saves the address of the next instruction to be executed plus 4 in R14_irq; saves CPSR in SPSR_irq
2. Forces M[4:0]=10010 (IRQ mode) and sets the I bit in the CPSR
3. Forces the PC to fetch the next instruction from address 0x18

Returning from IRQ

To return normally from IRQ, use SUBS PC,R14_irq,#4 which will restore both the PC and the CPSR and resume execution of the interrupted code.

4.5.3 Abort

An ABORT is signalled by the internal Memory Management Unit, and indicates that the current memory access cannot be completed. For instance, in a virtual memory system the data corresponding to the current address may have been moved out of memory onto a disc, and considerable processor activity may be required to recover the data before the access can be performed successfully.

The abort mechanism allows a *demand paged virtual memory system* to be implemented when suitable memory management software is available. The processor is allowed to generate arbitrary addresses, and when the data at an address is unavailable the MMU signals an abort. The processor traps into system software which must work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

ARM Processor checks for ABORT during memory access cycles. When successfully aborted ARM Processor responds in one of two ways:

- prefetch abort
- data abort

Prefetch abort

If the abort occurred during an instruction prefetch (a *prefetch abort*), the prefetched instruction is marked as invalid but the abort exception does not occur immediately. If the instruction is not executed, for example as a result of a branch being taken while it is in the pipeline, no abort will occur. An abort will take place if the instruction reaches the head of the pipeline and is about to be executed.

Data abort

If the abort occurred during a data access (a *data abort*), the action depends on the instruction type:

- single data transfer instructions (LDR, STR) write back modified base registers and the Abort handler must be aware of this
- the swap instruction (SWP) is aborted as though it had not executed, though externally the read access may take place
- block data transfer instructions (LDM, STM) complete, and if write-back is set, the base is updated. If the instruction would normally have overwritten the base with data (i.e. LDM with the base in the transfer list), this overwriting is prevented. All register overwriting is prevented after the Abort is indicated, which means in particular that R15 (which is always last to be transferred) is preserved in an aborted LDM instruction.

Abort sequence

When either a prefetch or data abort occurs, ARM Processor performs the following:

1. Saves the address of the aborted instruction plus 4 (for prefetch aborts) or 8 (for data aborts) in R14_abt; saves CPSR in SPSR_abt
2. Forces M[4:0]=10111 (Abort mode) and sets the I bit in the CPSR
3. Forces the PC to fetch the next instruction from either address 0x0C (prefetch abort) or address 0x10 (data abort)

Returning from an abort

To return after fixing the reason for the abort, use SUBS PC,R14_abt,#4 (for a prefetch abort) or SUBS PC,R14_abt,#8 (for a data abort). This will restore both the PC and the CPSR and retry the aborted instruction.

4.5.4 Software interrupt

The software interrupt instruction (SWI) is used for getting into Supervisor mode, usually to request a particular supervisor function. When a SWI is executed, ARM Processor performs the following:

1. Saves the address of the SWI instruction plus 4 in R14_svc; saves CPSR in SPSR_svc
2. Forces M[4:0]=10011 (Supervisor mode) and sets the I bit in the CPSR
3. Forces the PC to fetch the next instruction from address 0x08

Returning from a SWI

To return from a SWI, use MOVS PC,R14_svc. This will restore the PC and CPSR and return to the instruction following the SWI.

4.5.5 Undefined instruction trap

When the ARM Processor comes across an instruction which it cannot handle, it takes the undefined instruction trap. This includes all coprocessor instructions, except MCR and MRC operations which access the internal control coprocessor.

The trap may be used for software emulation of a coprocessor in a system which does not have the coprocessor hardware, or for general purpose instruction set extension by software emulation.

When ARM Processor takes the undefined instruction trap it performs the following:

1. Saves the address of the Undefined or coprocessor instruction plus 4 in R14_und; saves CPSR in SPSR_und
2. Forces M[4:0]=11011 (Undefined mode) and sets the I bit in the CPSR
3. Forces the PC to fetch the next instruction from address 0x04

Returning from an undefined instruction trap

To return from this trap after emulating the failed instruction, use MOVS PC,R14_und. This will restore the CPSR and return to the instruction following the undefined instruction.

4.5.6 Vector summary

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	-- reserved --	--
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Table 4-2: Vector summary

These are byte addresses, and will normally contain a branch instruction pointing to the relevant routine.

The FIQ routine might reside at 0x1C onwards, and thereby avoid the need for (and execution time of) a branch instruction.

4.5.7 Exception priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they will be handled:

1. Reset (highest priority)
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort
6. Undefined Instruction, software interrupt (lowest priority)

Note: Not all exceptions can occur at once. Undefined instruction and software interrupt are mutually exclusive since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (i.e. the F flag in the CPSR is clear), ARM Processor will enter the data abort handler and then immediately proceed to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection; the time for this exception entry should be added to worst-case FIQ latency calculations.

4.5.8 Interrupt latencies

Calculating the worst-case interrupt latency for the ARM Processor is quite complex due to the cache, MMU and write buffer and is dependant on the configuration of the whole system.

4.5.9 Reset

When the ARM7500 is reset, ARM Processor abandons the executing instruction and then performs idle cycles from incrementing word addresses.

When the ARM7500 comes out of reset, ARM Processor does the following:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and CPSR is not defined.
2. Forces M[4:0]=10011 (Supervisor mode); sets the I and F bits in the CPSR.
3. Forces the PC to fetch the next instruction from address 0x00.

End of reset sequence

At the end of the reset sequence:

- the MMU is disabled and the TLB is flushed, so forces “flat” translation (i.e. the physical address is the virtual address, and there is no permission checking)
- alignment faults are also disabled
- the cache is disabled and flushed

- the write buffer is disabled and flushed
- the ARM7 CPU core is put into 26 bit data and address mode, with early abort timing and Little Endian mode

4.6 Configuration control registers

The operation and configuration of ARM Processor is controlled both directly via coprocessor instructions and indirectly via the Memory Management Page tables. The coprocessor instructions manipulate a number of on-chip registers which control the configuration of the Cache, write buffer, MMU and a number of other configuration options.

To ensure backwards compatibility of future CPUs, all reserved or unused bits in registers and coprocessor instructions should be programmed to '0'. Invalid registers must not be read/written. The following bits must be programmed to '0':

- Register 1 bits[31:11]
- Register 2 bits[13:0]
- Register 5 bits[31:0]
- Register 6 bits[11:0]
- Register 7 bits[31:0]

Note: *The areas marked “Reserved” in the register and translation diagrams should be programmed 0 for future compatibility.*

4.6.1 Internal coprocessor instructions

The on-chip registers may be read using MRC instructions and written using MCR instructions. These operations are only allowed in non-user modes and the undefined instruction trap will be taken if accesses are attempted in user mode. ➲*5.14 Coprocessor register transfers (MRC, MCR)* on page 5-48

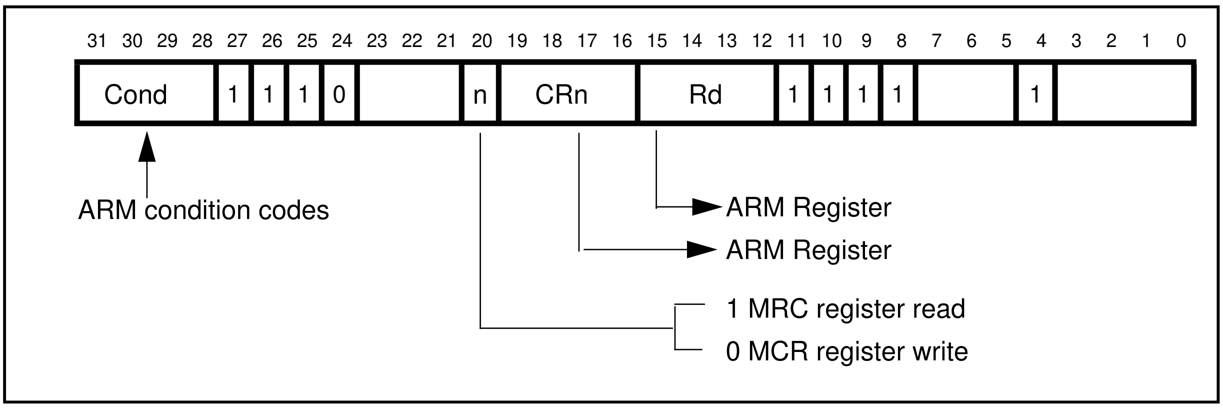


Figure 4-5: Format of Internal Coprocessor Instructions MRC and MCR

4.6.2 Registers

The ARM Processor contains registers which control the cache and MMU operation. These registers are accessed using CPRT instructions to Coprocessor #15 with the processor in a privileged mode. Only some of registers 0-7 are valid: an access to an invalid register will cause neither the access nor an undefined instruction trap, and therefore should never be carried out; an access to any of the registers 8-15 will cause the undefined instruction trap to be taken.

Register	Register reads	Register writes
0	Reserved	Reserved
1	Reserved	Control
2	Reserved	Translation Table Base
3	Reserved	Domain Access Control
4	Reserved	Reserved
5	Fault Status	Flush TLB
6	Fault Address	Purge TLB
7	Reserved	Flush IDC
8-15	Reserved	Reserved

Table 4-3: Cache and MMU control registers

Register 1: **Control**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	R	S	B	1	D	P	W	C	A	M

Register 1 is write only and contains control bits. All bits in this register are forced LOW by reset.

M Bit 0 Enable/disable

0 on-chip Memory Management Unit turned off
1 on-chip Memory Management Unit turned on.

A Bit 1 Address Fault Enable/Disable

0 alignment fault disabled
1 alignment fault enabled

C Bit 2 Cache Enable/Disable

0 Instruction / data cache turned off
1 Instruction / data cache turned on

W Bit 3 Write buffer Enable/Disable

0 Write buffer turned off
1 Write buffer turned on

P Bit 4 ARM 32/26 Bit Program Space

0 26-bit Program Space selected
1 32-bit Program Space selected

D Bit 5 ARM 32/26 Bit Data Space

0 26 bit Data Space selected
1 32 bit Data Space selected

B Bit 7 Big/Little Endian

0 Little-endian operation
1 Big-endian operation

S Bit 8 System bit, which controls the ARM Processor permission system.

R Bit 9 ROM bit, whicht controls the ARM Processor permission system

Register 2: **Translation Table Base**

31–14	13–0
Translation Table Base	

Register 2 is a write-only register which holds the base of the currently active Level One page table.

Register 3: **Domain Access Control**

31–30	29–28	27–26	25–24	23–22	21–20	19–18	17–16	15–14	13–12	11–10	9–8	7–6	5–4	3–2	1–0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Register 3 is a write-only register which holds the current access control for domains 0 to 15. See *7.10 Domain access control* on page 7-13 for the access permission definitions and other details.

Register 4: **Reserved**

Register 4 is Reserved. Accessing this register has no effect, but should never be attempted.

Register 5: **Fault Status/Translation Lookaside Buffer Flush**

31–12	11	10	9	8	7–4	3–0
	0	0	0	0	Domain	Status

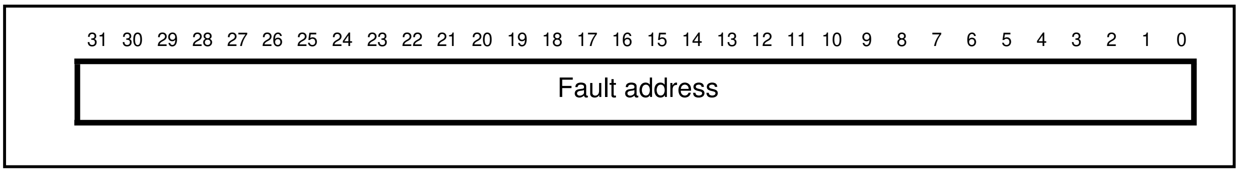
Read: Fault Status

Reading register 5 returns the status of the last data fault. It is not updated for a prefetch fault. See *Chapter 7, ARM Processor MMU* for more details. Note that only the bottom 12 bits are returned. The upper 20 bits will be the last value on the internal data bus, and therefore will have no meaning. Bits 11:8 are always returned as zero.

Write: Translation Lookaside Buffer Flush

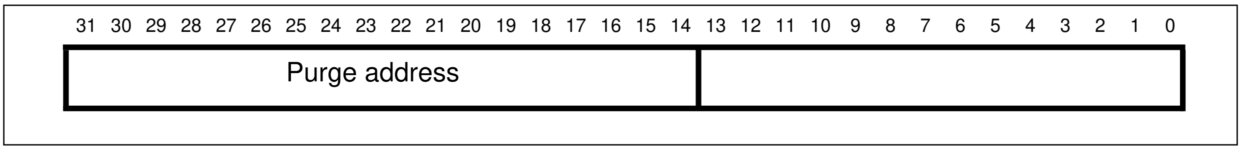
Writing Register 5 flushes the TLB. (The data written is discarded).

Register 6: **Fault Address/ TLB Purge**



Read: Fault Address

Reading register 6 returns the virtual address of the last data fault.



Write: TLB Purge

Writing Register 6 purges the TLB; the data is treated as an address and the TLB is searched for a corresponding page table descriptor. If a match is found, the corresponding entry is marked as invalid. This allows the page table descriptors in main memory to be updated and invalid entries in the on-chip TLB to be purged without requiring the entire TLB to be flushed.

Register 7: **IDC Flush**

Register 7 is a write-only register. The data written to this register is discarded and the IDC is flushed.

Registers 8 -15: **Reserved**

Accessing any of these registers will cause the undefined instruction trap to be taken.

5 ARM Processor Instruction Set

This chapter describes the ARM Processor instruction set.

5.1 Instruction set summary

A summary of the ARM Processor instruction set is shown in *Figure 5-1: Instruction set summary*.

Note: *Some instruction codes are not defined but do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions shall not be used, as their action may change in future ARM implementations.*

Bit positions: 31 28 27 26 25 24 23 22 21 20 19 16 15 12 11 8 7 5 4 3 0

Cond	0 0	I	Opcode	S	Rn	Rd	Operand 2				**Data Processing PSR Transfer**
Cond	0 0 0 0 0 0	A	S	Rd	Rn	Rs	1 0 0 1	Rm			**Multiply**
Cond	0 0 0 1 0	B	0 0	Rn	Rd	0 0 0 0	1 0 0 1	Rm			**Single Data Swap**
Cond	0 1	I	P	U	B	W	L	Rn	Rd	offset	**Single Data Transfer**
Cond	0 1 1	XXXXXXXXXXXXXXXXXXXX	1	XXXX							**Undefined**
Cond	1 0 0	P	U	S	W	L	Rn	Register List			**Block Data Transfer**
Cond	1 0 1	L	offset								**Branch**
Cond	1 1 0	P	U	N	W	L	Rn	CRd	CP#	offset	**Coproc Data Transfer**
Cond	1 1 1 0	CP Opc	CRn	CRd	CP#	CP	0	CRm			**Coproc Data Operation**
Cond	1 1 1 0	CP Opc	L	CRn	Rd	CP#	CP	1	CRm		**Coproc Register Transfer**
Cond	1 1 1 1	ignored by processor									**Software Interrupt**

Figure 5-1: Instruction set summary

5.2 The condition field

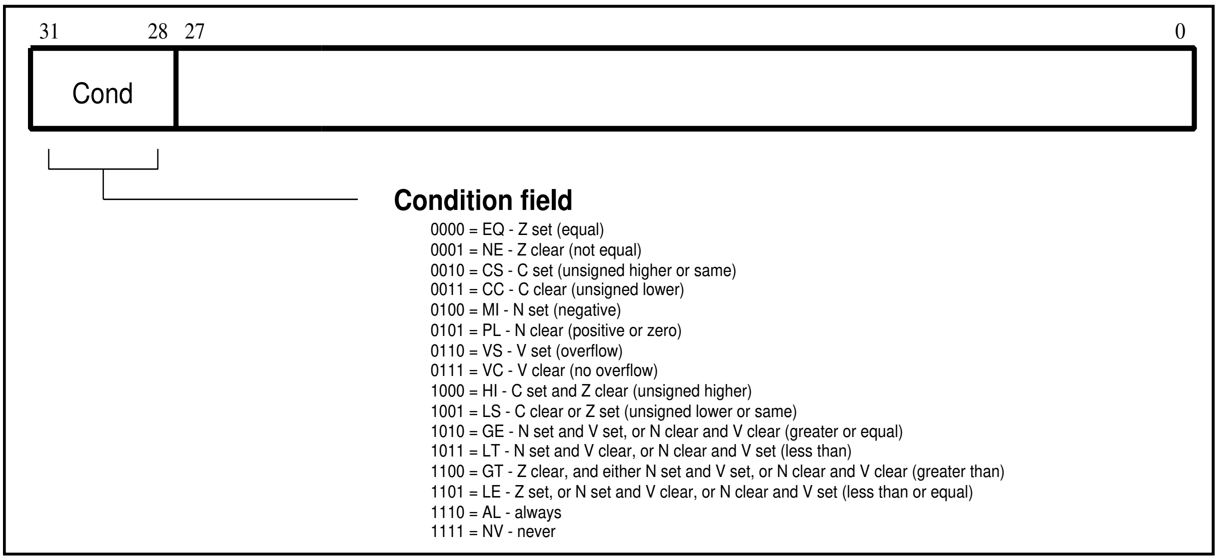


Figure 5-2: Condition codes

All ARM Processor instructions are conditionally executed, which means that their execution may or may not take place depending on the values of the N, Z, C and V flags in the CPSR.

The condition codes have meanings as detailed in *Figure 5-2: Condition codes*, for instance code 0000 (EQual) executes the instruction only if the Z flag is set. This would correspond to the case where a compare (CMP) instruction had found the two operands to be equal. If the two operands were different, the compare instruction would have cleared the Z flag and the instruction is not executed.

Note: *If the* always *(AL - 1110) condition is specified, the instruction will be executed irrespective of the flags. The never (NV - 1111) class of condition codes must not be used as they will be redefined in future variants of the ARM architecture. If a NOP is required it is suggested that MOV R0,R0 be used. The assembler treats the absence of a condition code as though* always *had been specified.*

5.3 Branch and branch with link (B, BL)

These instructions are only executed if the condition is true. The instruction encoding is shown in *Figure 5-3: Branch instructions*.

Branch instructions contain a signed 2's complement 24-bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

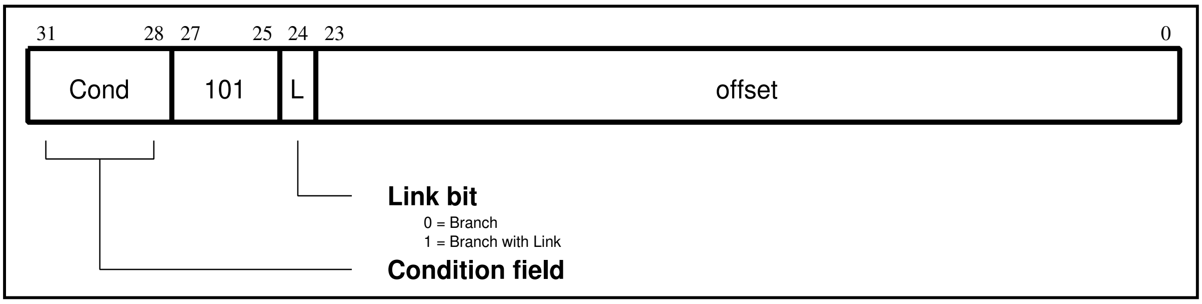


Figure 5-3: Branch instructions

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a branch with link type operation is required.

5.3.1 The link bit

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or use LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

5.3.2 Instruction cycle times

Branch and Branch with Link instructions take 3 instruction fetches. For more information see *5.16.7 Instruction speed summary* on page 5-55.

5.3.3 Assembler syntax

```
B{L}{cond} <expression>
```

Items in {} are optional. Items in <> must be present.

{L}	requests the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.
{cond}	is a two-char mnemonic as shown in *Figure 5-2: Condition codes* on page 5-3 (EQ, NE, VS etc). If absent then AL (ALways) will be used.
<expression>	is the destination. The assembler calculates the offset.

5.3.4 Examples

```
here   BAL   here     ;assembles to 0xEAFFFFFE (note effect of PC
                      ;offset)
       B     there    ;ALways condition used as default
       CMP   R1,#0    ;compare R1 with zero and branch to fred if R1
       BEQ   fred     ;was zero otherwise continue to next instruction
       BL    sub+ROM  ;call subroutine at computed address
       ADDS  R1,#1    ;add 1 to register 1, setting CPSR flags on the
       BLCC  sub      ;result then call subroutine if the C flag is
                      ;clear, which will be the case unless R1 held
                      ;0xFFFFFFFF
```

5.4 Data processing

The instruction is only executed if the condition is true, defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-4: Data processing instructions* on page 5-7.

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands.

First operand — is always a register (Rn).

Second operand — may be a shifted register (Rm) or a rotated 8-bit immediate value (Imm) according to the value of the **I** bit in the instruction.

The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S-bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set.

The instructions and their effects are listed in *Table 5-1: ARM data processing instructions* on page 5-8.

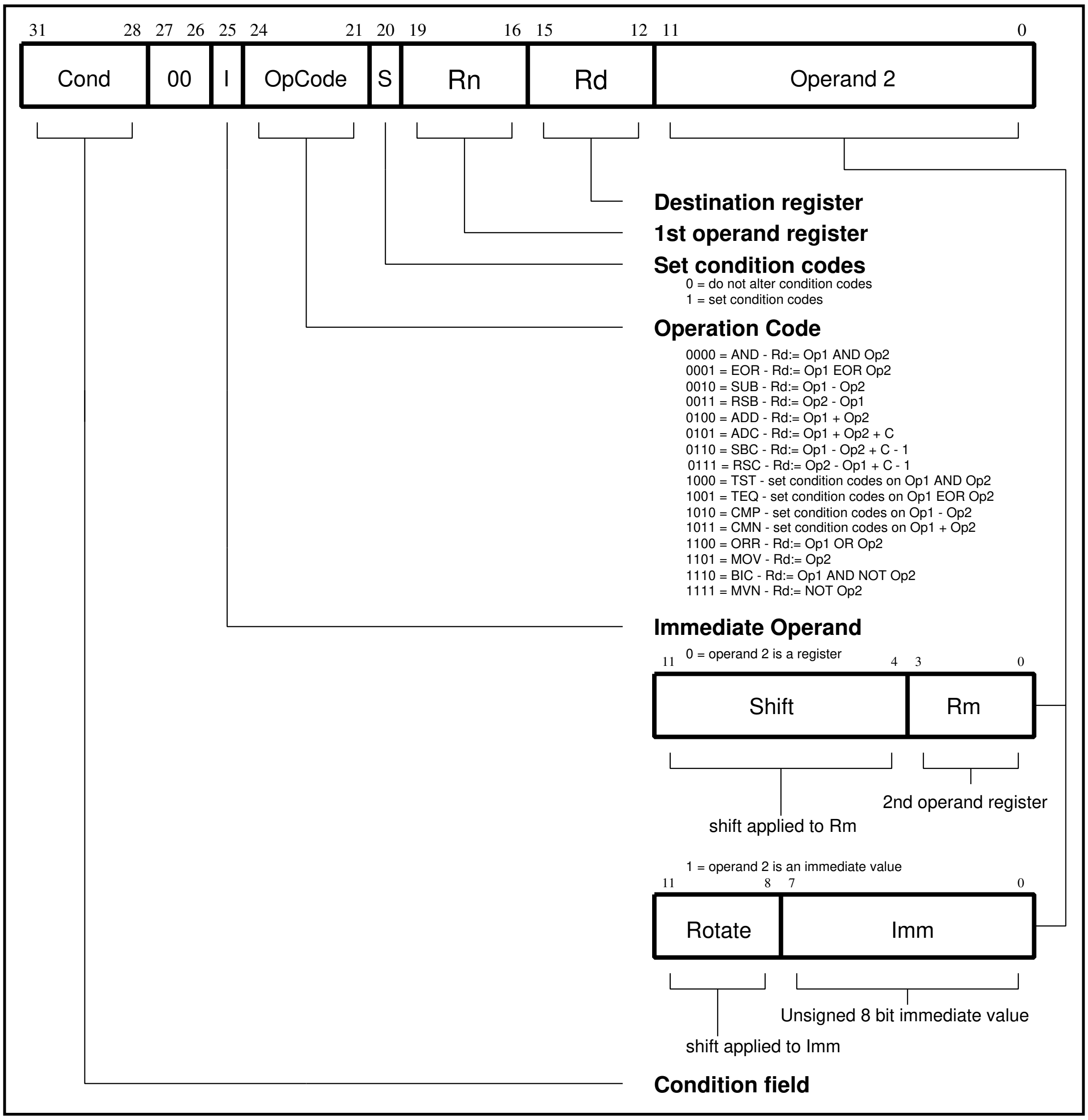


Figure 5-4: Data processing instructions

5.4.1 CPSR flags

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result.

If the S bit is set (and Rd is not R15):

- the V flag in the CPSR will be unaffected
- the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0)
- the Z flag will be set if and only if the result is all zeros
- the N flag will be set to the logical value of bit 31 of the result.

Assembler mnemonic	OpCode	Action
AND	0000	operand1 AND operand2
EOR	0001	operand1 EOR operand2
SUB	0010	operand1 - operand2
RSB	0011	operand2 - operand1
ADD	0100	operand1 + operand2
ADC	0101	operand1 + operand2 + carry
SBC	0110	operand1 - operand2 + carry - 1
RSC	0111	operand2 - operand1 + carry - 1
TST	1000	as AND, but result is not written
TEQ	1001	as EOR, but result is not written
CMP	1010	as SUB, but result is not written
CMN	1011	as ADD, but result is not written
ORR	1100	operand1 OR operand2
MOV	1101	operand2 (operand1 is ignored)
BIC	1110	operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

Table 5-1: ARM data processing instructions

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32-bit integer (either unsigned or 2's complement signed, the two are equivalent).

If the S bit is set (and Rd is not R15):

- the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2's complement signed
- the C flag will be set to the carry out of bit 31 of the ALU
- the Z flag will be set if and only if the result was zero
- the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

5.4.2 Shifts

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in *Figure 5-5: ARM shift operations*.

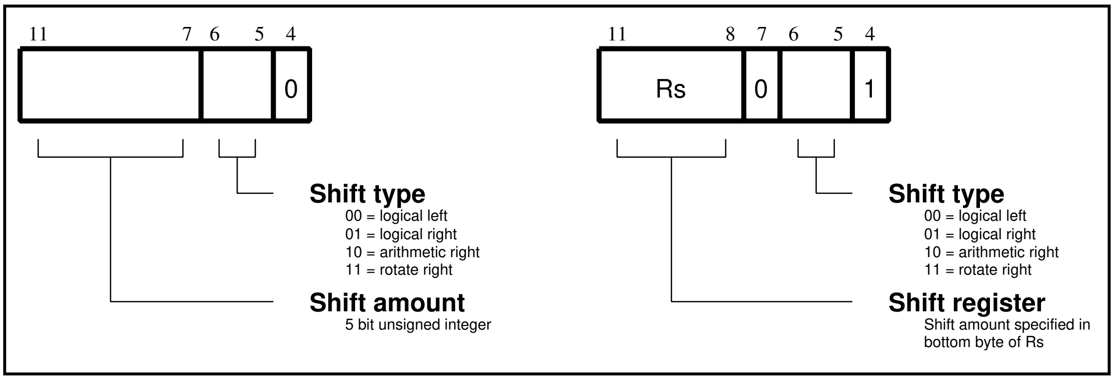


Figure 5-5: ARM shift operations

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5 bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in *Figure 5-6: Logical shift left* on page 5-10.

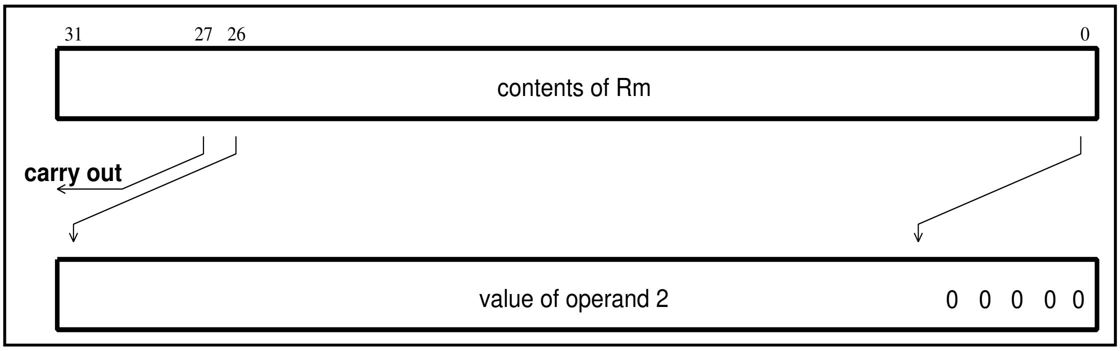


Figure 5-6: Logical shift left

Note: *LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand.*

Logical shift right

A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in *Figure 5-7: Logical shift right*.

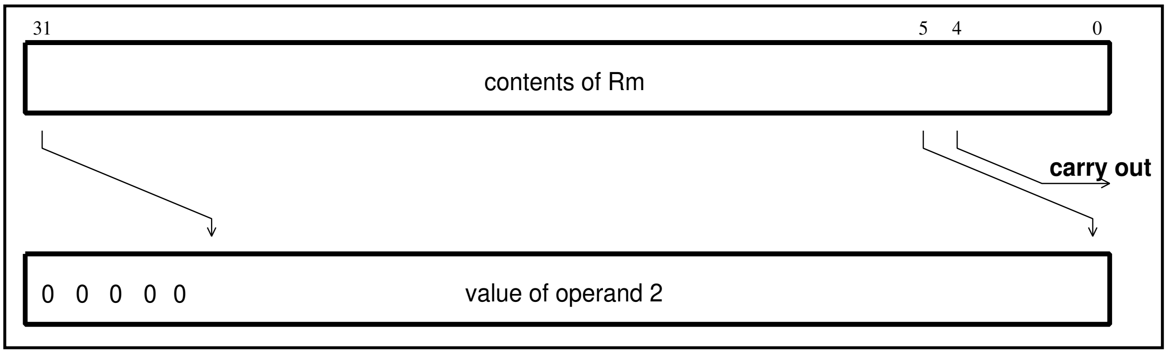


Figure 5-7: Logical shift right

The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

Arithmetic shift

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2's complement notation. For example, ASR #5 is shown in *Figure 5-8: Arithmetic shift right*.

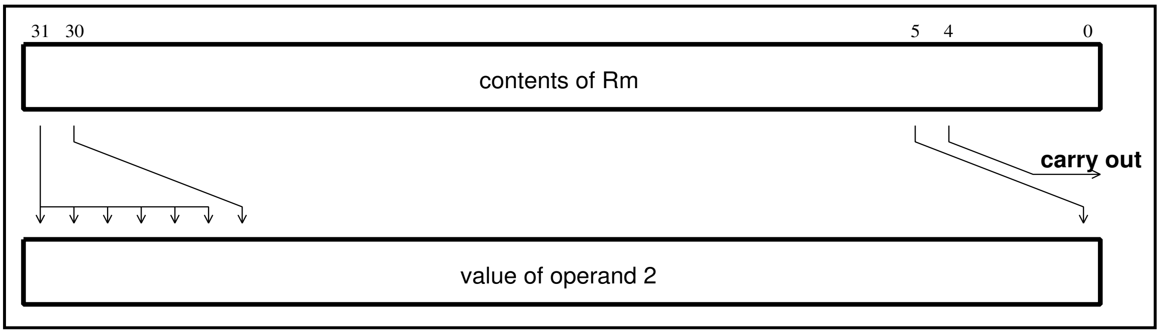


Figure 5-8: Arithmetic shift right

The form of the shift field which might be expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right

Rotate right (ROR) operations reuse the bits which 'overshoot' in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in *Figure 5-9: Rotate right* on page 5-11.

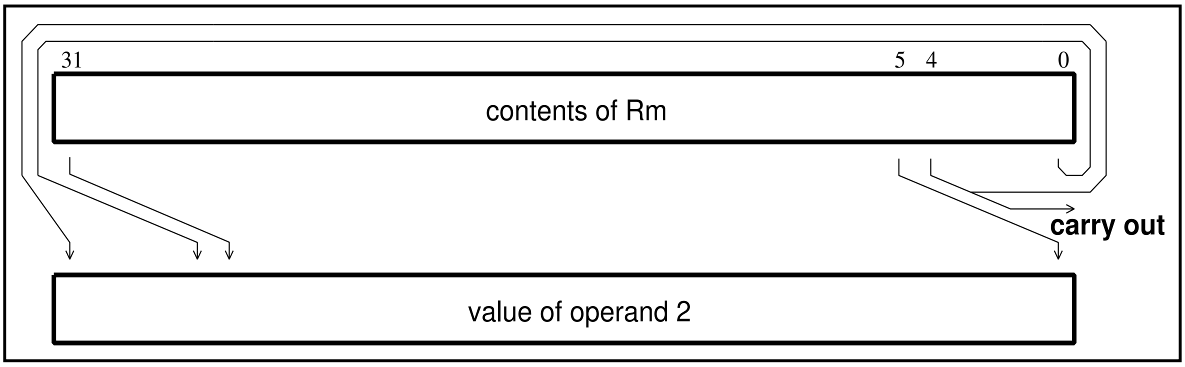


Figure 5-9: Rotate right

The form of the shift field which might be expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33 bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in *Figure 5-10: Rotate right extended*.

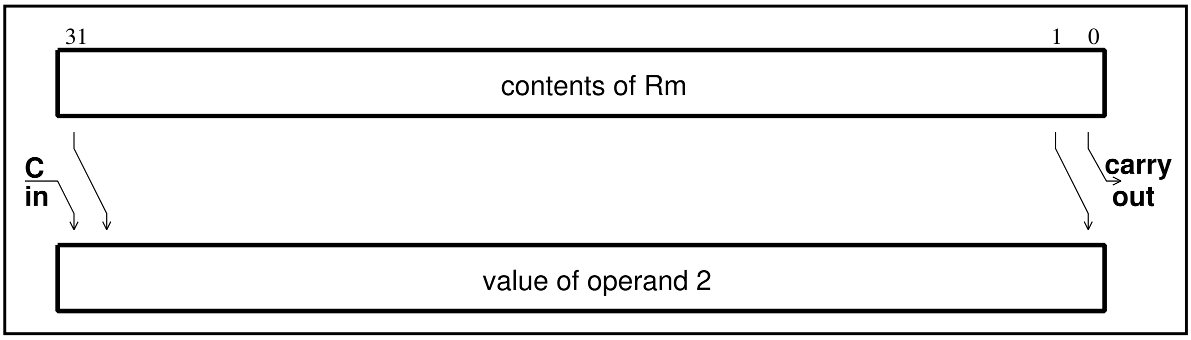


Figure 5-10: Rotate right extended

Register specified shift amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

Byte	Description
0	Unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output
1 - 31	The shifted result will exactly match that of an instruction specified shift with the same value and shift operation
32 or more	The result will be a logical extension of the shift described above: 1 LSL by 32 has result zero, carry out equal to bit 0 of Rm. 2 LSL by more than 32 has result zero, carry out zero. 3 LSR by 32 has result zero, carry out equal to bit 31 of Rm. 4 LSR by more than 32 has result zero, carry out zero. 5 ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm. 6 ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm. 7 ROR by *n* where *n* is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from *n* until the amount is in the range 1 to 32 and see above.

Table 5-2: Register specified shift amount

Note: The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

5.4.3 Immediate operand rotates

The immediate operand rotate field is a 4 bit unsigned integer which specifies a shift operation on the 8 bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

5.4.4 Writing to R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR.

Note: This form of instruction must not be used in User mode.

5.4.5 Using R15 as an operand

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

5.4.6 TEQ, TST, CMP & CMN opcodes

These instructions do not write the result of their operation but do set flags in the CPSR. An assembler shall always set the S flag for these instructions even if it is not specified in the mnemonic.

The TEQP form of the instruction used in earlier processors shall not be used in the 32-bit modes, the PSR transfer operations should be used instead. If used in these modes, its effect is to move SPSR_<mode> to CPSR if the processor is in a privileged mode and to do nothing if in User mode.

5.4.7 Instruction cycle times

Data Processing instructions vary in the number of incremental cycles taken as follows:

Instruction	Cycles
Normal Data Processing	1instruction fetch
Data Processing with register specified shift	1 instruction fetch + 1 internal cycle
Data Processing with PC written	3 instruction fetches
Data Processing with register specified shift and PC written	3 instruction fetches and 1 internal cycle

Figure 5-11: Instructin cycle times

See *5.16.7 Instruction speed summary* on page 5-55 for more information.

5.4.8 Assembler syntax

1 MOV,MVN - single operand instructions

```
<opcode>{cond}{S} Rd,<Op2>
```

2 CMP,CMN,TEQ,TST - instructions which do not produce a result.

```
<opcode>{cond} Rn,<Op2>
```

3 AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC

```
<opcode>{cond}{S} Rd,Rn,<Op2>
```

where:

<Op2> is `Rm{,<shift>}` or,`<#expression>`

{cond}	two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
{S}	set condition codes if S present (implied for CMP, CMN, TEQ, TST).
Rd, Rn and Rm	are expressions evaluating to a register number.
<#expression>	if used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.
<shift>	is <shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).
<shiftname>s	are: ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL; they assemble to the same code.)

5.4.9 Example

```
ADDEQ R2,R4,R5        ;if the Z flag is set make R2:=R4+R
TEQS  R4,#3           ;test R4 for equality with 3
                      ;(the S is in fact redundant as the
                      ;assembler inserts it automatically)
SUB   R4,R5,R7,LSR R2;
                      ;logical right shift R7 by the number in
                      ;the bottom byte of R2, subtract result
                      ;from R5, and put the answer into R4
MOV   PC,R14          ;return from subroutine
MOVS  PC,R14          ;return from exception and restore CPSR
                      ;from SPSR_mode
```

5.5 PSR transfer (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined in ➲*5.2 The condition field* on page 5-3.

The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in ➲*Figure 5-12: PSR transfer* on page 5-16.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register.

The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register. The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32-bit immediate value are written to the top four bits of the relevant PSR.

5.5.1 Operand restrictions

In User mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.

The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.

Note: R15 must not be specified as the source or destination register.

A further restriction is that you must not attempt to access an SPSR in User mode, since no such register exists.

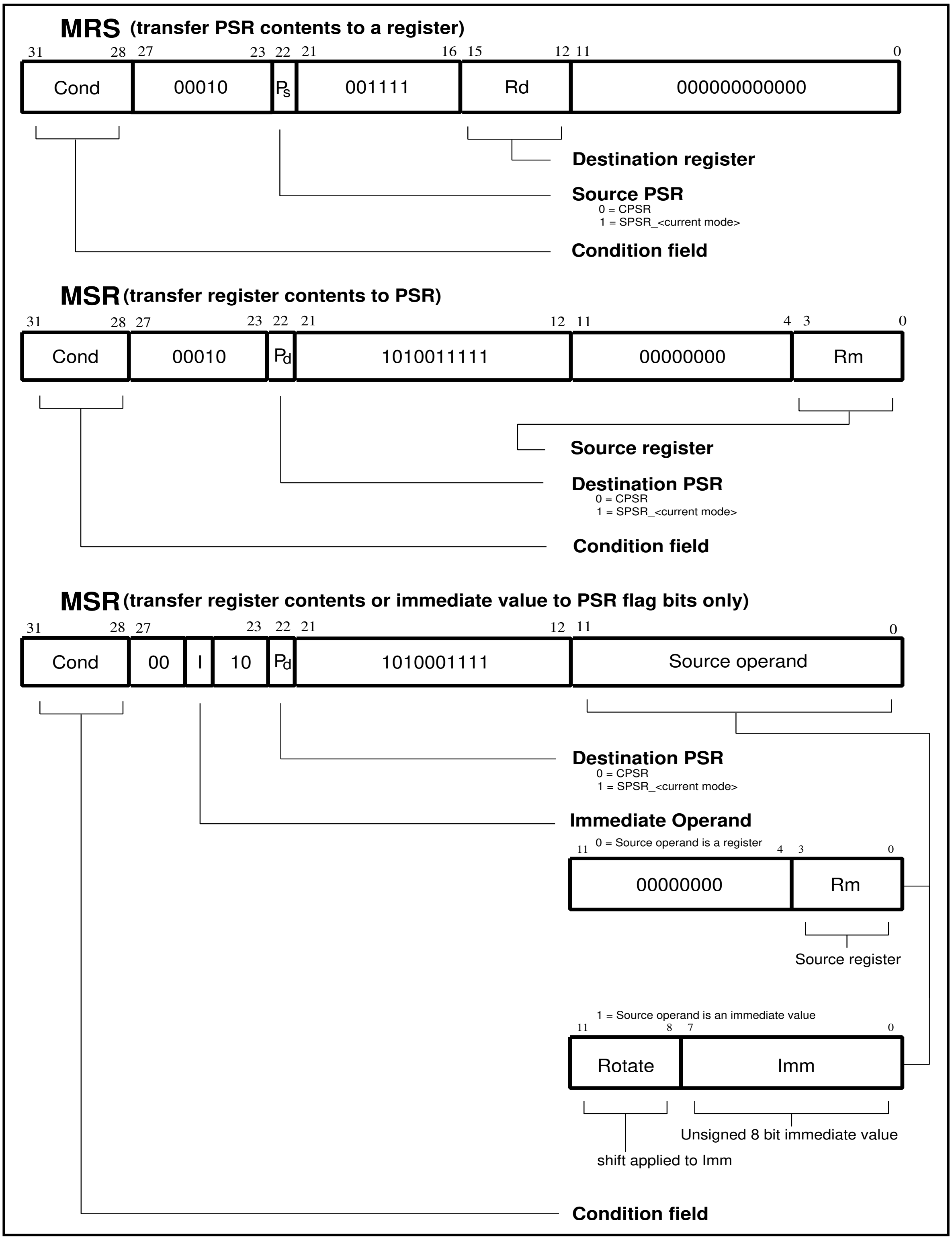


Figure 5-12: PSR transfer

5.5.2 Reserved bits

Only eleven bits of the PSR are defined in ARM Processor (N,Z,C,V,I,F & M[4:0]); the remaining bits (= PSR[27:8,5]) are reserved for use in future versions of the processor.

Compatibility

To ensure the maximum compatibility between ARM Processor programs and future processors, the following rules should be observed:

1. The reserved bit smust be preserved when changing the value in a PSR.
2. Programs must not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

For example, the following sequence performs a mode change:

```
MRS   R0,CPSR           ;take a copy of the CPSR
BIC   R0,R0,#0x1F       ;clear the mode bits
ORR   R0,R0,#new_mode   ;select new mode
MSR   CPSR,R0           ;write back the modified CPSR
```

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. e.g. The following instruction sets the N,Z,C & V flags:

```
MSR   CPSR_flg,#0xF0000000
                        ;set all the flags regardless of
                        ;their previous state (does not
                        ;affect any control bits)
```

Note: Do not attempt to write an 8 bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

5.5.3 Instruction cycle times

PSR Transfers take 1 instruction fetch. For more information see ➲*5.16.7 Instruction speed summary* on page 5-55.

Pre-release Draft - Unrestricted

5.5.4 Assembler syntax

1 MRS - transfer PSR contents to a register

```
MRS{cond} Rd,<psr>
```

2 MSR - transfer register contents to PSR

```
MSR{cond} <psr>,Rm
```

3 MSR - transfer register contents to PSR flag bits only

```
MSR{cond} <psrf>,Rm
```

The most significant four bits of the register contents are written to the N,Z,C & V flags respectively.

4 MSR - transfer immediate value to PSR flag bits only

```
MSR{cond} <psrf>,<#expression>
```

The expression should symbolise a 32-bit value of which the most significant four bits are written to the N,Z,C & V flags respectively.

where:

{cond}	two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
Rd and Rm	are expressions evaluating to a register number other than R15
<psr>	is CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)
<psrf>	is CPSR_flg or SPSR_flg
<#expression>	where used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

5.5.5 Examples

In User mode the instructions behave as follows:

```
MSR   CPSR_all,Rm      ;CPSR[31:28] <- Rm[31:28]
MSR   CPSR_flg,Rm      ;CPSR[31:28] <- Rm[31:28]
MSR   CPSR_flg,#0xA0000000;
                       ;CPSR[31:28] <- 0xA
                       ;(i.e. set N,C; clear Z,V)
MRS   Rd,CPSR          ;Rd[31:0] <- CPSR[31:0]
```

In privileged modes the instructions behave as follows:

```
MSR   CPSR_all,Rm      ;CPSR[31:0]  <- Rm[31:0]
MSR   CPSR_flg,Rm      ;CPSR[31:28] <- Rm[31:28]
MSR   CPSR_flg,#0x50000000;
                       ;CPSR[31:28] <- 0x5
                       ;(i.e. set Z,V; clear N,C)
MRS   Rd,CPSR          ;Rd[31:0] <- CPSR[31:0]
MSR   SPSR_all,Rm      ;SPSR_<mode>[31:0]  <- Rm[31:0]
MSR   SPSR_flg,Rm      ;SPSR_<mode>[31:28] <- Rm[31:28]
MSR   SPSR_flg,#0xC0000000;
                       ;SPSR_<mode>[31:28] <- 0xC
                       ;(i.e. set N,Z; clear C,V)
MRS   Rd,SPSR          ;Rd[31:0] <- SPSR_<mode>[31:0]
```

5.6 Multiply and multiply-accumulate (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in ➲*Figure 5-13: Multiply instructions*.

The multiply and multiply-accumulate instructions use a 2-bit Booth's algorithm to perform integer multiplication. They give the least significant 32-bits of the product of two 32-bit operands, and may be used to synthesize higher-precision multiplications.

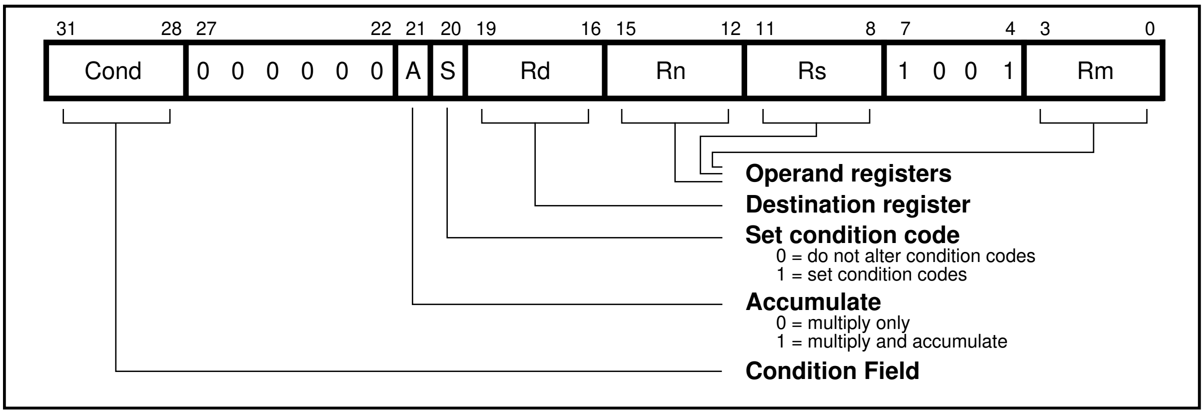


Figure 5-13: Multiply instructions

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set.

The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances.

The results of a signed multiply and of an unsigned multiply of 32-bit operands differ only in the upper 32 bits; the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

Example

For example consider the multiplication of the operands:

```
Operand A      Operand B      Result
0xFFFFFFF6     0x00000014     0xFFFFFF38
```

If the operands are interpreted as signed, operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFF38

If the operands are interpreted as unsigned, operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFF38.

5.6.1 Operand restrictions

Due to the way multiplication was implemented in other ARM processors, certain combinations of operand registers should be avoided. The ARM Processor's advanced multiplier can handle all operand combinations but by observing these restrictions code written for the ARM Processor will be more compatible with other ARM processors. (The assembler will issue a warning if these restrictions are overlooked.)

Note: The destination register Rd must not be the same as the operand register Rm. R15 shall not be used as an operand or as the destination register. All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.

5.6.2 CPSR flags

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

5.6.3 Instruction cycle times

The Multiply instructions take 1 instruction fetch and m internal cycles, as shown in *Table 5-3: Instruction cycle times*. For more information see *5.16.7 Instruction speed summary* on page 5-55..

Multiplication by	**Takes**
any number between *2^(2m-3)* and *2^(2m-1)-1*	1S+mI cycles for *1<m>16*.
Multiplication by 0 or 1	1S+1I cycles
any number greater than or equal to 2^(29)	1S+16I cycles.

Table 5-3: Instruction cycle times

m is the number of cycles required by the multiply algorithm, which is determined by the contents of Rs

The maximum time for any multiply is thus 1S+16I cycles.

5.6.4 Assembler syntax

```
MUL{cond}{S} Rd,Rm,Rs
MLA{cond}{S} Rd,Rm,Rs,Rn
```

where:

{cond}	two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
{S}	set condition codes if S present
Rd, Rm, Rs, Rn	are expressions evaluating to a register number other than R15.

5.6.5 Examples

```
MUL         R1,R2,R3         ;R1:=R2*R3
MLAEQS      R1,R2,R3,R4      ;conditionally
                             ;R1:=R2*R3+R4,
                             ;setting condition codes
```

5.7 Single data transfer (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-14: Single data transfer instructions*.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if 'auto-indexing' is required.

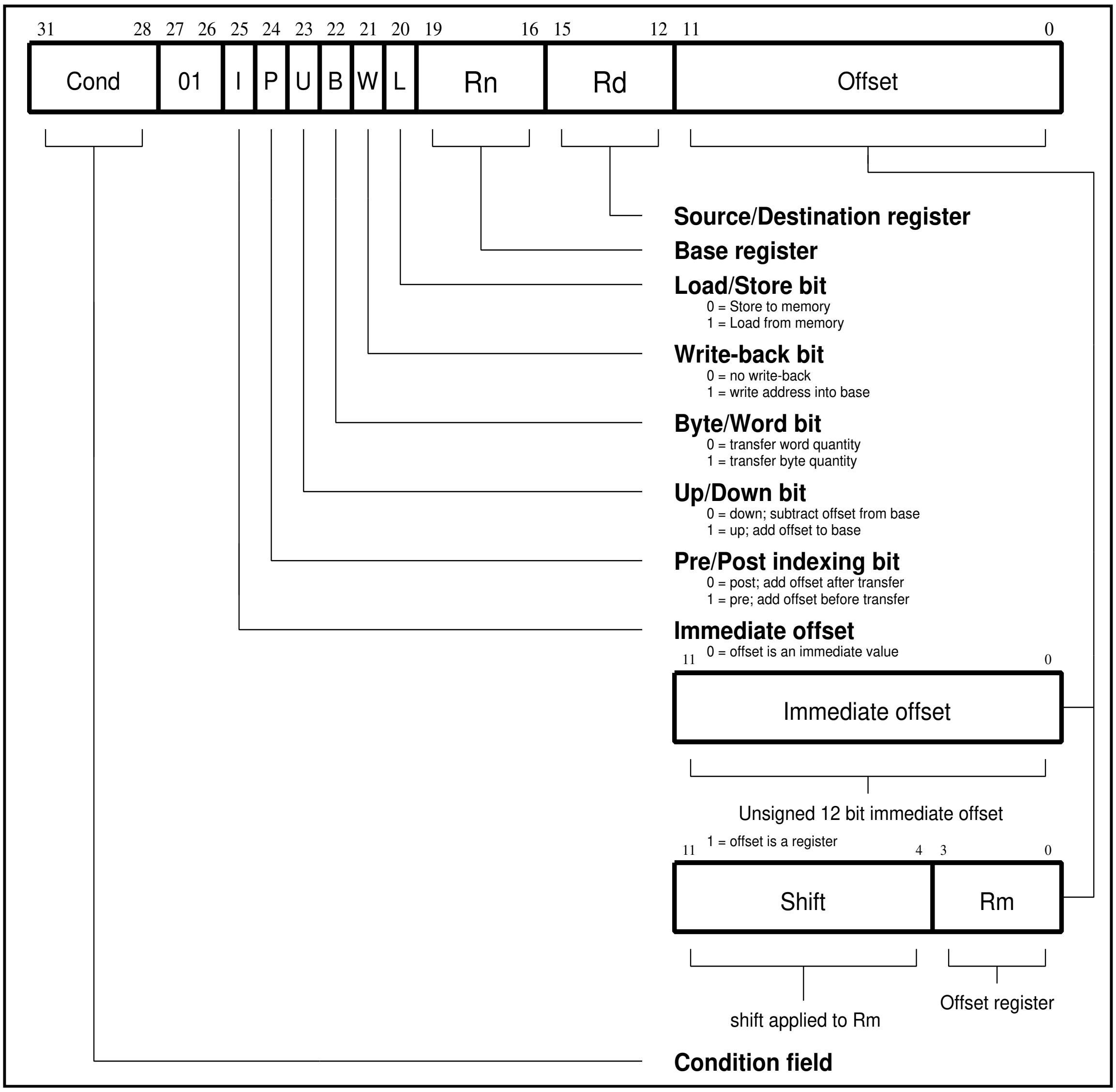


Figure 5-14: Single data transfer instructions

5.7.1 Offsets and auto-indexing

The offset from the base may be either a 12-bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes.
The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0).

Post-indexed addressing

In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

5.7.2 Shifted register offset

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See ❍*5.4.2 Shifts* on page 5-9.

5.7.3 Bytes and words

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM Processor register and memory. The following text assumes that the ARM7500 is operating with 32-bit wide memory. If it is operating with 16-bit wide memory, the positions of bytes on the *external* data bus will be different, although, on the ARM7500 internal data bus the positions will be as described here.

The action of LDR(B) and STR(B) instructions is influenced by the 3 instruction fetches. For more information see ❍*5.16.7 Instruction speed summary* on page 5-55. The two possible configurations are described below.

Little Endian Configuration

Byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. See *Figure 4-1: Little Endian addresses of bytes within words* on page 4-2.

Byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0.

Word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits. This is illustrated in *Figure 5-15: Little Endian offset addressing* on page 5-25.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

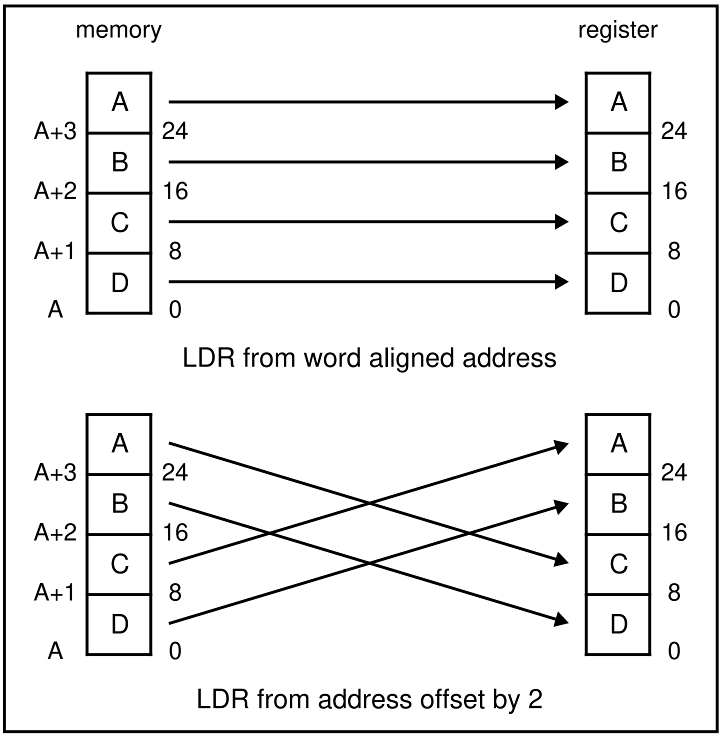


Figure 5-15: Little Endian offset addressing

Big Endian Configuration

Byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see *Figure 4-2: Big Endian addresses of bytes within words* on page 4-3.

Byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0.

Word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

5.7.4 Use of R15

Do not specify write-back if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

5.7.5 Restriction on the use of base register

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

For example:

```
LDR   R0,[R1],R1
      <LDR|STR> Rd, [Rn],{+/-}Rn{,<shift>}
```

Therefore a post-indexed LDR|STR where Rm is the same register as Rn shall not be used.

5.7.6 Data aborts

A transfer to or from a legal address may cause the MMU to generate an abort. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

5.7.7 Instruction cycle times

Instruction	Cycles
Normal LDR instruction	1 instruction fetch, 1 data read and 1 internal cycle
LDR PC	3 instruction fetches, 1 data read and 1 internal cycle.
STR instruction	1 instruction fetch and 1 data write incremental cycles.

Table 5-4: Instruction cycle times

For more information see ➲*5.16.7 Instruction speed summary* on page 5-55.

5.7.8 Assembler syntax

`<LDR|STR>{cond}{B}{T} Rd,<Address>`

LDR load from memory into a register

STR store from a register into memory

{cond} two-character condition mnemonic, see ➲*Figure 5-2: Condition codes* on page 5-3

{B} if B is present then byte transfer, otherwise word transfer

{T} if T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.

Rd is an expression evaluating to a valid register number.

<Address> can be:

1 An expression which generates an address:

`<expression>`

The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2 A pre-indexed addressing specification:

```
[Rn]                      offset of zero
[Rn,<#expression>]{!}     offset of <expression> bytes
[Rn,{+/-}Rm{,<shift>}]{!} offset of +/- contents of
                          index register, shifted by
                          <shift>
```

3 A post-indexed addressing specification:

`[Rn],<#expression>`	`offset of <expression> bytes`
`[Rn],{+/-}Rm{,<shift>}`	`offset of +/- contents of index register, shifted as by <shift>.`

Rn and Rm	are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7500 pipelining. In this case base write-back shall not be specified.
<shift>	is a general shift operation (see section on data processing instructions) but note that the shift amount may not be specified by a register.
{!}	writes back the base register (set the W bit) if ! is present.

5.7.9 Examples

```
STR   R1,[R2,R4]!     ;store R1 at R2+R4 (both of which are
                      ;registers) and write back address to R2
STR   R1,[R2],R4      ;store R1 at R2 and write back
                      ;R2+R4 to R2
LDR   R1,[R2,#16]     ;load R1 from contents of R2+16
                      ; Don't write back
LDR   R1,[R2,R3,LSL#2]
                      ;load R1 from contents of R2+R3*4
LDREQB
      R1,[R6,#5]      ;conditionally load byte at R6+5 into
                      ; R1 bits 0 to 7, filling bits 8 to 31
                      ; with zeros
STR   R1,PLACE        ;generate PC relative offset to address
      •               ;PLACE
      •
PLACE
```

5.8 Block data transfer (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-16: Block data transfer instructions*.

Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

5.8.1 The register list

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16 bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.

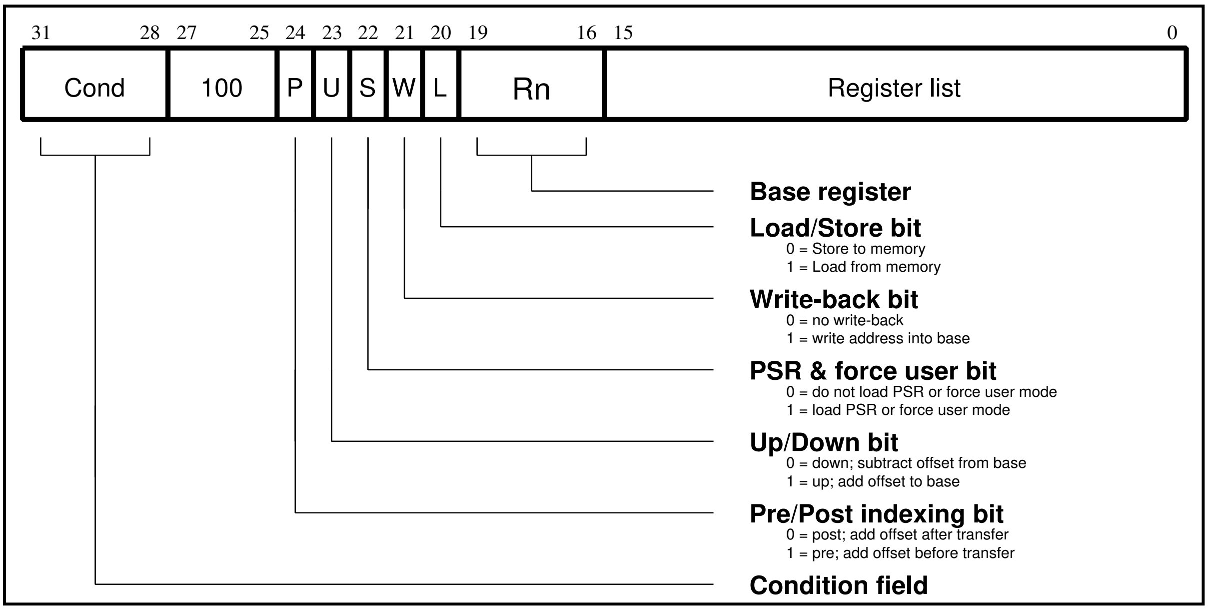


Figure 5-16: Block data transfer instructions

5.8.2 Addressing modes

The transfer addresses are determined by :

- the contents of the base register (Rn)
- the pre/post bit (P)
- the up/down bit (U)

The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address.

By way of illustration, consider the transfer of R1, R5 and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1).

Figure 5-17: Post-increment addressing, *Figure 5-18: Pre-increment addressing*, *Figure 5-19: Post-decrement addressing*, and *Figure 5-20: Pre-decrement addressing* on page 5-32, show the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

5.8.3 Address alignment

The address should always be a word aligned quantity.

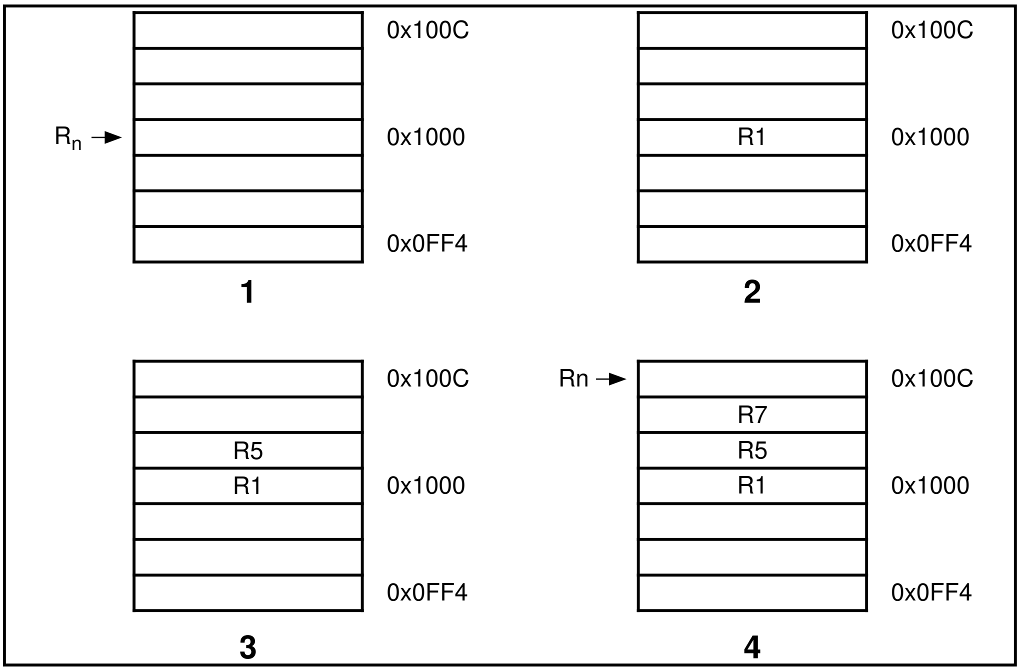


Figure 5-17: Post-increment addressing

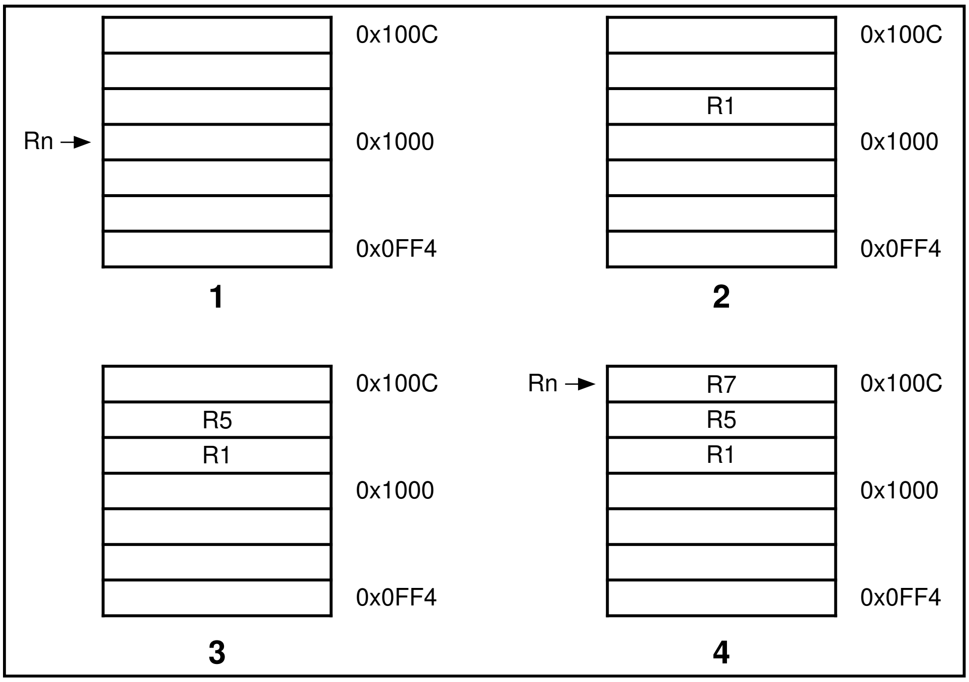


Figure 5-18: Pre-increment addressing

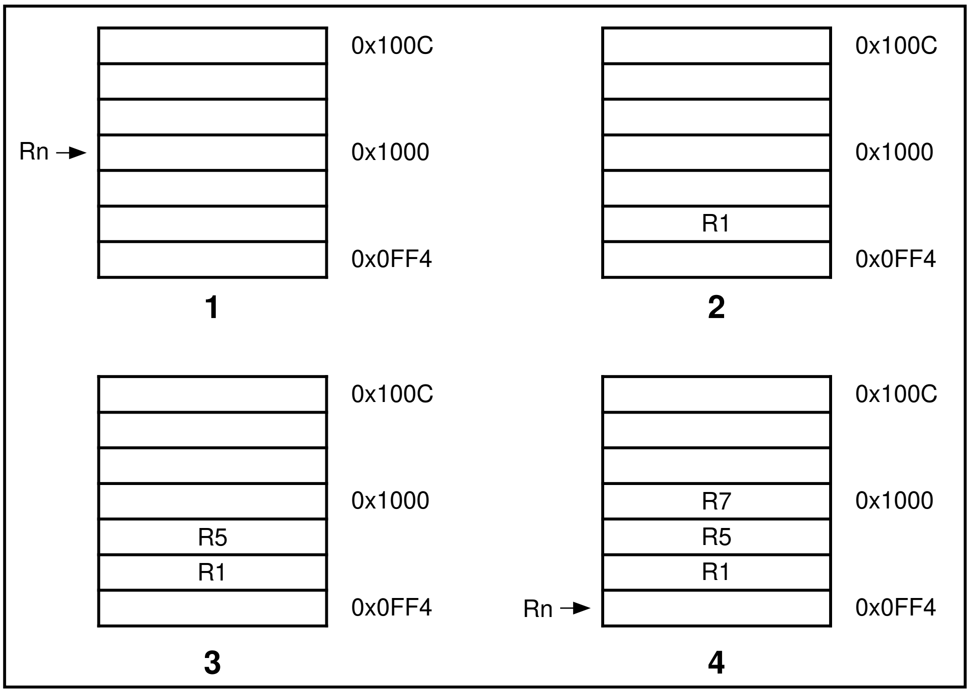


Figure 5-19: Post-decrement addressing

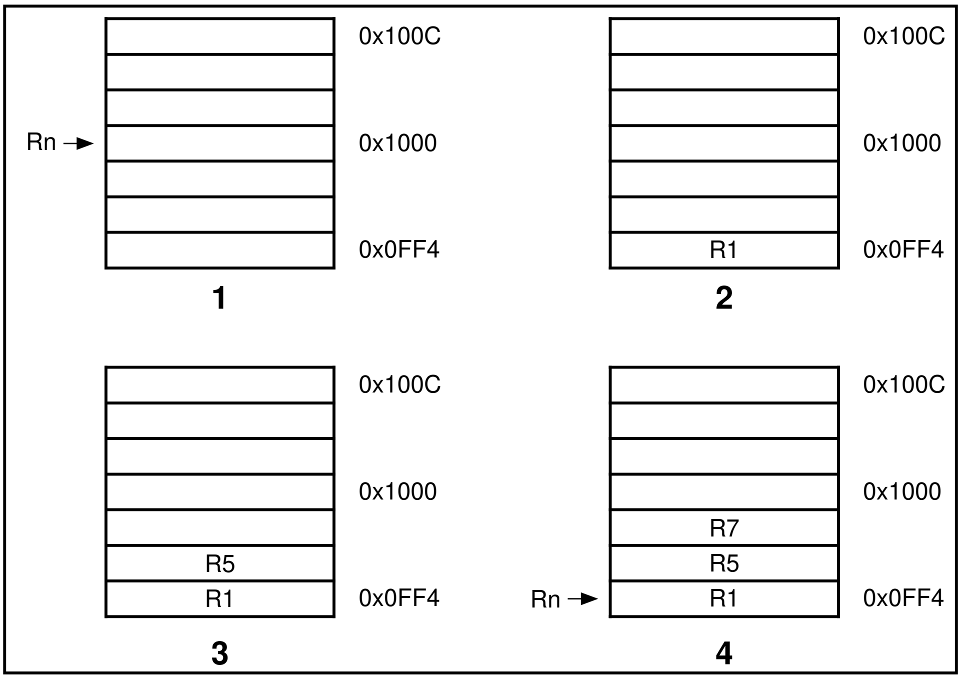


Figure 5-20: Pre-decrement addressing

5.8.4 Use of the S bit

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in transfer list and S bit set (Mode changes)

If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in transfer list and S bit set (User bank transfer)

The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back shall not be used when this mechanism is employed.

R15 not in list and S bit set (User bank transfer)

For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back shall not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a NOP after the LDM will ensure safety).

5.8.5 Use of R15 as the base register

R15 must not be used as the base register in any LDM or STM instruction.

5.8.6 Inclusion of the base in the register list

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During an STM, the first register is written out at the start of the second cycle. An STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. An LDM will always overwrite the updated base if the base is in the list.

5.8.7 Data aborts

Some legal addresses may be unacceptable to the MMU. The MMU will then cause an abort. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM7500 is to be used in a virtual memory system.

Aborts during STM instructions

If the abort occurs during a store multiple instruction, ARM Processor takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM instructions

When ARM Processor detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

1. Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
2. The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

5.8.8 Instruction cycle times

Instruction	Cycles
Normal LDM instructions	1 instruction fetch, *n* data reads and 1 internal cycle
LDM PC	3 instruction fetches, *n* data reads and 1 internal cycle.
STM instructions	instruction fetch, *n* data reads and 1 internal cycle, where *n* is the number of words transferred.

Table 5-5: Instruction cycle times

For more information see *5.16.7 Instruction speed summary* on page 5-55.

5.8.9 Assembler syntax

`<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}`

{cond} - two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

Rn is an expression evaluating to a valid register number

<Rlist> is a list of registers and register ranges enclosed in {} (e.g. {R0,R2-R7,R10}).

{!} if present requests write-back (W=1), otherwise W=0

{^} if present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode

5.8.10 Addressing mode names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalencies between the names and the values of the bits in the instruction are shown in *Table 5-6: Addressing mode names*:

Key to table

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required.

F	Full stack (a pre-index has to be done before storing to the stack)
E	Empty stack
A	The stack is ascending (an STM will go up and LDM down)
D	The stack is decending (an STM will go down and LDM up)

The following symbols allow control when LDM/STM are not being used for stacks:

IA	Increment After
IB	Increment Before
DA	Decrement After
DB	Decrement Before

Name	Stack	Other	L-bit	P-bit	U-bit
pre-increment load	LDMED	LDMIB	1	1	1
post-increment load	LDMFD	LDMIA	1	0	1
pre-decrement load	LDMEA	LDMDB	1	1	0
post-decrement load	LDMFA	LDMDA	1	0	0
pre-increment store	STMFA	STMIB	0	1	1
post-increment store	STMEA	STMIA	0	0	1
pre-decrement store	STMFD	STMDB	0	1	0
post-decrement store	STMED	STMDA	0	0	0

Table 5-6: Addressing mode names

5.8.11 Examples

```
LDMFD SP!,{R0,R1,R2} ;unstack 3 registers
STMIA R0,{R0-R15}    ;save all registers
LDMFD SP!,{R15}      ;R15 <- (SP),CPSR unchanged
LDMFD SP!,{R15}^     ;R15 <- (SP), CPSR <- SPSR_mode (allowed
                     ;only in privileged modes)
STMFD R13,{R0-R14}^  ;save user mode regs on stack (allowed
                     ;only in privileged modes)
```

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

```
STMED SP!,{R0-R3,R14};
                     ;save R0 to R3 to use as workspace
                     ;and R14 for returning
BL    somewhere      ;this nested call will overwrite R14
LDMED SP!,{R0-R3,R15}
                     ;restore workspace and return
```

5.9 Single data swap (SWP)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-21: Swap instruction.*

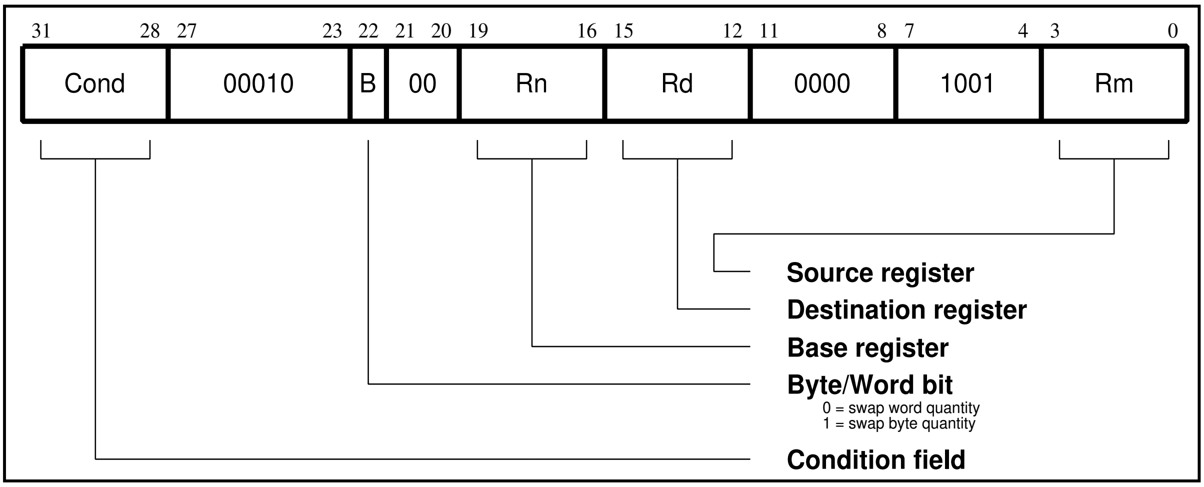


Figure 5-21: Swap instruction

Data swap instruction

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

Swap address

The swap address is determined by the contents of the base register (Rn). The processor first read the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register can be specified as both the source and the destination.

ARM710 lock feature

The ARM7500 does not use the lock feature available in the ARM710 macrocell. You must take care to ensure that control of the memory is not removed from the ARM Processor while it is performing this instruction.

5.9.1 Bytes and words

This instruction class may be used to swap a byte (B=1) or a word (B=0) between an ARM Processor register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and Little Endian configuration applies to the SWP instruction.

5.9.2 Use of R15

Do not use R15 as an operand (Rd, Rn or Rs) in a SWP instruction.

5.9.3 Data aborts

If the address used for the swap is unacceptable to the MMU, it will cause an abort. This can happen on either the read or write cycle (or both), and, in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem. The instruction can then be restarted and the original program continued.

5.9.4 Instruction cycle times

Swap instructions take 1 instruction fetch, 1 data read, 1 data write and 1 internal cycle. For more information see *5.16.7 Instruction speed summary* on page 5-55.

5.9.5 Assembler syntax

```
<SWP>{cond}{B} Rd,Rm,[Rn]
```

{cond}	two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
{B}	if B is present then byte transfer, otherwise word transfer
Rd,Rm,Rn	are expressions evaluating to valid register numbers

5.9.6 Examples

```
SWP   R0,R1,[R2]     ;load R0 with the word addressed by R2, and
                     ;store R1 at R2
SWPB  R2,R3,[R4]     ;load R2 with the byte addressed by R4, and
                     ;store bits 0 to 7 of R3 at R4
SWPEQ R0,R0,[R1]     ;conditionally swap the contents of R1
                     ;with R0
```

5.10 Software interrupt (SWI)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-22: Software interrupt instruction*. The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

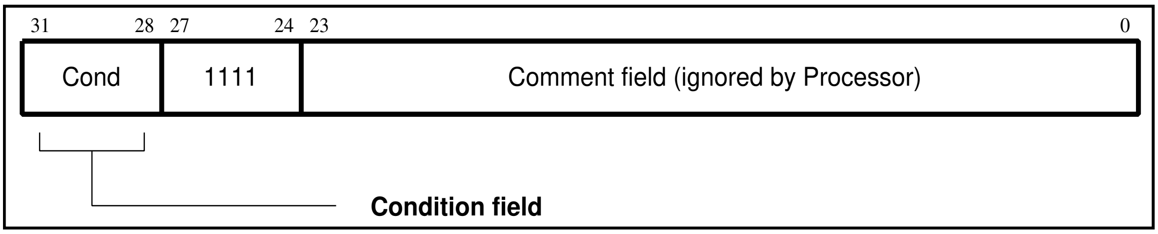


Figure 5-22: Software interrupt instruction

5.10.1 Return from the supervisor

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note: The link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

5.10.2 Comment field

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

5.10.3 Instruction cycle times

Software interrupt instructions take 3 instruction fetches. For more information see *5.16.7 Instruction speed summary* on page 5-55.

5.10.4 Assembler syntax

```
SWI{cond} <expression>
```

{cond} — two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

<expression> — is evaluated and placed in the comment field (ignored by ARM Processor).

5.10.5 Examples

```
SWI   ReadC          ;get next character from read stream
SWI   WriteI+"k"     ;output a "k" to the write stream
SWINE 0              ;conditionally call supervisor
                     ;with 0 in comment field
```

The above examples assume that suitable supervisor code exists, for instance:

```
0x08 B Supervisor    ;SWI entry point

EntryTable           ;addresses of supervisor routines
               DCD ZeroRtn
               DCD ReadCRtn
               DCD WriteIRtn
               ...
      Zero     EQU      0
      ReadC    EQU      256
      WriteI   EQU      512

Supervisor

;SWI has routine required in bits 8-23 and data (if any) in bits
;0-7.
;Assumes R13_svc points to a suitable stack

STMFD R13,{R0-R2,R14}; save work registers and return address
LDR   R0,[R14,#-4]   ;get SWI instruction
BIC   R0,R0,#0xFF000000;
                     ;clear top 8 bits
MOV   R1,R0,LSR#8    ;get routine offset
ADR   R2,EntryTable  ;get start address of entry table
LDR   R15,[R2,R1,LSL#2];
                     ;branch to appropriate routine

WriteIRtn            ;enter with character in R0 bits 0-7
          .  .  .  .  .  .
LDMFD R13,{R0-R2,R15}^;
                     ;restore workspace and return
                     ; restoring processor mode and flags
```

5.11 Coprocessor Instructions on ARM Processor

The ARM Processor, unlike some other ARM processors, does not have an external coprocessor interface. The ARM Processor only supports a single on chip coprocessor, #15, which is used to program the on-chip control registers. This only supports the Coprocessor Register Transfer instructions (MRC and MCR).

All other coprocessor instructions will cause the undefined instruction trap to be taken on the ARM Processor. These coprocessor instructions can be emulated in software by the undefined trap handler. Even though external coprocessors can not be connected to ARM Processor, the coprocessor instructions are still described here in full for completeness. It must be kept in mind that any external coprocessor referred to will be a software emulation.

5.12 Coprocessor data operations (CDP)

Use of the CDP instruction on the ARM Processor will cause an undefined instruction trap to be taken, which may be used to emulate the coprocessor instruction.

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-23: Coprocessor data operation instruction*.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to the processor, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity allowing the coprocessor and the processor to perform independent tasks in parallel.

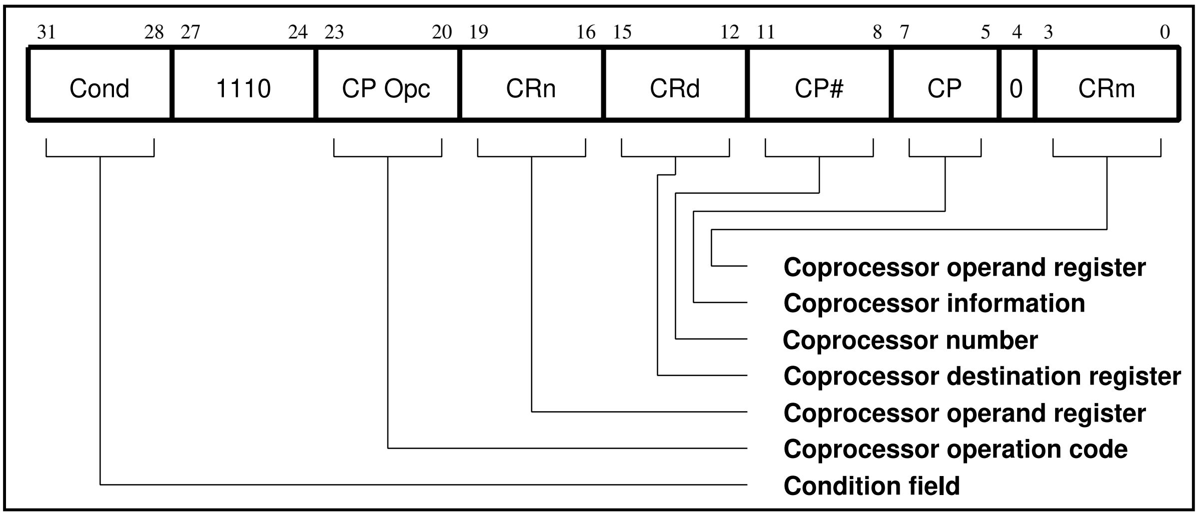


Figure 5-23: Coprocessor data operation instruction

5.12.1 The coprocessor fields

Only bit 4 and bits 24 to 31 are significant to the processor; the remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

5.12.2 Instruction cycle times

All CDP instructions are emulated in software: the number of cycles taken will depend on the coprocessor support software.

5.12.3 Assembler syntax

```
CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}
```

{cond}	two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
p#	the unique number of the required coprocessor
<expression1>	evaluated to a constant and placed in the CP Opc field
cd, cn and cm	evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively
<expression2>	where present, is evaluated to a constant and placed in the CP field

5.12.4 Examples

```
CDP   p1,10,c1,c2,c3 ;request coproc 1 to do operation 10
                     ;on CR2 and CR3, and put the result in CR1
CDPEQ p2,5,c1,c2,c3,2;
                     ;if Z flag is set request coproc 2 to do
                     ;operation 5 (type 2) on CR2 and CR3,
                     ;and put the result in CR1
```

5.13 Coprocessor data transfers (LDC, STC)

Use of the LDC or STC instruction on the ARM Processor will cause an undefined instruction trap to be taken, which may be used to emulate the coprocessor instruction.

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-24: Coprocessor data transfer instructions*.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessors's registers directly to memory. The processor is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.

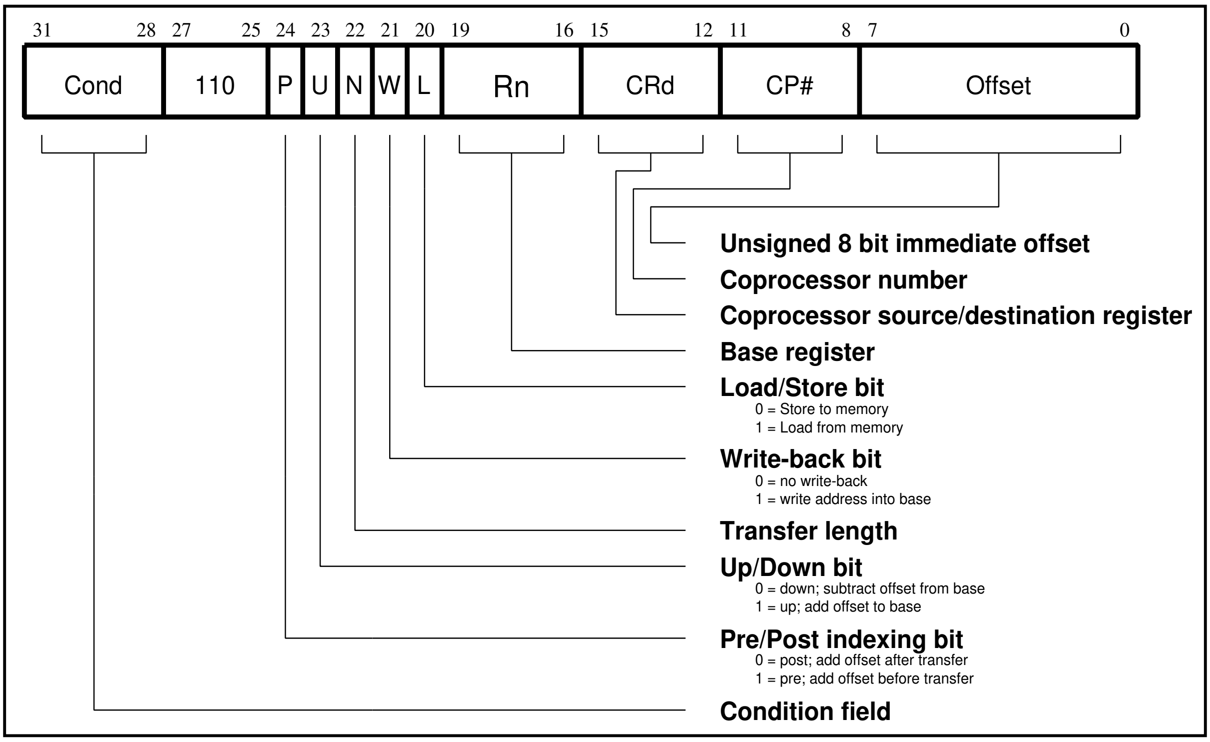


Figure 5-24: Coprocessor data transfer instructions

5.13.1 The coprocessor fields

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options.

For example:

N=0	could select the transfer of a single register
N=1	could select the transfer of all the registers for context switching.

5.13.2 Addressing modes

The processor is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8 bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0).

Note: Post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

5.13.3 Address alignment

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

5.13.4 Use of R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

5.13.5 Data aborts

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

5.13.6 Instruction cycle times

All LDC instructions are emulated in software: the number of cycles taken will depend on the coprocessor support software.

5.13.7 Assembler syntax

```
<LDC|STC>{cond}{L} p#,cd,<Address>
```

LDC	load from memory to coprocessor
STC	store from coprocessor to memory
{L}	when present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}	two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
p#	the unique number of the required coprocessor
cd	is an expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

1 An expression which generates an address:

```
<expression>
```

The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2 A pre-indexed addressing specification:

```
[Rn]                          offset of zero
[Rn,<#expression>]{!}         offset of <expression> bytes
```

3 A post-indexed addressing specification:

```
[Rn],<#expression>            offset of <expression> bytes
```

Rn	is an expression evaluating to a valid processor register number. Note, if Rn is R15 then the assembler will subtract 8 from the offset value to allow for processor pipelining.
{!}	write back the base register (set the W bit) if ! is present

5.13.8 Examples

```
LDC   p1,c2,table        ;load c2 of coproc 1 from address table,
                         ;using a PC relative address.
STCEQLp2,c3,[R5,#24]!    ;conditionally store c3 of coproc 2
                         ;into an address 24 bytes up from R5,
                         ;write this address back to R5, and use
                         ;long transfer
                         ;option (probably to store multiple
                         ;words)
```

Note: Though the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

5.14 Coprocessor register transfers (MRC, MCR)

Use of the MRC or MCR instruction on the ARM Processor to a coprocessor other than number 15 will cause an undefined instruction trap to be taken, which may be used to emulate the coprocessor instruction.

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-25: Coprocessor register transfer instructions*.

This class of instruction is used to communicate information directly between ARM Processor and a coprocessor. An example of a coprocessor to processor register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32-bit integer within the coprocessor, and the result is then transferred to a processor register. A FLOAT of a 32-bit value in a processor register into a floating point value within the coprocessor illustrates the use of a processor register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the processor CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

Note: *The ARM Processor has an internal coprocessor (#15) for control of on-chip functions. Accesses to this coprocessor are performed during coprocessor register transfers.*

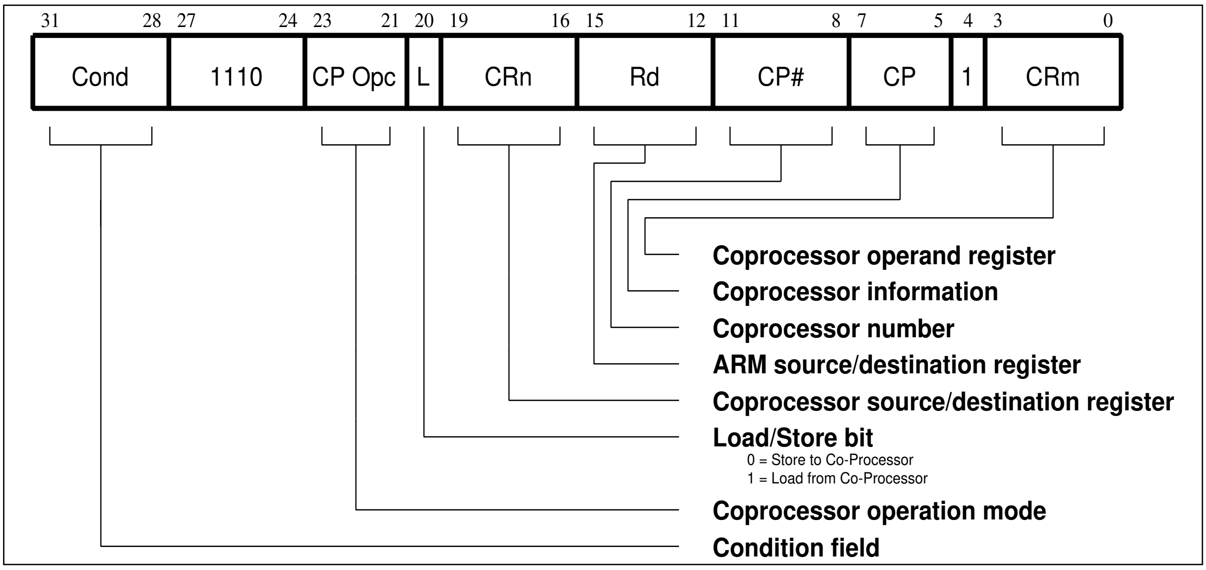


Figure 5-25: Coprocessor register transfer instructions

5.14.1 The coprocessor fields

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.

5.14.2 Transfers to R15

When a coprocessor register transfer to ARM Processor has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

5.14.3 Transfers from R15

A coprocessor register transfer from ARM Processor with R15 as the source register will store the PC+12.

5.14.4 Instruction cycle times

Access to the internal configuration register takes 3 internal cycles. All other MRC instructions default to software emulation, and the number of cycles taken will depend on the coprocessor support software.

5.14.5 Assembler syntax

```
<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}
```

MRC - move from coprocessor to ARM7500 register (L=1)

MCR - move from ARM7500 register to coprocessor (L=0)

{cond} - two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

p# - the unique number of the required coprocessor

<expression1> - evaluated to a constant and placed in the CP Opc field

Rd is an expression evaluating to a valid ARM Processor register number

cn and cm are expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively

<expression2> - where present is evaluated to a constant and placed in the CP field

5.14.6 Examples

```
MRC   2,5,R3,c5,c6   ;request coproc 2 to perform operation 5
                     ;on c5 and c6, and transfer the (single
                     ;32-bit word) result back to R3
MCR   6,0,R4,c6      ;request coproc 6 to perform operation 0
                     ;on R4 and place the result in c6
MRCEQ 3,9,R3,c5,c6,2 ;conditionally request coproc 2 to
                     ;perform
                     ;operation 9 (type 2) on c5 and c6, and
                     ;transfer the result back to R3
```

5.15 Undefined instruction

31–28	27–25	24–5	4	3–0
Cond	011	xxxxxxxxxxxxxxxxxxxx	1	xxxx

Figure 5-26: Undefined instruction

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction format is shown in *Figure 5-26: Undefined instruction* on page 5-51.

If the condition is true, the undefined instruction trap will be taken.

5.15.1 Assembler syntax

At present the assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction shall not be used.

5.16 Instruction set examples

The following examples show ways in which the basic ARM Processor instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

5.16.1 Using the conditional instructions

1 using conditionals for logical OR

```
CMP      Rn,#p    ;if Rn=p OR Rm=q THEN GOTO Label
BEQ      Label
CMP      Rm,#q
BEQ      Label
```

can be replaced by

```
CMP      Rn,#p
CMPNE    Rm,#q    ;if condition not satisfied try other
                  ;test
BEQ      Label
```

2 absolute value

```
TEQ      Rn,#0    ;test sign
RSBMI    Rn,Rn,#0 ;and 2's complement if necessary
```

3 multiplication by 4, 5 or 6 (run time)

```
MOV      Rc,Ra,LSL#2;
                  ;multiply by 4
CMP      Rb,#5    ; test value
ADDCS    Rc,Rc,Ra ; complete multiply by 5
ADDHI    Rc,Rc,Ra ; complete multiply by 6
```

4 combining discrete and range tests

```
TEQ      Rc,#127  ;discrete test
CMPNE    Rc,#” “-1;
                  ;range test
MOVLS    Rc,#”.”  ;IF   Rc<=” “ OR Rc=ASCII(127)
                  ;THEN Rc:=”.”
```

5 division and remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine follows.

```
                        ;enter with numbers in Ra and Rb
                        ;
      MOV      Rcnt,#1  ;bit to control the division
Div1  CMP      Rb,#0x80000000;
                        ;move Rb until greater than Ra
      CMPCC    Rb,Ra
      MOVCC    Rb,Rb,ASL#1
      MOVCC    Rcnt,Rcnt,ASL#1
```

```
      BCC     Div1
      MOV     Rc,#0
Div2  CMP     Ra,Rb    ;test for possible subtraction
      SUBCS   Ra,Ra,Rb ;subtract if ok
      ADDCS   Rc,Rc,Rcnt;
                       ;put relevant bit into result
      MOVS    Rcnt,Rcnt,LSR#1;
                       ;shift control bit
      MOVNE   Rb,Rb,LSR#1;
                       ;halve unless finished
      BNE     Div2
                       ;
                       ;divide result in Rc
                       ;remainder in Ra
```

5.16.2 Pseudo random binary sequence generator

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32-bit generator needs more than one feedback tap to be maximal length (i.e. 2^32-1 cycles before repetition), so this example uses a 33-bit register with taps at bits 33 and 20. The basic algorithm is newbit:=bit 33 or bit 20, shift left the 33-bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32 bits). The entire operation can be done in 5 S cycles:

```
                     ;enter with seed in Ra (32 bits),
                     ;Rb (1 bit in Rb lsb), uses Rc
                     ;
TST   Rb,Rb,LSR#1    ;top bit into carry
MOVS  Rc,Ra,RRX      ;33 bit rotate right
ADC   Rb,Rb,Rb       ;carry into lsb of Rb
EOR   Rc,Rc,Ra,LSL#12;
                     ;(involved!)
EOR   Ra,Rc,Rc,LSR#20;
                     ;(similarly involved!)
                     ;
                     ;new seed in Ra, Rb as before
```

5.16.3 Multiplication by constant using the barrel shifter

1 Multiplication by 2^n (1,2,4,8,16,32..)

```
MOV      Ra, Rb, LSL #n
```

2 Multiplication by 2^n+1 (3,5,9,17..)

```
ADD      Ra,Ra,Ra,LSL #n
```

3 Multiplication by 2^n-1 (3,7,15..)

```
RSB      Ra,Ra,Ra,LSL #n
```

4 Multiplication by 6

```
ADD      Ra,Ra,Ra,LSL #1;      ;multiply by 3
MOV      Ra,Ra,LSL#1;          ;and then by 2
```

5 Multiply by 10 and add in extra number

```
ADD      Ra,Ra,Ra,LSL#2;       ;multiply by 5
ADD      Ra,Rc,Ra,LSL#1;       ;multiply by 2
                               ;and add in next digit
```

6 General recursive method for Rb := Ra*C, C a constant:

a) If C even, say C = 2^n*D, D odd:

```
D=1:     MOV   Rb,Ra,LSL #n
D<>1:    {Rb := Ra*D}
         MOV   Rb,Rb,LSL #n
```

b) If C MOD 4 = 1, say C = 2^n*D+1, D odd, n>1:

```
D=1:     ADD   Rb,Ra,Ra,LSL #n
D<>1:    {Rb := Ra*D}
         ADD   Rb,Ra,Rb,LSL #n
```

c) If C MOD 4 = 3, say C = 2^n*D-1, D odd, n>1:

```
D=1:     RSB   Rb,Ra,Ra,LSL #n
D<>1:    {Rb := Ra*D}
         RSB   Rb,Ra,Rb,LSL #n
```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```
RSB      Rb,Ra,Ra,LSL#2;       ;multiply by 3
RSB      Rb,Ra,Rb,LSL#2;       ;multiply by 4*3-1 = 11
ADD      Rb,Ra,Rb,LSL# 2;      ;multiply by 4*11+1 = 45
```

rather than by:

```
ADD      Rb,Ra,Ra,LSL#3;       ;multiply by 9
ADD      Rb,Rb,Rb,LSL#2;       ;multiply by 5*9 = 45
```

5.16.4 Loading a word from an unknown alignment

```
                         ;enter with address in Ra (32 bits)
                         ;uses Rb, Rc; result in Rd.
```

```
                             ; Note d must be less than c e.g. 0,1
                             ;
BIC    Rb,Ra,#3              ;get word aligned address
LDMIA Rb,{Rd,Rc}             ;get 64 bits containing answer
AND    Rb,Ra,#3              ;correction factor in bytes
MOVS   Rb,Rb,LSL#3           ;...now in bits and test if aligned
MOVNE Rd,Rd,LSR Rb           ;produce bottom of result word
                             ;(if not aligned)
RSBNE Rb,Rb,#32              ;get other shift amount
ORRNE Rd,Rd,Rc,LSL Rb;       ;combine two halves to get result
```

5.16.5 Loading a halfword (Little Endian)

```
LDR    Ra, [Rb,#2]     ;get halfword to bits 15:0
MOV    Ra,Ra,LSL #16   ;move to top
MOV    Ra,Ra,LSR #16   ;and back to bottom
                       ;use ASR to get sign extended version
```

5.16.6 Loading a halfword (Big Endian)

```
LDR    Ra, [Rb,#2]     ;get halfword to bits 31:16
MOV    Ra,Ra,LSR #16   ;and back to bottom
                       ;use ASR to get sign extended version
```

5.16.7 Instruction speed summary

Due to the pipelined architecture of the CPU, instructions overlap considerably. In a typical cycle one instruction may be using the data path while the next is being decoded and the one after that is being fetched. For this reason the following table presents the incremental number of cycles required by an instruction, rather than the total number of cycles for which the instruction uses part of the processor. Elapsed time (in cycles) for a routine may be calculated from these figures which are shown in ➲*Table 5-7: ARM instruction speed summary* on page 5-56.

These figures assume that the instruction is actually executed.

Unexecuted instructions take one instruction fetch cycle.

Instruction	Cycle count
Data Processing - normal with register specified shift with PC written with register specified shift & PC written	1 instruction fetch 1 instruction fetch and 1 internal cycle 3 instruction fetches 3 instruction fetches and 1 internal cycle
MSR, MRS	1 instruction fetch
LDR - normal if the destination is the PC	1 instruction fetch, 1 data read and 1 internal cycle 3 instruction fetches, 1 data read and 1 internal cycle
STR	1 instruction fetch and 1 data write
LDM - normal if the destination is the PC	1 instruction fetch, n data reads and 1 internal cycle 3 instruction fetches, n data reads and 1 internal cycle
STM	1 instruction fetch and n data writes
SWP	1 instruction fetch, 1 data read, 1 data write and 1 internal cycle
B,BL	3 instruction fetches
SWI, trap	3 instruction fetches
MUL,MLA	1 instruction fetch and m internal cycles
CDP	the undefined instruction trap will be taken
LDC	the undefined instruction trap will be taken
STC	the undefined instruction trap will be taken
MCR	1 instruction fetch and 2 internal cycles for coproc 15
MRC	1 instruction fetch and 2 internal cycles for coproc 15

Table 5-7: ARM instruction speed summary

Where:

n is the number of words transferred.

m is the number of cycles required by the multiply algorithm, which is determined by the contents of Rs. Multiplication by any number between *2^(2m-3)* and *2^(2m-1)-1* takes 1S+mI cycles for *1<m>16*. Multiplication by 0 or 1 takes 1S+1I cycles, and multiplication by any number greater than or equal to 2^(29) takes 1S+16I cycles. The maximum time for any multiply is thus 1S+16I cycles.

The time taken for:

- an internal cycle - will always be one FCLK cycle
- an instruction fetch and data read - will be FCLK if a cache hit occurs, otherwise a full memory access is performed.

- a data write - will be FCLK if the write buffer (if enabled) has available space, otherwise the write will be delayed until the write buffer has free space. If the write buffer is not enabled a full memory access is always performed.
- Co-processor cycles - will be one CPCLK cycle, but see the section on Co-processors for more informational coprocessor operations except MCR or MRC to registers 0 to 7 on coprocessor #15 (used for internal control) will cause the undefined instruction trap to be taken.
- memory accesses - are dealt with elsewhere in the ARM7500 datasheet.

6 Cache, Write Buffer and Coprocessors

The chapter describes the ARM processor instruction and data cache, and its write buffer.

6.1 Instruction and Data Cache (IDC)

ARM processor contains a 4Kbyte mixed instruction and data cache. The IDC has 256 lines of 16 bytes (4 words), organised as a 4-way set associative cache, and uses the virtual addresses generated by the processor core. The IDC is always reloaded a line at a time (4 words). It may be enabled or disabled via the ARM processor Control Register and is disabled on **nRESET**.

The operation of the cache is further controlled by the *Cacheable* or C bit stored in the Memory Management Page Table (see the Memory Management Unit chapter). For this reason, in order to use the IDC, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register.

6.1.1 Cacheable bit

The *Cacheable* bit determines whether data being read may be placed in the IDC and used for subsequent read operations. Typically main memory will be marked as Cacheable to improve system performance, and I/O space as Non-cacheable to stop the data being stored in ARM7500's cache. [For example if the processor is polling a hardware flag in I/O space, it is important that the processor is forced to read data from the external peripheral, and not a copy of initial data held in the cache]. The Cacheable bit can be configured for both pages and sections.

6.1.2 IDC operation

In the ARM processor the cache will be searched regardless of the state of the C bit, only reads that miss the cache will be affected.

Cacheable Reads	C = 1
	A linefetch of 4 words will be performed and it will be randomly placed in a cache bank.
Uncacheable Reads	C = 0
	An external memory access will be performed and the cache will not be written.

6.1.3 IDC validity

The IDC operates with virtual addresses, so care must be taken to ensure that its contents remain consistent with the virtual to physical mappings performed by the Memory Management Unit. If the Memory Mappings are changed, the IDC validity must be ensured.

Software IDC Flush

The entire IDC may be marked as invalid by writing to the ARM processor IDC Flush Register (Register 7). The cache will be flushed immediately the register is written, but note that the next two instruction fetches may come from the cache before the register is written.

6.1.4 Doubly mapped space

Since the cache works with virtual addresses, it is assumed that every virtual address maps to a different physical address. If the same physical location is accessed by more than one virtual address, the cache cannot maintain consistency, since each virtual address will have a separate entry in the cache, and only one entry will be updated on a processor write operation. To avoid any cache inconsistencies, both doubly-mapped virtual addresses should be marked as uncacheable.

6.2 Read-Lock-Write

The IDC treats the Read-Locked-Write instruction as a special case. The read phase always forces a read of external memory, regardless of whether the data is contained in the cache. The write phase is treated as a normal write operation (and if the data is already in the cache, the cache will be updated). Externally the two phases are flagged as indivisible by asserting the **LOCK** signal.

6.3 IDC Enable/Disable and Reset

The IDC is automatically disabled and flushed on **nRESET**. Once enabled, cacheable read accesses will cause lines to be placed in the cache.

6.3.1 To enable the IDC

To enable the IDC, make sure that the MMU is enabled first by setting bit 0 in Control Register, then enable the IDC by setting bit 2 in Control Register. The MMU and IDC may be enabled simultaneously with a single control register write.

6.3.2 To disable the IDC

To disable the IDC, clear bit 2 in the Control Register and perform a flush by writing to the flush register.

6.4 Write buffer (WB)

The ARM processor write buffer is provided to improve system performance. It can buffer up to 8 words of data, and 4 independent addresses. It may be enabled or disabled via the W bit (bit 3) in the ARM processor Control Register and the buffer is disabled and flushed on reset.

The operation of the write buffer is further controlled by one bit, B, or Bufferable, which is stored in the Memory Management Page Tables. For this reason, in order to use the write buffer, the MMU must be enabled.

The two functions may however be enabled simultaneously, with a single write to the Control Register. For a write to use the write buffer, both the W bit in the Control Register, and the B bit in the corresponding page table must be set.

6.4.1 Bufferable bit

This bit controls whether a write operation may or may not use the write buffer. Typically main memory will be bufferable and I/O space unbufferable. The Bufferable bit can be configured for both pages and sections.

6.4.2 Write buffer operation

When the CPU performs a write operation, the translation entry for that address is inspected and the state of the B bit determines the subsequent action. If the write buffer is disabled via the ARM processor Control Register, bufferable writes are treated in the same way as unbuffered writes.

Bufferable write

If the write buffer is enabled and the processor performs a write to a bufferable area, the data is placed in the write buffer at FCLK speeds and the CPU continues execution. The write buffer then performs the external write in parallel. If however the write buffer is full (either because there are already 8 words of data in the buffer, or because there is no slot for the new address) then the processor is stalled until there is sufficient space in the buffer.

Unbufferable writes

If the write buffer is disabled or the CPU performs a write to an unbufferable area, the processor is stalled until the write buffer empties and the write completes externally, which may require synchronisation and several external clock cycles.

Read-lock-write

The write phase of a read-lock-write sequence is treated as an Unbuffered write, even if it is marked as buffered.

Note: *A single write requires one address slot and one data slot in the write buffer; a sequential write of n words requires one address slot and n data slots. The total of 8 data slots in the buffer may be used as required. So for instance there could be 3 non-sequential writes and one sequential write of 5 words in the buffer, and the processor could continue as normal: a 5th write or an 6th word in the 4th write would stall the processor until the first write had completed.*

To enable the write buffer

To enable the write buffer, ensure the MMU is enabled by setting bit 0 in the Control Register, then enable the write buffer by setting bit 3 in the Control Register. The MMU and write buffer may be enabled simultaneously with a single write to the Control Register.

To disable the write buffer

To disable the write buffer, clear bit 3 in the Control Register.

Note: *Any writes already in the write buffer will complete normally.*

6.5 Coprocessors

ARM processor has no external coprocessor bus, so it is not possible to add external coprocessors to this device.

The ARM processor still has an internal coprocessor designated #15 for internal control of the device. All coprocessor operations except MCR or MRC to registers 0 to 7 on coprocessor #15 will cause the undefined instruction trap to be taken.

7 ARM Processor MMU

This chapter describes the ARM processor Memory Management Unit.

7.1 Introduction

The MMU performs two primary functions: it translates virtual addresses into physical addresses, and it controls memory access permissions. The MMU hardware required to perform these functions consists of a Translation Look-aside Buffer (TLB), access control logic, and translation table walking logic.

The MMU supports memory accesses based on Sections or Pages. Sections are comprised of 1MB blocks of memory. Two different page sizes are supported: Small Pages consist of 4Kb blocks of memory and Large Pages consist of 64Kb blocks of memory. (Large Pages are supported to allow mapping of a large region of memory while using only a single entry in the TLB.) Additional access control mechanisms are extended within Small Pages to 1Kb Sub-Pages and within Large Pages to 16Kb Sub-Pages.

The MMU also supports the concept of domains - areas of memory that can be defined to possess individual access rights. The Domain Access Control Register is used to specify access rights for up to 16 separate domains.

The TLB caches 64 translated entries. During most memory accesses, the TLB provides the translation information to the access control logic.

If the TLB contains a translated entry for the virtual address, the access control logic determines whether access is permitted. If access is permitted, the MMU outputs the appropriate physical address corresponding to the virtual address. If access is not permitted, the MMU signals the CPU to abort.

If the TLB misses (it does not contain a translated entry for the virtual address), the translation table walk hardware is invoked to retrieve the translation information from a translation table in physical memory. Once retrieved, the translation information is placed into the TLB, possibly overwriting an existing value. The entry to be overwritten is chosen by cycling sequentially through the TLB locations.

When the MMU is turned off (as happens on reset), the virtual address is output directly onto the physical address bus.

7.2 MMU program-accessible registers

The ARM processor provides several 32-bit registers which determine the operation of the MMU. The format for these registers and a brief description is shown in *Figure 7-1: MMU register summary* on page 7-3. Each register will be discussed in more detail within the section that describes its use.

Data is written to and read from the MMUs registers using the ARM CPU's MRC and MCR coprocessor instructions.

Register	Bits 31–0
1 write	0 (bit 31) · Control (bits 30–10) · R · S · B · 1 · D · P · W · C · A · M (bits 9–0)
2 write	Translation Table Base (upper bits) · (shaded, lower bits)
3 write	Domain Access Control: 15 · 14 · 13 · 12 · 11 · 10 · 9 · 8 · 7 · 6 · 5 · 4 · 3 · 2 · 1 · 0
5 read	Fault Status (bits 31–12) · 0 · 0 · 0 · 0 (bits 11–8) · Domain (bits 7–4) · Status (bits 3–0)
5 write	Flush TLB
6 read	Fault Address
6 write	TLB Purge Address (upper bits) · (shaded, lower bits)

Figure 7-1: MMU register summary

Translation Table Base Register

The Translation Table Base Register holds the physical address of the base of the translation table maintained in main memory. Note that this base must reside on a 16KB boundary.

Domain Access Control Register

The Domain Access Control Register consists of sixteen 2-bit fields, each of which defines the access permissions for one of the sixteen Domains (D15-D0).

Note: *The registers not shown are reserved and should not be used.*

Fault Status Register

The Fault Status Register indicates the domain and type of access being attempted when an abort occurred. Bits 7:4 specify which of the sixteen domains (D15-D0) was being accessed when a fault occurred. Bits 3:1 indicate the type of access being attempted. The encoding of these bits is different for internal and external faults (as indicated by bit 0 in the register) and is shown in *Table 7-4: Priority encoding of fault status* on page 7-12. A write to this register flushes the TLB.

Fault Address Register

The Fault Address Register holds the virtual address of the access which was attempted when a fault occurred. A write to this register causes the data written to be treated as an address and, if it is found in the TLB, the entry is marked as invalid. (This operation is known as a TLB purge). The Fault Status Register and Fault Address Register are only updated for data faults, not for prefetch faults.

7.3 Address translation

The MMU translates virtual addresses generated by the CPU into physical addresses to access external memory, and also derives and checks the access permission. Translation information, which consists of both the address translation data and the access permission data, resides in a translation table located in physical memory. The MMU provides the logic needed to traverse this translation table, obtain the translated address, and check the access permission.

There are three routes by which the address translation (and hence permission check) takes place. The route taken depends on whether the address in question has been marked as a section-mapped access or a page-mapped access; and there are two sizes of page-mapped access (large pages and small pages). However, the translation process always starts out in the same way, as described below, with a Level One fetch. A section-mapped access only requires a Level One fetch, but a page-mapped access also requires a Level Two fetch.

7.4 Translation process

7.4.1 Translation table base

The translation process is initiated when the on-chip TLB does not contain an entry for the requested virtual address. The Translation Table Base (TTB) Register points to the base of a table in physical memory which contains Section and/or Page descriptors. The 14 low-order bits of the TTB Register are set to zero as illustrated in *Figure 7-2: Translation table base register*; the table must reside on a 16Kb boundary.

31 — 14	13 — 0
Translation Table Base	

Figure 7-2: Translation table base register

7.4.2 Level one fetch

Bits 31:14 of the Translation Table Base register are concatenated with bits 31:20 of the virtual address to produce a 30-bit address as illustrated in *Figure 7-3: Accessing the translation table first level descriptors* on page 7-5. This address selects a four-byte translation table entry which is a First Level Descriptor for either a Section or a Page (bit1 of the descriptor returned specifies whether it is for a Section or Page).

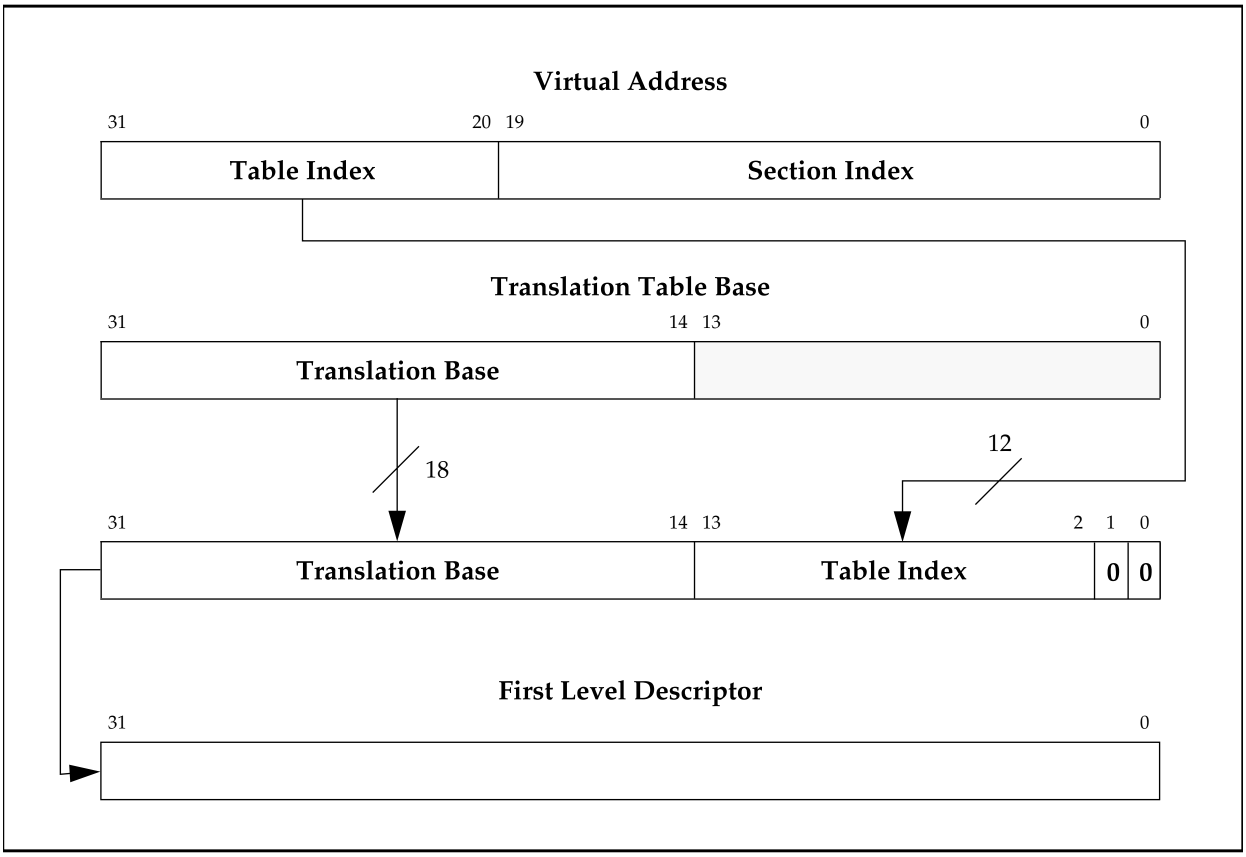


Figure 7-3: Accessing the translation table first level descriptors

7.4.3 Level one descriptor

The Level One Descriptor returned is either a Page Table Descriptor or a Section Descriptor, and its format varies accordingly. The following figure illustrates the format of Level One Descriptors.

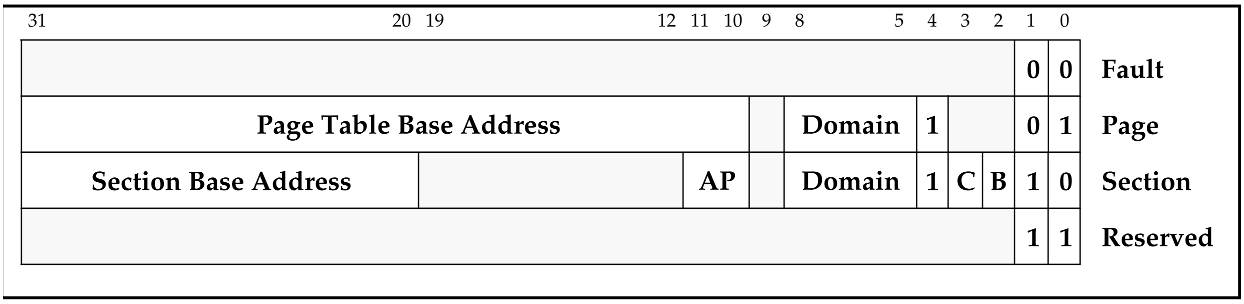


Figure 7-4: Level one descriptors

The two least significant bits indicate the descriptor type and validity, and are interpreted as shown below

Value	Meaning	Notes
0 0	Invalid	Generates a Section Translation Fault
0 1	Page	Indicates that this is a Page Descriptor
1 0	Section	Indicates that this is a Section Descriptor
1 1	Reserved	Reserved for future use

Table 7-1: Interpreting level one descriptor bits [1:0]

7.4.4 Page table descriptor

Bits 3:2 are always written as 0.

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the sixteen possible domains (held in the Domain Access Control Register) that contain the primary access controls.

Bits 31:10 form the base for referencing the Page Table Entry. (The page table index for the entry is derived from the virtual address as illustrated in *Figure 7-7: Small page translation* on page 7-10).

If a Page Table Descriptor is returned from the Level One fetch, a Level Two fetch is initiated as described below.

7.4.5 Section descriptor

Bits 3:2 (C, & B) control the cache- and write-buffer-related functions as follows:

C - Cacheable: indicates that data at this address will be placed in the cache (if the cache is enabled).

B - Bufferable: indicates that data at this address will be written through the write buffer (if the write buffer is enabled).

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the sixteen possible domains (held in the Domain Access Control Register) that contain the primary access controls.

Bits 11:10 (AP) specify the access permissions for this section and are interpreted as shown in *Table 7-2: Interpreting access permission (AP) bits* on page 7-7. Their interpretation is dependent upon the setting of the S and R bits (control register bits 8 and 9). Note that the Domain Access Control specifies the primary access control; the AP bits only have an effect in client mode. Refer to section on access permissions.

AP	S	R	Supervisor permissions	User permissions	Notes
00	0	0	No Access	No Access	Any access generates a permission fault
00	1	0	Read Only	No Access	Supervisor read only permitted
00	0	1	Read Only	Read Only	Any write generates a permission fault
00	1	1	Reserved		
01	x	x	Read/Write	No Access	Access allowed only in Supervisor mode
10	x	x	Read/Write	Read Only	Writes in User mode cause permission fault
11	x	x	Read/Write	Read/Write	All access types permitted in both modes.
xx	1	1	Reserved		

Table 7-2: Interpreting access permission (AP) bits

Bits 19:12 are always written as 0.

Bits 31:20 form the corresponding bits of the physical address for the 1MByte section.

7.5 Translating section references

➲*Figure 7-5: Section translation* illustrates the complete Section translation sequence. Note that the access permissions contained in the Level One Descriptor must be checked before the physical address is generated. The sequence for checking access permissions is described below.

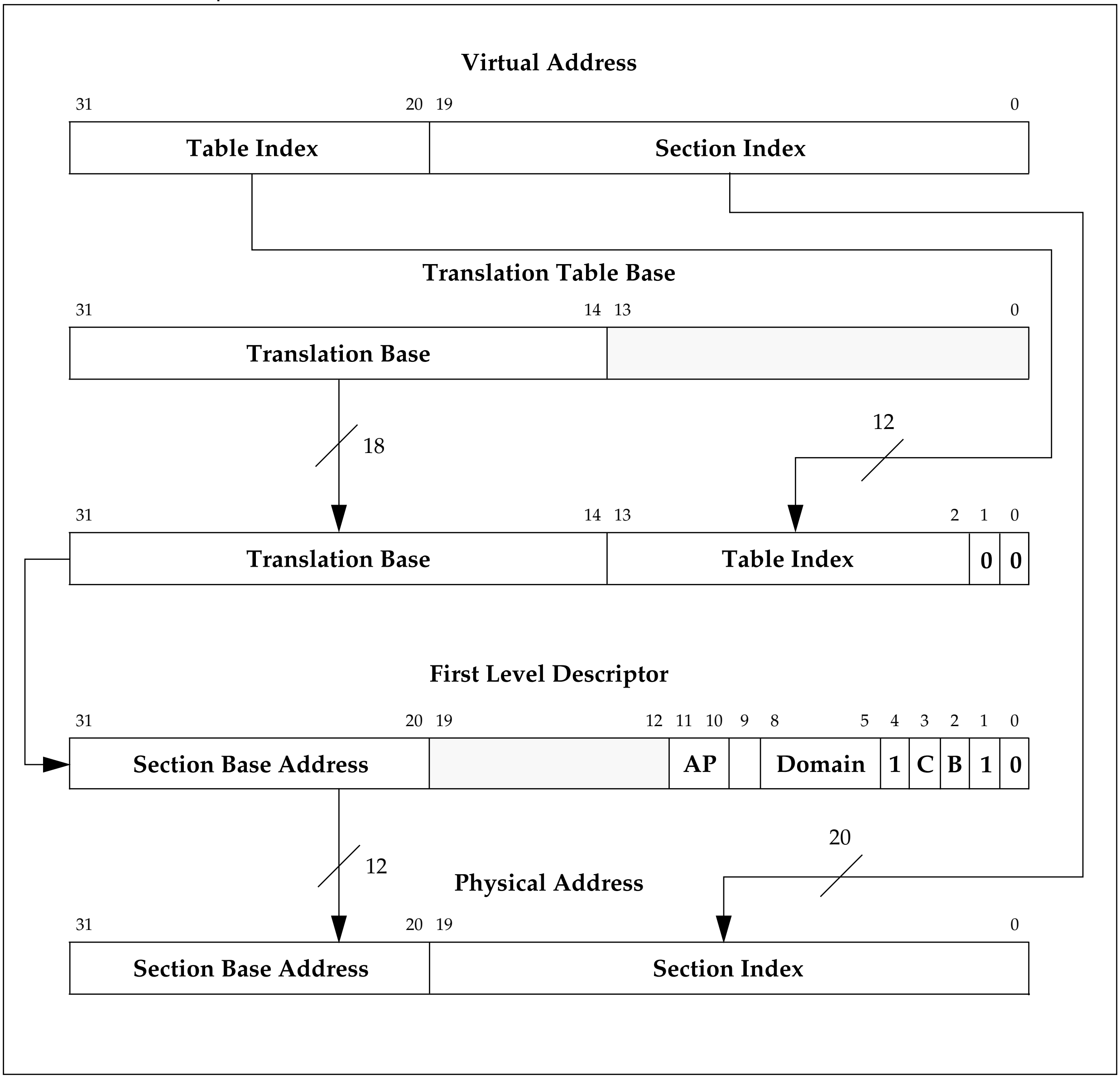


Figure 7-5: Section translation

7.5.1 Level two descriptor

If the Level One fetch returns a Page Table Descriptor, this provides the base address of the page table to be used. The page table is then accessed as described in *Figure 7-7: Small page translation*, and a Page Table Entry, or Level Two Descriptor, is returned. This in turn may define either a Small Page or a Large Page access. *Figure 7-6: Page table entry (level two descriptor)* on page 7-9 shows the format of Level Two Descriptors.

31 – 20	19 – 16	15 – 12	11 10	9 8	7 6	5 4	3	2	1	0	
									0	0	**Fault**
Large Page Base Address			**ap3**	**ap2**	**ap1**	**ap0**	**C**	**B**	0	1	**Large Page**
Small Page Base Address			**ap3**	**ap2**	**ap1**	**ap0**	**C**	**B**	1	0	**Small Page**
									1	1	**Reserved**

Figure 7-6: Page table entry (level two descriptor)

The two least significant bits indicate the page size and validity, and are interpreted as follows.

Value	Meaning	Notes
0 0	Invalid	Generates a Page Translation Fault
0 1	Large Page	Indicates that this is a 64 Kb Page
1 0	Small Page	Indicates that this is a 4 Kb Page
1 1	Reserved	Reserved for future use

Table 7-3: Interpreting page table entry bits 1:0

Bit 2 B - Bufferable: indicates that data at this address will be written through the write buffer (if the write buffer is enabled).

Bit 3 C - Cacheable: indicates that data at this address will be placed in the IDC (if the cache is enabled).

Bits 11:4 specify the access permissions (ap3 - ap0) for the four sub-pages and interpretation of these bits is described earlier in *Table 7-1: Interpreting level one descriptor bits [1:0]* on page 7-6.

For large pages, **bits 15:12** are programmed as 0.

Bits 31:12 (small pages) or bits **31:16** (large pages) are used to form the corresponding bits of the physical address - the physical page number. (The page index is derived from the virtual address as illustrated in *Figure 7-7: Small page translation* on page 7-10 and *Figure 7-8: Large page translation* on page 7-11).

7.6 Translating small page references

Figure 7-7: Small page translation illustrates the complete translation sequence for a 4kB Small Page. Page translation involves one additional step beyond that of a section translation: the Level One descriptor is the Page Table descriptor, and this is used to point to the Level Two descriptor, or Page Table Entry. (Note that the access permissions are now contained in the Level Two descriptor and must be checked before the physical address is generated. The sequence for checking access permissions is described later).

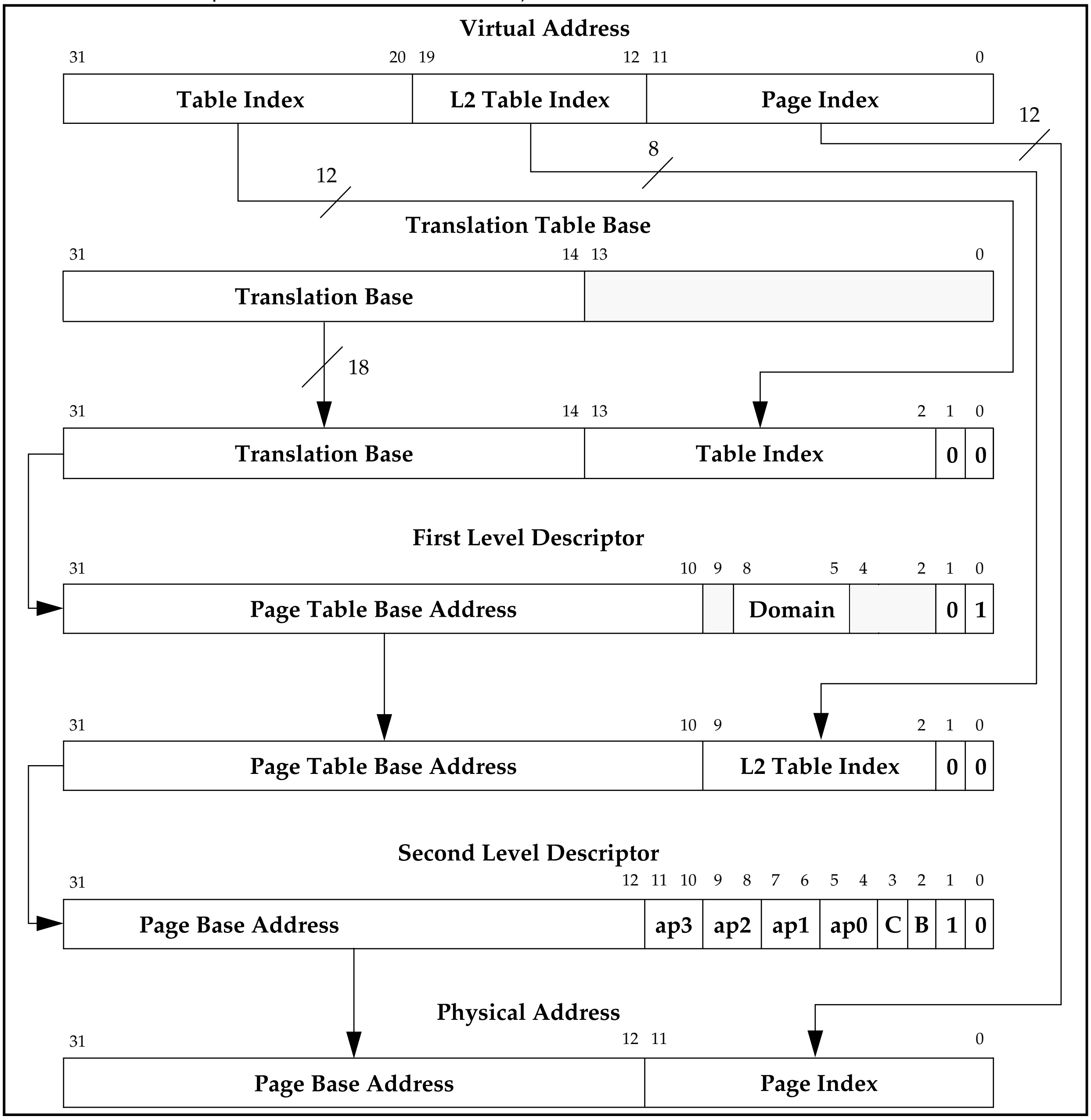


Figure 7-7: Small page translation

7.7 Translating large page references

Figure 7-8: Large page translation illustrates the complete translation sequence for a 64Kb Large Page. Note that since the upper four bits of the Page Index and low-order four bits of the Page Table index overlap, each Page Table Entry for a Large Page must be duplicated 16 times (in consecutive memory locations) in the Page Table.

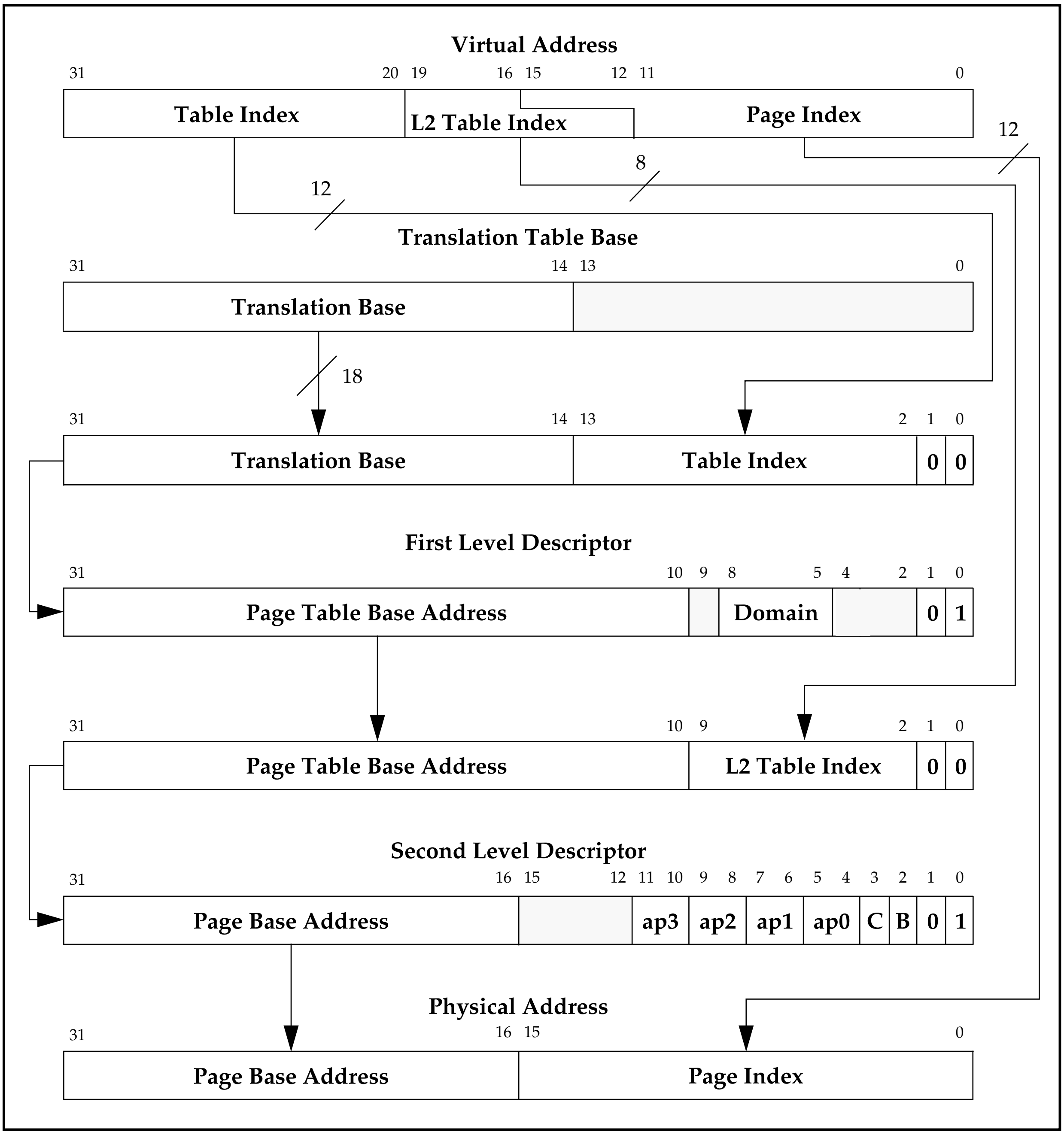


Figure 7-8: Large page translation

7.8 MMU faults and CPU aborts

The MMU generates four types of faults:

Alignment Fault

Translation Fault

Domain Fault

Permission Fault

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as the result of a memory access, the MMU will abort the access and signal the fault condition to the CPU. The MMU is also capable of retaining status and address information about the abort. The CPU recognises two types of abort: data aborts and prefetch aborts, and these are treated differently by the MMU.

If the MMU detects an access violation, it will do so before the external memory access takes place, and it will therefore inhibit the access.

7.9 Fault Address & Fault Status Registers (FAR & FSR)

Aborts resulting from data accesses (data aborts) are acted upon by the CPU immediately, and the MMU places an encoded 4 bit value FS[3:0], along with the 4 bit encoded Domain number, in the Fault Status Register (FSR). In addition, the virtual processor address which caused the data abort is latched into the Fault Address Register (FAR). If an access violation simultaneously generates more than one source of abort, they are encoded in the priority given in *Table 7-4: Priority encoding of fault status* on page 7-12.

CPU instructions on the other hand are prefetched, so a prefetch abort simply flags the instruction as it enters the instruction pipeline. Only when (and if) the instruction is executed does it cause an abort; an abort is not acted upon if the instruction is not used (i.e. it is branched around). Because instruction prefetch aborts may or may not be acted upon, the MMU status information is not preserved for the resulting CPU abort; for a prefetch abort, the MMU does not update the FSR or FAR.

The sections that follow describe the various access permissions and controls supported by the MMU and detail how these are interpreted to generate faults.

Priority	Source	FS[3210]	Domain [3:0]	FAR
Highest	Alignment	00x1	x	valid
	Translation (Section)	0101	Note 2	valid
	Translation (Page)	0111	valid	valid
	Domain (Section)	1001	valid	valid
	Domain (Page)	1011	valid	valid

Table 7-4: Priority encoding of fault status

Priority	Source	FS[3210]	Domain [3:0]	FAR
	Permission (Section)	1101	valid	valid
Lowest	Permission (Page)	1111	valid	valid

Table 7-4: Priority encoding of fault status (Continued)

x is undefined, and may read as 0 or 1

Notes: *Any abort masked by the priority encoding may be regenerated by fixing the primary abort and restarting the instruction.*
In fact this register will contain bits[8:5] of the Level 1 entry which are undefined, but would encode the domain in a valid entry.

7.10 Domain access control

MMU accesses are primarily controlled via domains. There are 16 domains, and each has a 2-bit field to define it. Two basic kinds of users are supported: Clients and Managers. Clients use a domain; Managers control the behaviour of the domain. The domains are defined in the Domain Access Control Register. *Figure 7-9: Domain access control register format* illustrates how the 32 bits of the register are allocated to define the sixteen 2-bit domains.

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 7-9: Domain access control register format

Table 7-5: Interpreting access bits in domain access control register defines how the bits within each domain are interpreted to specify the access permissions.

Value	Meaning	Notes
00	No Access	Any access will generate a Domain Fault.
01	Client	Accesses are checked against the access permission bits in the Section or Page descriptor.
10	Reserved	Reserved. Currently behaves like the no access mode.
11	Manager	Accesses are NOT checked against the access Permission bits so a Permission fault cannot be generated.

Table 7-5: Interpreting access bits in domain access control register

7.11 Fault checking sequence

The sequence by which the MMU checks for access faults is slightly different for Sections and Pages. The figure below illustrates the sequence for both types of accesses. The sections and figures that follow describe the conditions that generate each of the faults.

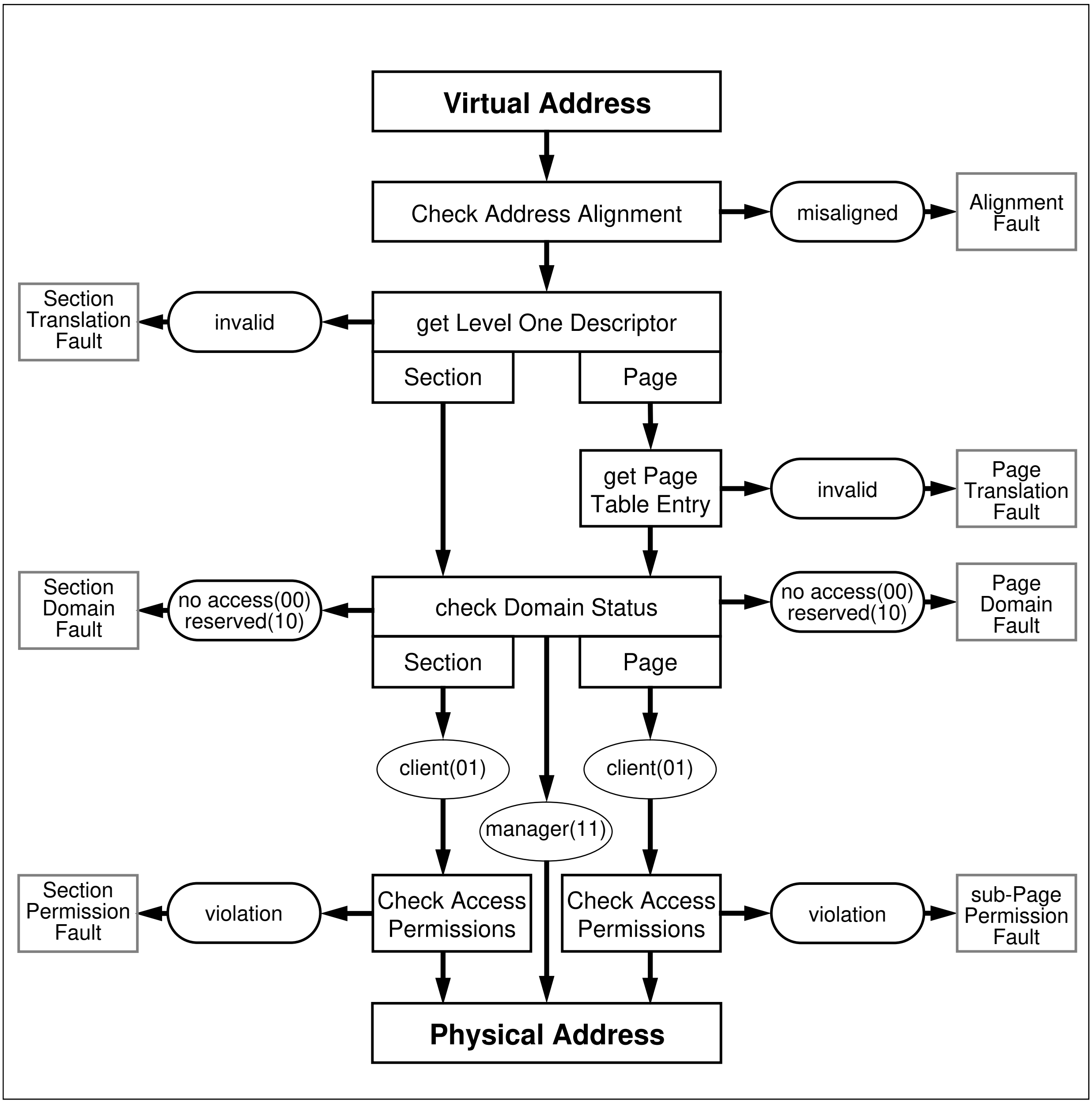


Figure 7-10: Sequence for checking faults

7.11.1 Alignment fault

If Alignment Fault is enabled (bit 1 in Control Register set), the MMU will generate an alignment fault on any data word access the address of which is not word-aligned irrespective of whether the MMU is enabled or not; in other words, if either of virtual address bits [1:0] are not 0. Alignment fault will not be generated on any instruction fetch, nor on any byte access. Note that if the access generates an alignment fault, the access sequence will abort without reference to further permission checks.

7.11.2 Translation fault

There are two types of translation fault: section and page.

1. A Section Translation Fault is generated if the Level One descriptor is marked as invalid. This happens if bits[1:0] of the descriptor are both 0 or both 1.
2. A Page Translation Fault is generated if the Page Table Entry is marked as invalid. This happens if bits[1:0] of the entry are both 0 or both 1.

7.11.3 Domain fault

There are two types of domain fault: section and page. In both cases the Level One descriptor holds the 4-bit Domain field which selects one of the sixteen 2-bit domains in the Domain Access Control Register. The two bits of the specified domain are then checked for access permissions as detailed in *Table 7-2: Interpreting access permission (AP) bits* on page 7-7. In the case of a section, the domain is checked once the Level One descriptor is returned, and in the case of a page, the domain is checked once the Page Table Entry is returned.

If the specified access is either No Access (00) or Reserved (10) then either a Section Domain Fault or Page Domain Fault occurs.

7.11.4 Permission fault

There are two types of permission fault: section and sub-page. Permission fault is checked at the same time as Domain fault. If the 2-bit domain field returns client (01), then the permission access check is invoked as follows:

Section

If the Level One descriptor defines a section-mapped access, then the AP bits of the descriptor define whether or not the access is allowed according to *Table 7-2: Interpreting access permission (AP) bits* on page 7-7. Their interpretation is dependent upon the setting of the S bit (Control Register bit 8). If the access is not allowed, then a Section Permission fault is generated.

Sub-page

If the Level One descriptor defines a page-mapped access, then the Level Two descriptor specifies four access permission fields (ap3..ap0) each corresponding to one quarter of the page. Hence for small pages, ap3 is selected by the top 1Kb of the page, and ap0 is selected by the bottom 1kB of the page; for large pages, ap3 is selected by the top 16Kb of the page, and ap0 is selected by the bottom 16Kb of the page. The selected AP bits are then interpreted in exactly the same way as for a section (see *Table 7-2: Interpreting access permission (AP) bits* on page 7-7), the only difference being that the fault generated is a sub-page permission fault.

7.12 External aborts

The ARM7500 does not support external aborts.

7.12.1 Interaction of the MMU, IDC and write buffer

The MMU, IDC and WB may be enabled/disabled independently. However there are only five valid combinations. There are no hardware interlocks on these restrictions, so invalid combinations will cause undefined results.

MMU	IDC	WB
off	off	off
on	off	off
on	on	off
on	off	on
on	on	on

Table 7-6: Valid MMU, IDC, and WB combinations

The following procedures must be observed.

To enable the MMU:

1. Program the Translation Table Base and Domain Access Control Registers
2. Program Level 1 and Level 2 page tables as required
3. Enable the MMU by setting bit 0 in the Control Register.

Note: *Care must be taken if the translated address differs from the untranslated address as the two instructions following the enabling of the MMU will have been fetched using "flat translation" and enabling the MMU may be considered as a branch with delayed execution. A similar situation occurs when the MMU is disabled. Consider the following code sequence:*

```
MOV             R1, #0x1
MCR             15,0,R1,0,0    ; Enable MMU
Fetch Flat
Fetch Flat
Fetch Translated
```

To disable the MMU

1. Disable the WB by clearing bit 3 in the Control Register.
2. Disable the IDC by clearing bit 2 in the Control Register.
3. Disable the MMU by clearing bit 0 in the Control Register.

Note: *If the MMU is enabled, then disabled and subsequently re-enabled the contents of the TLB will have been preserved. If these are now invalid, the TLB should be flushed before re-enabling the MMU.*

Disabling of all three functions may be done simultaneously.

7.13 Effect of reset

See *Chapter 4: ARM Processor Programmer's Model* .

8 The Video and Sound Macrocell

This chapter introduces the ARM7500 video and sound system.

8.1 Introduction

The ARM7500 single chip computer contains a high performance video and sound controller, capable of meeting the requirements of a wide range of configurations.

The video and sound macrocell handles all the video processing aspects of the ARM7500 functionality, making the ARM7500 suitable for incorporation into a wide range of end products ranging from portable hand-held LCD systems through to higher performance SuperVGA desktop products.

The flexible bus interface provides hardware support for interfacing to DRAM memory systems in conjunction with the ARM7500 memory controller. The video and sound macrocell obtains data from external DRAM under DMA control. The macrocell also incorporates two stereo sound systems - an 8-bit (logarithmic) system, featuring up to eight channels, each with its own stereo position, and an serial sound output port suitable for connection to an external CD DAC.

Features include:

- VGA, SuperVGA, XGA resolution
- three 8-bit DACs giving 16M colours
- direct driving of LCD or CRT screens
- 1, 2, 4, 8, 16, 32 bits per pixel modes
- up to 120MHz pixel rate
- very low power consumption

8.2 Features

8.2.1 Flexible video system

The video and sound macrocell contains 296 write-only registers which offer a high degree of flexibility to the system programmer. 256 of these are used as the 28-bit video palette entries. These are programmed via an auto-incrementing address pointer. The remaining registers are specific control registers and allow the user to program the display parameters.

8.2.2 Hardware cursor

The video and sound macrocell has a hardware cursor for all its display modes - Normal, Hi-Res, and LCD. By offering cursor support on chip the designer benefits in terms of speed and lower software overhead. The cursor is 32 pixels wide and any number of pixels high and can be displayed in 4 colours including transparent from its own 28-bit wide palette. In this way a cursor of any shape and size can be defined within the 32-pixel wide limit.

8.2.3 Palette

The video subsystem has a 28-bit wide 256-entry palette where each entry uses 8 bits for Red, 8 for Green and 8 for Blue, and 4 bits for external data. These external bits may be used outside the chip for a variety of purposes such as supremacy, fading, Hi-Res and LCD driving.

Look Up Tables (LUT) allow for logical to physical translation and gamma correction.The Red Green and Blue LUTs each drive their respective DACs, and the Ext LUT is normally configured to drive the 4-bit output port.

There are three 8-bit linear monotonic DACs (Red, Green and Blue) which give a total of 16M possible colours. The DACs are designed to operate up to 120 MHz and drive doubly-terminated 75Ω lines directly.

8.2.4 Pixel clock

The ARM7500 is capable of generating a display at any pixel rate up to 120MHz. The pixel clock may be selected from one of 3 sources, and then the selected frequency of this clock may be further divided down by a factor of between 1 and 8.

The video and sound macrocell contains an on-chip phase comparator which, when used in conjunction with an external Voltage Controlled Oscillator (VCO), forms a Phase Locked Loop. This configuration allows a single reference clock to generate all the required frequencies for any display mode thus obviating the need for multiple external crystals.

8.2.5 Display modes

Irrespective of the memory configuration used, the video subsystem is capable of many different display formats. In addition to the normal linear CRT display, the video subsystem can generate a display suitable for either very high resolution displays, single or dual-panel LCDs.

For CRT displays, the video and sound macrocell is capable of operating in a variety of pixel modes - 1,2,4,8,16,32 bits/pixel, and can also directly drive LCD displays in 1,2 or 4 bits per pixel via an internal 16-level grey scaler. The grey scaler algorithm adopted is patented.

8.2.6 Power management

The macrocell is designed for power sensitive applications and incorporates design features to minimise power consumption. A *power down mode* allows power savings to be made when the device is not in use, for example, in conjunction with a battery powered LCD system. Additional power sensitive features include the powering down of functions of the device currently not in use, such as the video DACs, sound DACs and the LCD grey scaler. In addition the palette design has been segmented such that only one eighth of the palette is enabled and clocked at any one time. The power-down mode can be used in conjunction with the ARM7500's STOP mode to ensure minimum power consumption when clocks are stopped.

8.2.7 On-chip sound system

The ARM7500 supports two systems - an 8-bit (logarithmic) system using an internal dedicated DAC, featuring up to eight channels each with its own stereo position, and a 32-bit serial sound output suitable for driving external CD DACs.

In the 8-bit mode the device can work with 1,2,4 or 8 stereo channels, using time division multiplexing to synthesise left and right outputs. The sample rate is programmable through the Sound Frequency Register.

Enhanced 32-bit stereo sound is offered by the serial sound output which consists of a three pin serial interface. Each 32-bit sample consists of 16 bits for the left channel and 16 bits for the right channel.

8.3 Block diagram

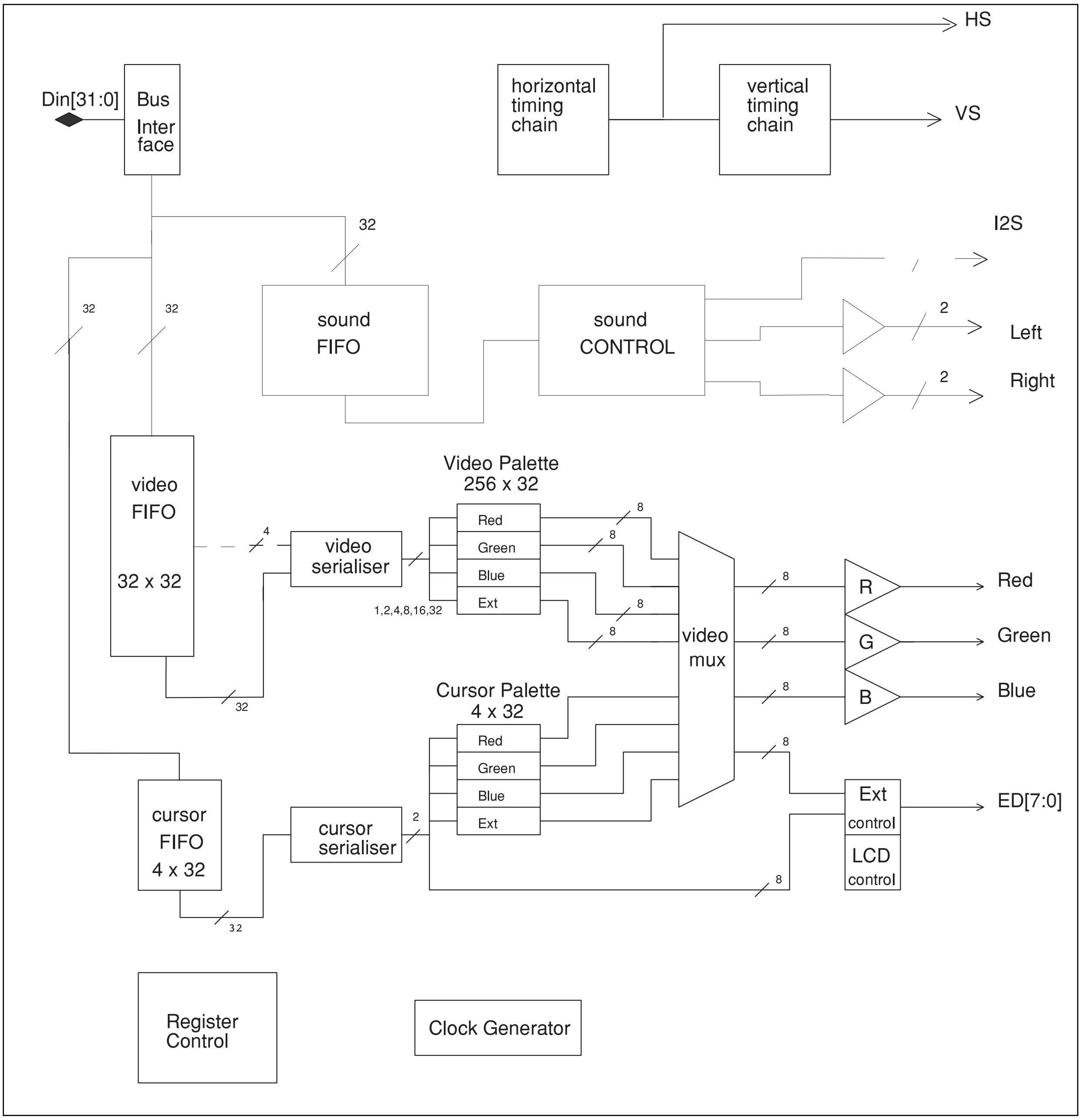


Figure 8-1: video and sound macrocell block diagram

9 Video and Sound Programmer's Model

This chapter details the video and sound macrocell programmable registers.

9.1 The video and sound macrocell registers

The video and sound macrocell contains 296 write-only registers. These are split into 2 categories; the 256 28-bit video palette entries, and the remaining control registers. The video palette entries are written via an auto-incrementing address pointer. All the other registers (including the 28-bit cursor palette) are written directly with the address encoded in the top 4 or 8 bits of the data word. To program the registers, the ARM7500 address bus should be set to between 0x03400000 and 0x034FFFFF, and the data word written should include the individual register address in the upper 4 or 8 bits, as appropriate.

In order to define the display format correctly, eleven registers need to be programmed as shown in the diagram below:

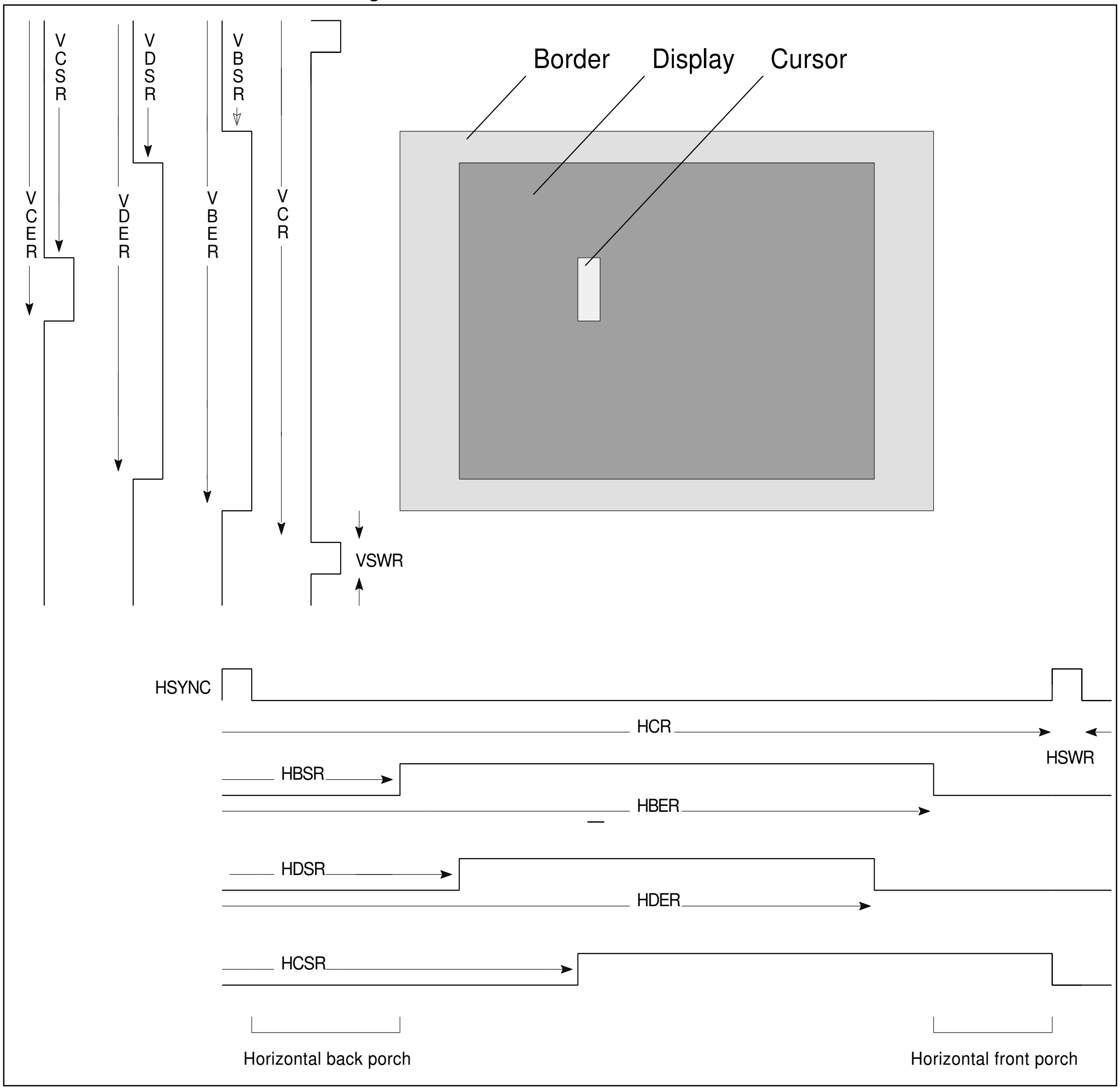


Figure 9-1: The video and sound macrocell display format definitions

The register allocation is shown in the following table. An x denotes the actual data field, and any unused bit should be programmed with a logic zero. Do not access any register at any location other than that shown as the actual register map is multiple-mapped.

The External Register, Control Register, Sound Control Register and Data Control Register all contain bits that are not initialised at power up, and so must be programmed before the video and sound macrocell will operate correctly.

Address(hex)	Register
0xxxxxxx	Video Palette
100000xx	Video Palette Address Register
20000000	RESERVED
300000xx	LCD Offset register 0
310000xx	LCD offset register 1
4xxxxxxx	Border Colour Register
5xxxxxxx	Cursor Palette logical colour 1
6xxxxxxx	Cursor Palette logical colour 2
7xxxxxxx	Cursor Palette logical colour 3
8000xxxx	Horizontal Cycle Register
8100xxxx	Horizontal Sync Width Register
8200xxxx	Horizontal Border Start Register
8300xxxx	Horizontal Display Start Register
8400xxxx	Horizontal Display End Register
8500xxxx	Horizontal Border End Register
8600xxxx	Horizontal Cursor Start Register
8700xxxx	Reserved
8800xxxx	Test Register
8C00xxxx	Test Register
9000xxxx	Vertical Cycle Register
9100xxxx	Vertical Sync Width Register
9200xxxx	Vertical Border Start Register
9300xxxx	Vertical Display Start Register
9400xxxx	Vertical Display End Register

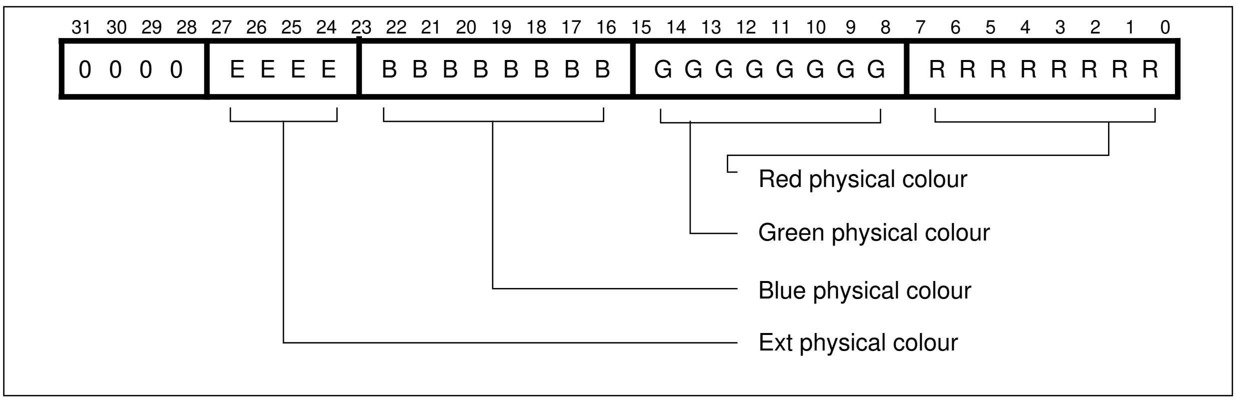
Table 9-1: The video and sound macrocell register allocation

Address(hex)	Register
9500xxxx	Vertical Border End Register
9600xxxx	Vertical Cursor Start Register
9700xxxx	Vertical Cursor End Register
9800xxxx	Test Register
9A00xxxx	Test Register
9C00xxxx	Test Register
A000000x}	Stereo Image Registers
A700000x}	
B00000x	Sound Frequency Generator
B10000x	Sound Control Register
C00xxxxx	External Register
D000xxxx	Frequency Synthesis Register
E00xxxxx	Control Register
F000xxxx	Data Control Register

Table 9-1: The video and sound macrocell register allocation (Continued)

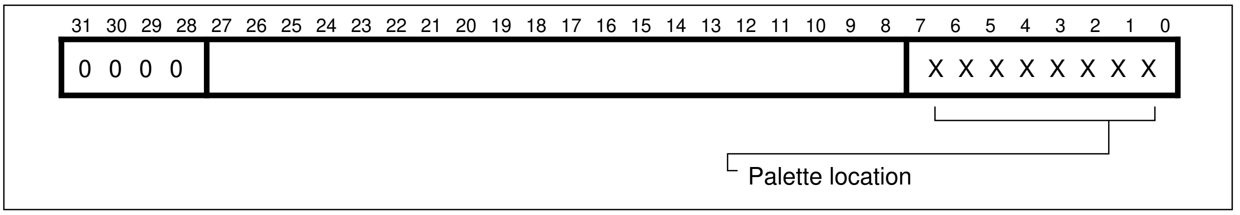
9.2 Video palette: Address 0x0

All entries of the video palette are written at address 0. In order to write any or all of the palette locations, the address pointer must first be written, as described below. The palette is programmed with a 28-bit word representing the physical data field.



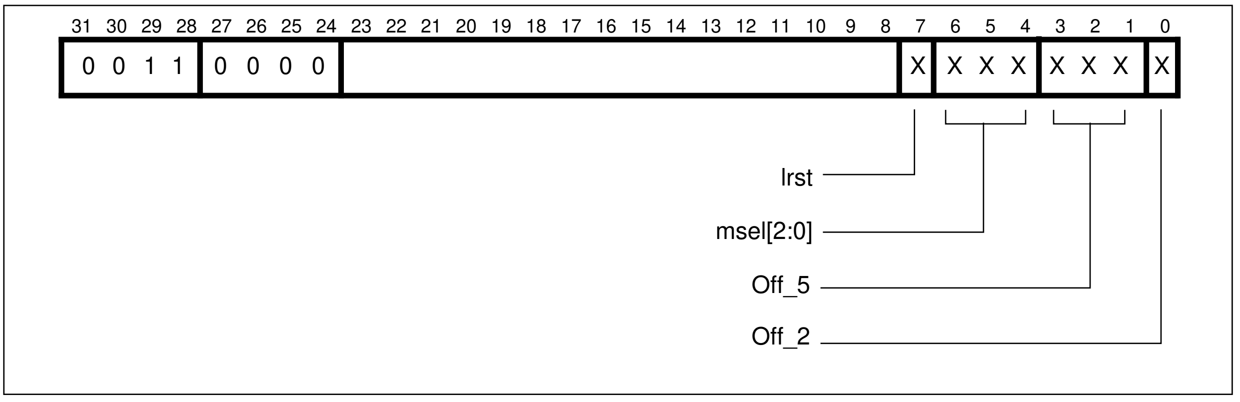
9.3 Video palette address pointer: Address 0x1

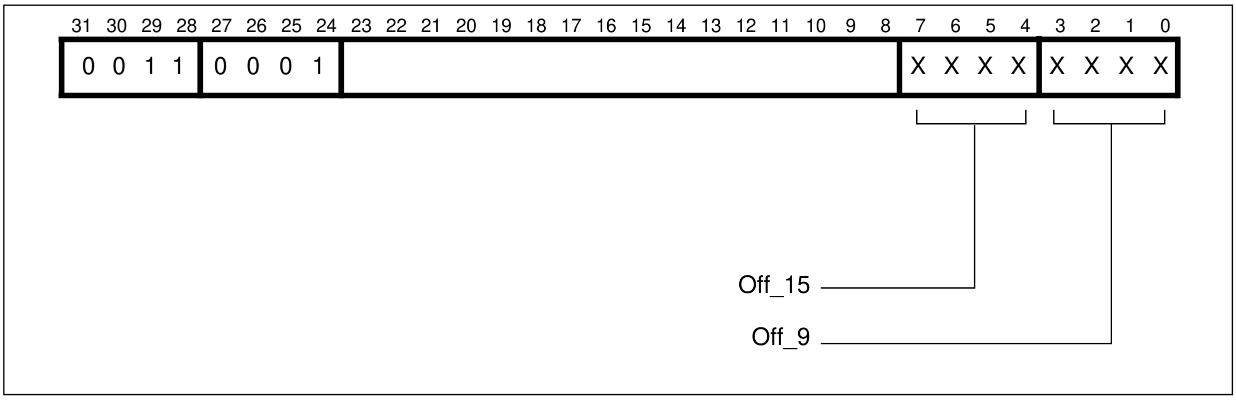
The address pointer is programmed at address 1, and it may be programmed to any value from 0 to 255. The first write to the palette will then occur at this location, and the address pointer will post-increment so that the next palette write will occur to the following location. The counter will wrap around from 255 to 0. Once the address pointer has been written, any number of palette locations can be programmed, and the pointer can be reprogrammed at any time if only part of the whole palette is to be updated.



9.4 LCD offset registers: Addresses 0x30 and 0x31

These two, 8-bit registers define the offsets required for driving a dual panel LCD screen. Register 0 defines the offsets for the five and two frame duty cycle grey scales, as well as reset and test mode bits. Register 1 defines the offsets for the nine and fifteen frame duty cycle grey scales.





The registers values are dependent upon the size of the LCD screen to be driven, and are calculated in the following way:

Off_15 = (3xL + 8) mod 15
Off_9 = (7xL + 4) mod 9
Off_5 = (1xL + 3) mod 5
Off_2 = 0

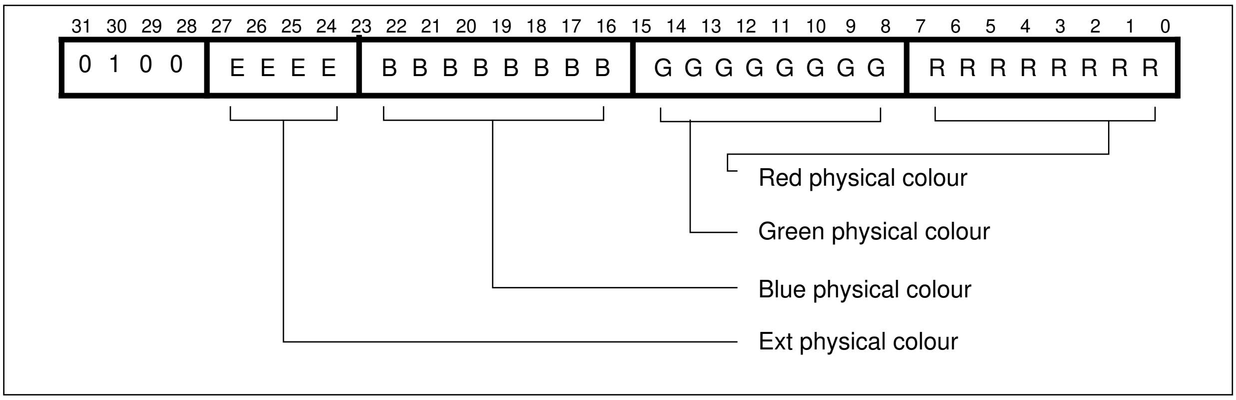
Where L is the number of lines in the upper panel of the dual panel LCD screen.

Bits 7-4 of register 0 are only used in test mode, and must all be set to zero in normal operation.

msel[2:0] are test bits and should be programmed LOW.

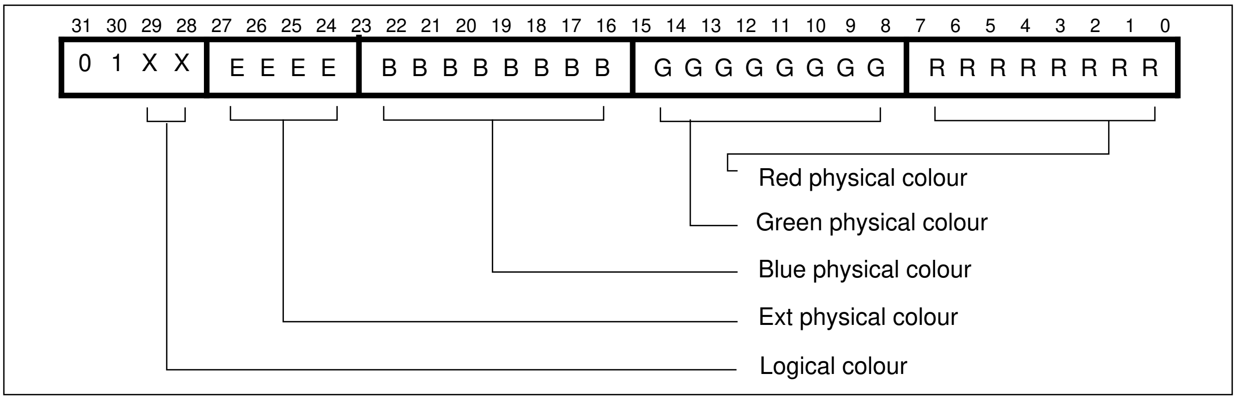
9.5 Border colour register: Address 0x4

This register defines the physical border colour, and is programmed with a 28-bit word. Note that this register is programmed directly, independent of the value of the video palette address pointer.



9.6 Cursor palette: Addresses 0x5-0x7

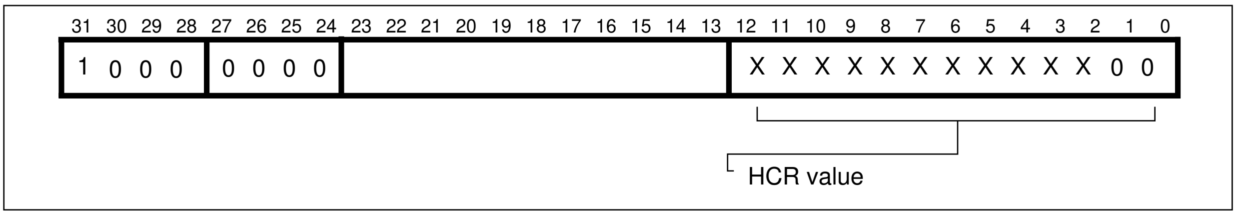
These three registers are programmed with the physical colour of the three logical cursor colours. Note that cursor logical colour 00 is defined as being transparent (i.e. no cursor display), and its location is used for the Border Colour Register above.



9.7 Horizontal cycle register (HCR): Address 0x80

This register defines the period, in pixels, of the horizontal scan, i.e. display time + retrace time.

This is a 14-bit register of which the bottom 2 bits must be programmed to 0. If N pixels are required in the horizontal scan period, then value (N-8) should be programmed into the HCR. (N must be a multiple of 4).



9.8 Horizontal sync width register (HSWR): Address 0x81

This register defines the period, in pixels, of the **HSYNC** pulse.

This is a 14-bit register of which the bottom bit must be programmed to 0. If N pixels are required in the **HSYNC** pulse, then value (N-8) should be programmed into the HSWR. (N must be a multiple of 2).

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0	0 0 0 1		X X X X X X X X X X X X 0
			HSWR value

9.9 Horizontal border start register (HBSR): Address 0x82

This register defines the time, in pixels, from the start of the **HSYNC** pulse to the start of the border display.

This is a 14-bit register of which the bottom bit must be programmed to 0. If N pixels are required in this time, then value (N-12) should be programmed into the HBSR. (N must be a multiple of 2).

Note that this register must always be programmed, even when a border is not required. If a border is not required, then the value in the HBSR must be such as to start the border in the same place as the display start. i.e. $N_{HBSR}= N_{HDSR}$.

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0	0 0 1 0		X X X X X X X X X X X X 0
			HBSR value

9.10 Horizontal display start register (HDSR): Address 0x83

This register defines the time, in pixels, from the start of the **HSYNC** pulse to the start of the video display.

This is a 14-bit register of which the bottom bit must be programmed to 0. If N pixels are required in this time, then value (N-18) should be programmed into the HBSR. (N must be a multiple of 2).

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0	0 0 1 1		X X X X X X X X X X X X 0
			HDSR value

9.11 Horizontal display end register (HDER): Address 0x84

This register defines the time, in pixels, from the start of the **HSYNC** pulse to the end of the video display. (i.e. the first pixel which is not display).

This is a 14-bit register of which the bottom bit must be programmed to 0. If N pixels are required in this time, then value (N-18) should be programmed into the HBER. (N must be a multiple of 2)

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0	0 1 0 0		X X X X X X X X X X X X 0
			HDER value

9.12 Horizontal border end register (HBER): Address 0x85

This register defines the time, in pixels, from the start of the **HSYNC** pulse to the end of the border display. (i.e. the first pixel which is not border).

This is a 14-bit register of which the bottom bit must be programmed to 0. If N pixels are required in this time, then value (N-12) should be programmed into the HBER. (N must be a multiple of 2).

Again, if no border is required, this register must still be programmed such that $N_{HBER} = N_{HDER}$.

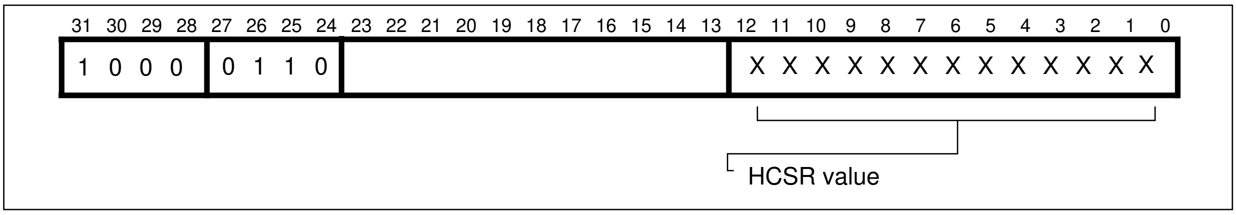
31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0	0 1 0 1		X X X X X X X X X X X X 0
			HBER value

9.13 Horizontal cursor start register (HCSR): Address 0x86

This register defines the time, in pixels, from the start of the **HSYNC** pulse to the start of the cursor display.

This is a 14-bit register of which all bits may be programmed. If N pixels are required in this time, then value (N-17) should be programmed into the HCSR. The cursor can thus be programmed to start on any pixel. In HiRes mode, the cursor can still only be programmed to start on a normal pixel boundary. However, because the resolution of the cursor can be defined to a micro-pixel, by using different cursor images it is possible to position the cursor to any micro-pixel.

Note that only the cursor start position needs to be defined, as the cursor is automatically disabled after 32 pixels in normal mode, or 16 pixels in HiRes mode. If a cursor smaller than this is required, then the remaining bits in the cursor pattern should be programmed to logical colour 00 (transparent).



9.14 Horizontal interlace register (HIR): Address 0x87

Address 87H is reserved. Do not attempt to program this register.

9.15 Horizontal test registers: Addresses 0x88 & 0x8H

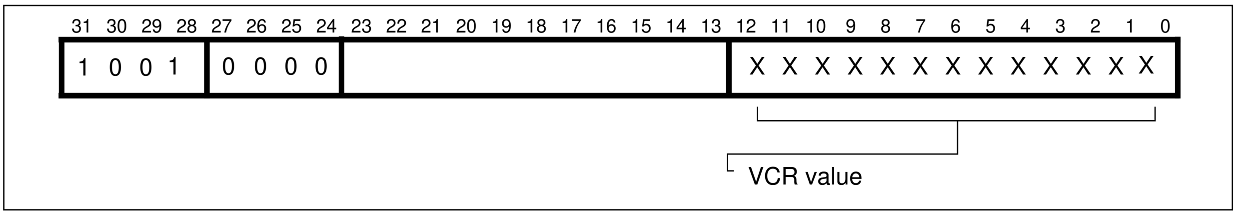
Two registers are provided for testing the chip in production. Neither of these registers are intended to be used during normal operation of the device.

9.16 Vertical cycle register (VCR): Address 0x90

This 13-bit register defines the period, in units of a raster, of the vertical scan. i.e. display time + flyback time.

If N rasters are required in a complete frame, then value (N-2) should be programmed into the VCR.

If an interlaced display is selected, (N-3)/2 must be programmed into the VCR. [N must be odd]. Here N is still the number of rasters in a complete frame, *not* a field.



9.17 Vertical sync width register (VSWR): Address 0x91

This 13-bit register defines the width, in units of a raster, of the **VSYNC** pulse.

If N rasters are required in the **VSYNC** pulse, then value (N - 2) should be programmed into the VSWR. The minimum value allowed for N is 2.

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 1	0 0 0 1		X X X X X X X X X X X X X

VSWR value

9.18 Vertical border start register (VBSR): Address 0x92

This 13-bit register defines the time, in units of a raster, from the start of the **VSYNC** pulse to the start of the border display.

If N rasters are required in this time, then value (N-1) should be programmed into the VBSR.

If no border is required, this register must still be programmed, in this case to the same value as the VDSR.

31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 1	0 0 1 0		X X X X X X X X X X X X X

VBSR value

9.19 Vertical display start register (VDSR): Address 0x93

This 13-bit register defines the time, in units of a raster, from the start of the **VSYNC** pulse to the start of the video display.

If N rasters are required in this time, then value (N-1) should be programmed into the VDSR.

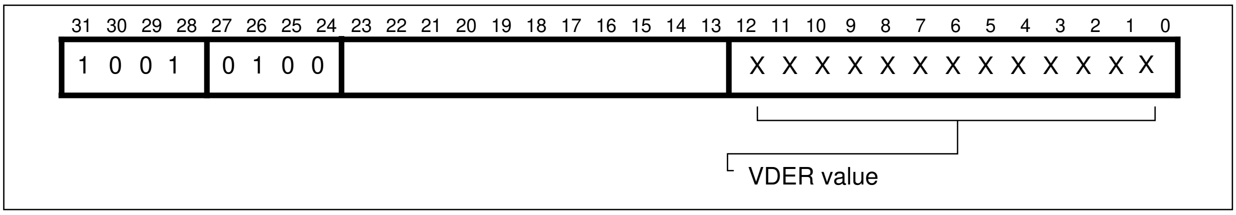
31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 1	0 0 1 1		X X X X X X X X X X X X X

VDSR value

9.20 Vertical display end register (VDER): Address 0x94

This 13-bit register defines the time, in units of a raster, from the start of the **VSYNC** pulse to the end of the video display. (i.e. the first raster on which the display is *not* present).

If N rasters are required in this time, then value (N-1) should be programmed into the VDER.

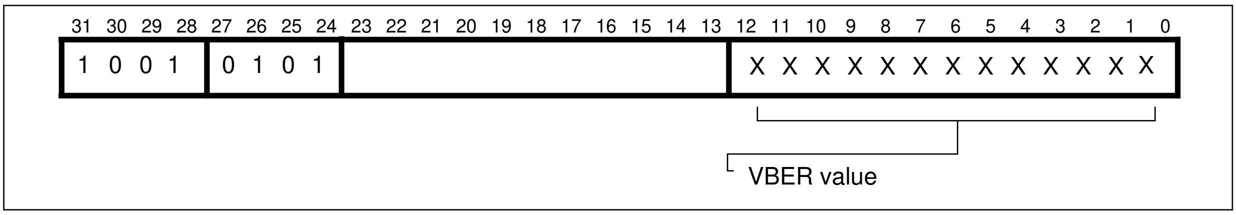


9.21 Vertical border end register (VBER): Address 0x95

This 13-bit register defines the time, in units of a raster, from the start of the **VSYNC** pulse to the end of the border display. (i.e. the first raster on which the border is not present).

If N rasters are required in this time, then value (N-1) should be programmed into the VBER.

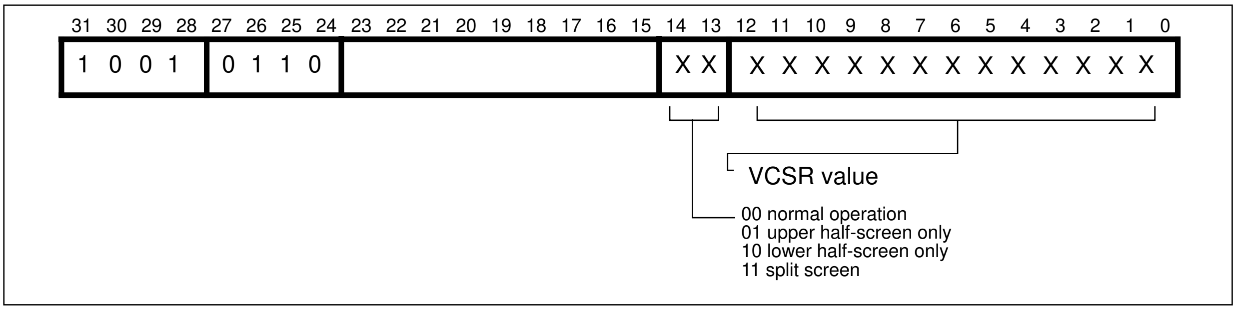
If no border is required, then this register must be programmed to the same value as the VDER.



9.22 Vertical cursor start register (VCSR): Address 0x96

This is a 15-bit register. The lower 13 bits define the time, in units of a raster, from the start of the **VSYNC** pulse to the start of the cursor display. If N rasters are required in this time, then value (N-1) should be programmed into the VCSR. The upper 2 bits are used to control the display of the cursor in duplex LCD mode. They should be programmed to zero in all other modes.

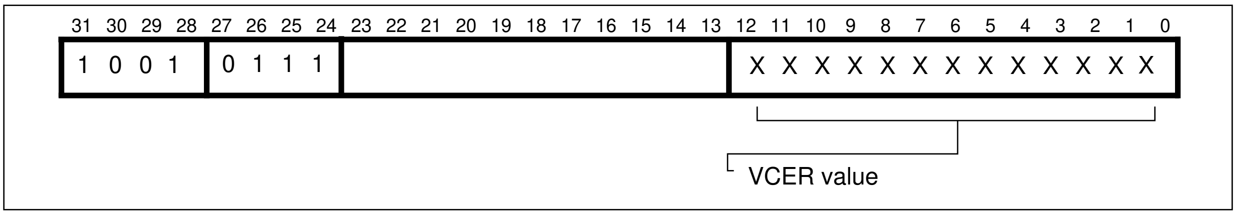
When the upper 2 bits are programmed to be 11 (split screen) the meaning of VCSR and VCER are altered as follows. The cursor is displayed in the lower half-screen only from the value of VDSR to the value of VCSR, and again in the upper half screen only from the value of VCER to the value of VDER. This allows a cursor to be positioned across the boundary of the upper and lower half screens of an LCD.



9.23 Vertical cursor end register (VCER): Address 0x97

This 13-bit register defines the time, in units of a raster, from the start of the **VSYNC** pulse to the end of the cursor display. (i.e. the first raster on which the cursor is not present).

If N rasters are required in this time, then value (N-1) should be programmed into the VCER.

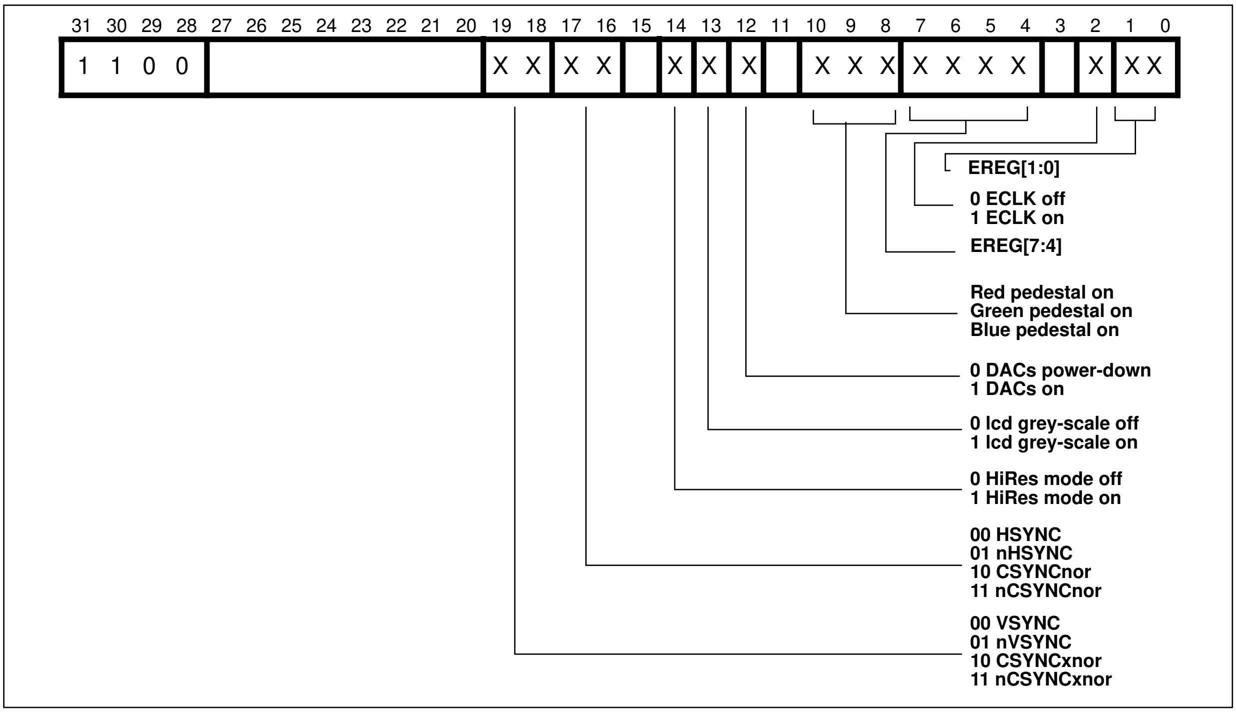


9.24 Vertical test registers: Addresses 0x98, 0x9A & 0x9C

Three registers are provided for testing the chip in production. None of these registers are intended to be used during normal operation of the device.

9.25 External register (ereg): Address 0xC

This register contains the control bits for the external functions of video and sound macrocell. In particular it controls the DACs, the configuration of the External Port **ED[7:0]**, and the configuration of the sync lines.



EREG[1:0] are internally mapped to drive esel[1:0] by ARM7500.

EREG[7:4] are exported from the chip on **ED[7:4]** if EREG[1:0]**=**3. Refer to *11.6 External support* on page 11-9.

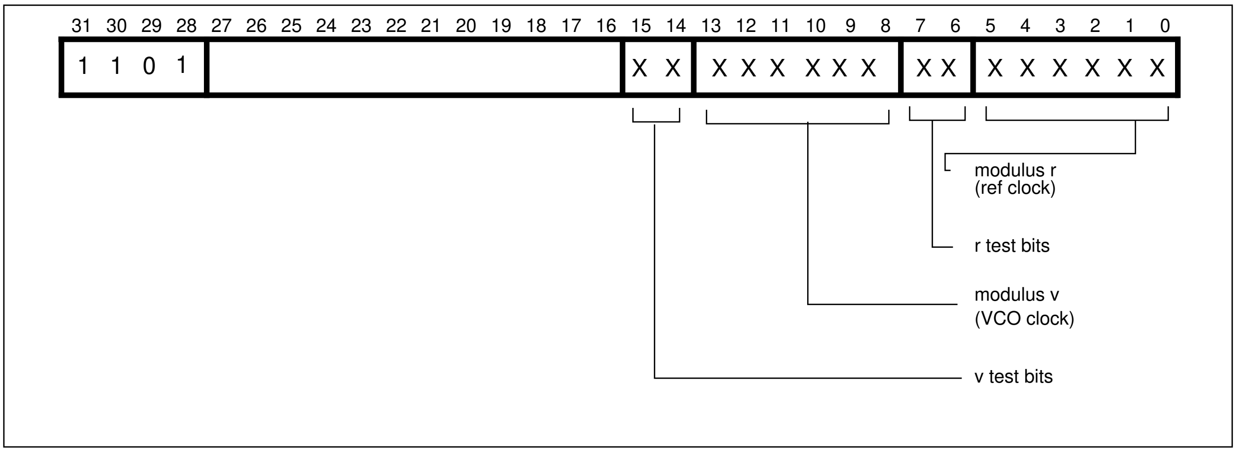
The use of pedon[2:0] and DAC is defined in *11.7 Analog outputs* on page 11-11.

The uses of lcd and hrm are defined in *11.6 External support* on page 11-9.

ARM7500 can export a variety of sync configurations on the pins **HSYNC** and **VSYNC**, as specified by the bits 16-17 and 18-19 respectively. For further explanation see *11.6.3 Vertical and horizontal synchronisation* on page 11-11.

9.26 Frequency synthesizer register (fsynreg): Address 0xD

ARM7500 is able to drive a VCO to provide a suitable input frequency for the pixel clock derived from a reference clock. This is achieved by dividing the reference clock by modulus r, and the VCO clock by modulus v, and comparing the resulting frequencies. Refer to *11.1 Pixel clock* on page 11-2 for a more detailed explanation. The two moduli, r and v are each 6-bit values, and are programmed in this register.



Associated with each counter are 2 test bits which should normally be programmed to 0.

Setting bit[6] forces the phase comparator HIGH, which drives **PCOMP** HIGH.

Setting bit[7] clears the r-modulus counter.

Setting bit[14] forces the phase comparator LOW, which drives **PCOMP** LOW.

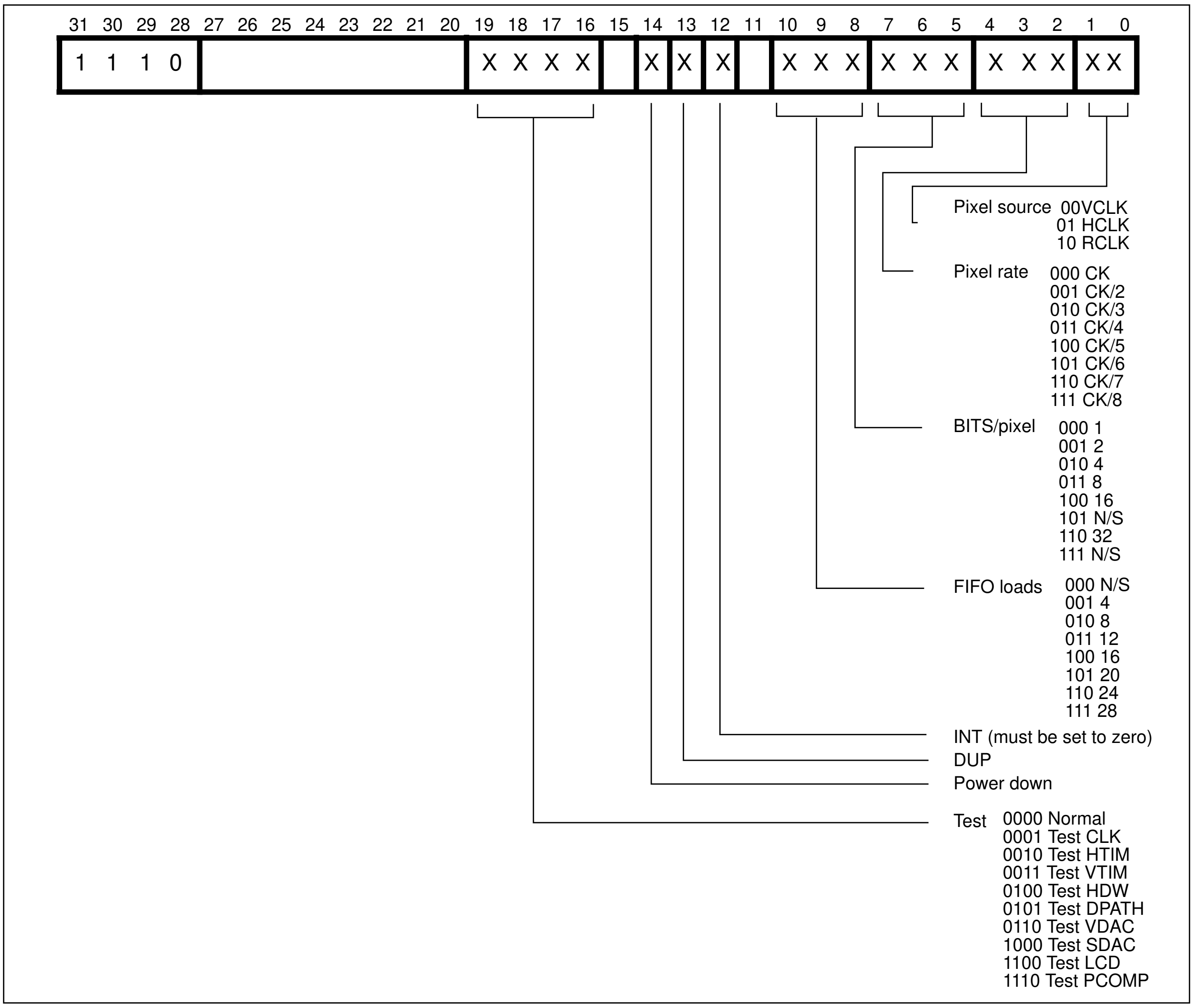
Setting bit[15] clears the v-modulus counter.

To reduce power consumption, this register should be programmed with large values when the frequency synthesizer is not in use. In particular, bits [6] and [14] should not be set at the same time.

To get a modulus of r, then value (r-1) should be programmed into the fsynreg. Likewise for the v-modulus.

9.27 Control register (conreg): Address 0xE

The main control register determines the basic operation of the chip. In particular the pixel clock source, the pixel rate, the number of bits/pixel, the control of the video FIFO, and the data format are programmed. In addition there is a 4-bit test register which must be programmed to zero for normal operation.



Note *The INT bit should always be set to zero.*

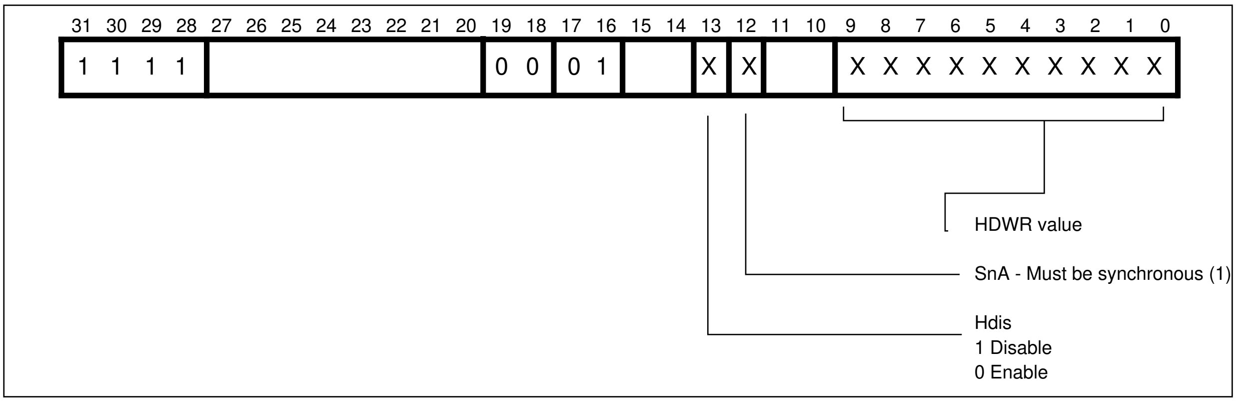
The pixel clock (pixclk) is selected from one of 3 sources, corresponding to the respective input pins, and the selected clock is then fed through a prescaler as defined by the 3 bits conreg[4:2]. The output of this prescaler is the actual pixel clock. See *Chapter 11: Video Features* for more detail.

The Video FIFO can be programmed to have any number of quad words loaded into it at the start of display. The value chosen should take into account the bandwidth of the display as well as the latency of the DMA subsystem. Refer to *Chapter 10: Video Macrocell Interface* before programming these values.

Setting the dup bit configures the display for dual-panel LCDs. This is described further in *Chapter 11: Video Features* .

Note that after a reset the Control Register should be the first register programmed. The Power Down bit (14) must immediately be programmed LOW. The test registers bits (16 to 19) also should be programmed LOW, as any other setting will inhibit normal operation.

9.28 Data control register (DCTL): Address 0xF

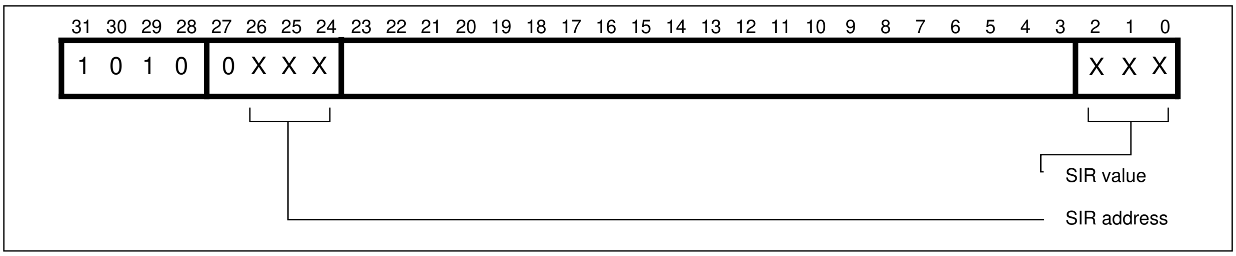


The horizontal display width is also defined in this register, and should be programmed to be the number of words of data in a displayed raster. It must be programmed in most configurations of the device, as it inhibits a DMA request near the end of a raster, when there are enough words in the video FIFO for that raster. The request is uninhibited after the **HSYNC** at the start of the next raster. When driving a dual panel LCD screen, this register must be programmed with twice the number of words in a displayed raster. Hdis should normally be programmed to zero. If Hdis is programmed to one, the inhibition of DMA requests is disabled.

Note *Bits 19:16 MUST be set to 0001 (binary).*

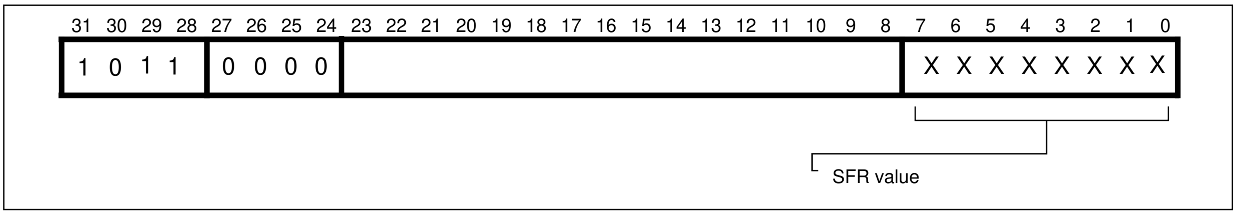
9.29 Stereo image register 0-7: Addresses 0xA0-0xA7

These are eight, 3-bit registers which define the stereo position for the eight possible channels, as defined in *Chapter 12: Sound Features* .



9.30 Sound frequency register: Address 0xB0

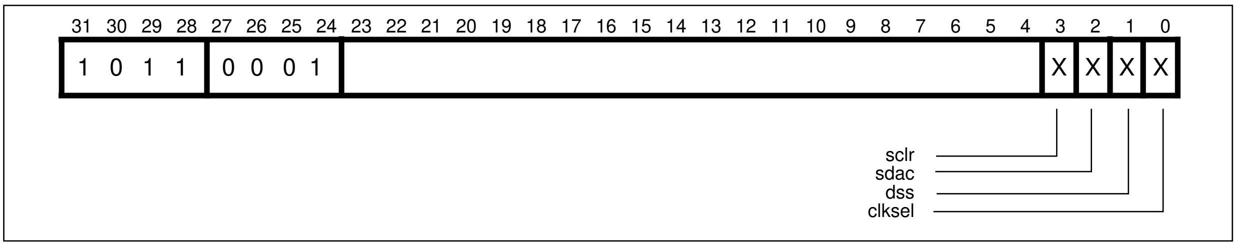
This 8-bit register specifies the byte sample rate of the sound data. It is defined in units of 1µS. See *Chapter 12: Sound Features* for more detail.



If a sample rate of N µs is required, then N-2 should be programmed into the SFR. N may take any value between 3 and 256.

9.31 Sound control register: Address 0xB1

This is a 4-bit register which defines various control bits for the sound system.



Bit 3: SCLR	This bit should always be programmed LOW.
Bit 2: SDAC	When HIGH, the sound DACs are enabled. Two digital signals are also output from the chip in this mode. The first, **WS_LnR**, denotes whether the sound is for the left or right stereo channel. The other, **SD0_MUTE**, goes HIGH between samples, when the sound DACs are being muted to allow for settling. These two signals are intended to ease the connection of external audio processing systems, but for basic operation can be ignored. Note that these two signals have different functions when the serial sound interface is in use.
Bit 1: serial sound	This bit is used to select serial sound mode.
Bit 0: CLKSEL	This bit is used to select which clock is used in the sound system. When HIGH, the ARM7500's internal 32MHz I/O reference clock is used, when LOW the optional sound clock is used.

10 Video Macrocell Interface

This chapter describes the video macrocell interface within the ARM7500.

Pre-release Draft - Unrestricted

10.1 Bus interface

The video macrocell does not use the ARM address bus. The address for programming video and sound registers (0x03400000 to 0x034FFFFF) is decoded elsewhere in ARM7500 and the internal nPROG signal is generated as a general register write strobe. The specific register to be programmed is selected according to the state of the upper bits of the 32-bit input data bus.

All video and sound data is then obtained by DMA under the control of the nVIDRQ internal request signal. This signals to the main ARM7500 bus arbitration logic that a DMA request is pending, and the request will be serviced at the first available opportunity. All DMA is quad word, so four complete data words will be read from memory and stored in the appropriate video, cursor or sound FIFO for each DMA burst. Note that video DMA may be read from memory in bursts of more than 4 words allowing almost continuous DRAM page mode access to occur.

The system software should create a video frame buffer in DRAM memory, and program the DMA address pointers to the start, end and desired initial location within the buffer. All DMA pointer addresses should be quad word aligned. Once the display has been enabled, video registers should only be programmed during the flyback period to ensure flicker free updating of the screen. See *Chapter 13: Memory and I/O Programmer's Model* for details of how to program the DMA controller.

10.2 Setting the FIFO preload value

The Video FIFO is a 32-entry, 32-bit wide FIFO. Words of video data are clocked into the top of the FIFO under control of the internal ARM7500 signals, BUSCLK and nVIDAK. Words are clocked out of the bottom of the FIFO as the video system displays the data, which is controlled by the pixel clock.

The FIFO is flushed during vertical flyback time, so before the start of the frame the FIFO is empty. At the start of the frame a video request is made to the memory subsystem by asserting the internal ARM7500 signal, nVIDRQ. When a predetermined number of words have been loaded into the FIFO the request is removed. As the data in the FIFO is displayed, further video requests are made to refill the FIFO to the desired level.

The Control Register includes a 3-bit field (bits 10:8) to set the preload value of the Video FIFO. In this way the FIFO can be programmed to load 4,8,12,16,20,24 or 28 words of data into the FIFO at the start of frame. After the start of frame, the FIFO will request more data when the number of words in it falls below the preloaded value.

The point at which the FIFO should request more data to be loaded is dependent upon system considerations: if the FIFO is reloaded too late, there is a danger that it will run out of data (underflow); if it is reloaded too early, then there is a danger that the data will not fit into the FIFO (overflow). In general, the higher the bandwidth of the screen, then the more words need to be preloaded into the FIFO. In a low bandwidth screen mode, it is not always desirable to have a large preload value, as the bus traffic will have long bursts of data transfer at the start of the frame.

The optimum value to be preloaded depends upon the screen mode in use (i.e. the rate at which data is read from the FIFO), and both the latency of the memory controller and the rate at which data is provided to ARM7500. It is generally prudent to program the minimum value possible to keep the bus traffic even.

Let:

n	be the value programmed into the control register.
v	(words/ s) be the rate at which video data is displayed
Lmax	(s) be the maximum latency in the memory system. (This is the maximum time between ARM7500 requesting more video data and the memory system delivering the first word of that data.)

If the FIFO is almost empty then it takes 0.025µs for a word of data to reach the bottom of the FIFO before it can be used.

The minimum value for *n* is deduced from the following condition to avoid the FIFO underflowing:

There are 4*n* words in the FIFO when the FIFO requests more data, and if not refilled, then the FIFO would be empty in 4n/v µs.

So *n* must be chosen such that 4*n*/v > (*Lmax*+ 0.025).

The maximum value for *n* is deduced from the following condition to avoid the FIFO overflowing:

In 32-bit bus mode, n may take the maximum value of 7, and the FIFO can never overflow, as there will always be 4 words available in the top of the FIFO, even if the video request is serviced immediately.

10.2.1 Example

For ARM7500, the value of *v* (words/µs) will change depending on the video mode selected and the pixel clock rate chosen, and the worst case DMA latency *Lmax* will alter depending on whether ROM accesses, DRAM accesses or internal programming bursts are slowest, and the MEMCLK frequency used.

The memory subsystems chapter demonstrates how to calculate the worst case DMA latency for a particular system using the ARM7500, and the value calculated there should be imported as lmax into the formula in the previous section.

Assume that an 8 bit per pixel mode is being used with a pixel clock rate of 60MHz (period = 16.7ns). In each pixel clock tick, 1/4 of a word will be used, so in a whole s, 0.25 x 1/0.0167 = 14.9 words will be required.

Hence the value of *n* must be such that:

$$4n/v > (Lmax + 0.025)$$

So, assuming an *Lmax* value of 1.0µs

$$n > 3.74(1.0 + 0.025) \Rightarrow n > 3.83$$

So in this case the minimum value for n to prevent FIFO underflow is 4.

11 Video Features

This chapter details the video capabilities available with the ARM7500.

11.1 Pixel clock

The video and sound macrocell is capable of generating a display at any pixel rate up to 120MHz. The pixel clock may be selected from one of three sources, and the frequency of this clock may be further divided down by a factor of between 1 and 8. These attributes are programmed by the lower 5 bits of the control register, CONREG.

If a maximum of three master frequencies are sufficient, then the clock inputs can be used directly. However, it is often a requirement to have many different master clock frequencies. In order to obviate the need for many crystals on the PCB, the video and sound macrocell is designed to drive a Voltage Controlled Oscillator (VCO) to provide the master frequency. The VCO and filter are external to ARM7500, but everything else is built into the chip. Operation is described below:

An internal reference frequency of 32 MHz is supplied via the **I_OCLK** input of ARM7500. The signal from the VCO is input into ARM7500 on the pin **VCLKI**. **VCLKO** is simply the inverse of **VCLKI**, and this may be used to bias the input signal about the threshold if the VCO output is not a full amplitude signal. The mark-space ratio of the VCO output should be as close as possible to 50-50 if operation at 120MHz is to be achieved.

The reference clock is divided by a programmable number set by the r-modulus in the fsynreg. The VCO clock is divided by a programmable number set by the v-modulus in the fsynreg. Each of the moduli may be a 6 bit number. The output of each of these dividers is fed into a phase comparator, and the result is output from ARM7500 as **PCOMP**. This pin should then be filtered and used to control the VCO output frequency. In this way, the VCO can be set to have a frequency of $v/r * F_{ref}$.

The phase comparator is of the phase-frequency type. The output **PCOMP** is normally tristate, but when the VCO frequency needs to be decreased the output is LOW, and when the VCO frequency needs to be increased the output is HIGH. When the 2 frequencies are in lock, **PCOMP** will normally be tristate, but will be driven to the mid point for a very short time (a few ns) every r/F_{ref}+ period. The output impedance of this pin when it is driven is about 50Ω. ➲*Figure 11-1: ARM 7500 internal subsystems for pixel clock generation* on page 11-3.

The choice of filter and VCO is left to the user. It is important to avoid any low-frequency modulation of the VCO frequency. It has been found that a suitable VCO is a 74AC04 inverter element with feedback, with the supply voltage controlled by the **PCOMP** output. (See ➲*Appendix E: ARM7500 Video Clock sources*.)

With this approach, an enormous number of frequencies are possible. The 32MHz reference frequency generated within ARM7500 can be used to yield the following common VCO frequencies in the table on the next page. For some frequencies, there are many possible values of r and v. In this case it is sensible to choose a set of values which favours the filter response. (Remember large moduli yield a lower comparison frequency).

It may be best to limit the VCO range, and use the prescaler within video and sound macrocell to get a lower pixel rate than the VCO frequency. It is expected that the VCO range may have to be constrained so that it cannot provide the highest frequencies at which the video and sound macrocell can operate. In this case, a single high-frequency clock can be fed into ARM7500 on the **HCLK** pin, and this can be selected for the pixel clock.

r-modulus	v-modulus	VCO frequency/ MHz
8	2	8.0
16	6	12.0
4	2	16.0
8	6	24.0
2	2	32.0
8	9	36.0
16	35	70.0
4	15	120.0

Table 11-1: Synthesised VCO frequency settings

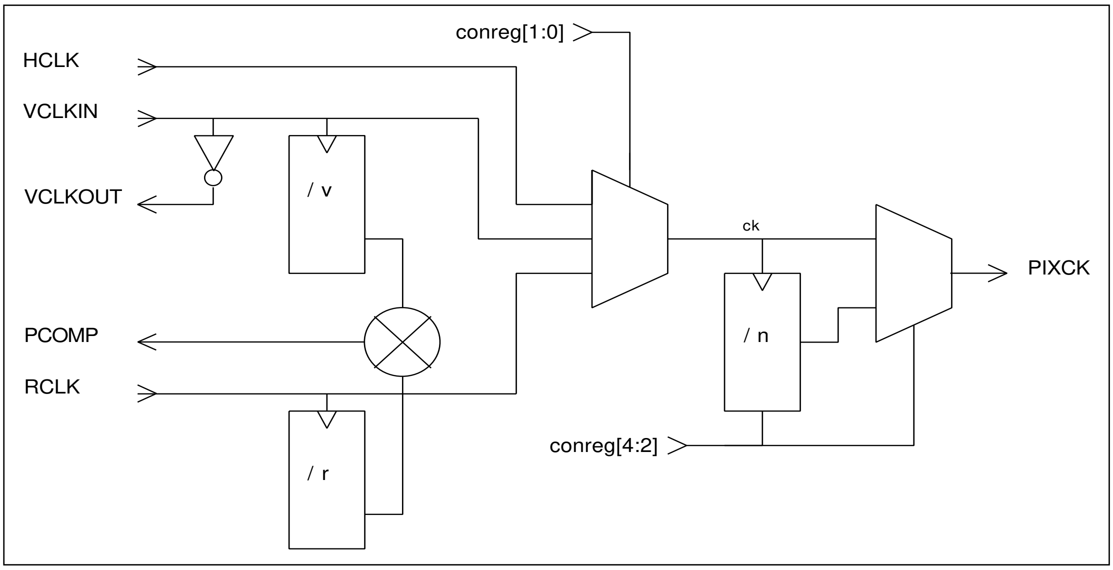


Figure 11-1: ARM 7500 internal subsystems for pixel clock generation

11.2 The palette

ARM7500 has a 28-bit wide 256-entry palette which is constructed out of three 8-bit wide look-up-tables (LUTs), each with 256 entries, named Red, Green, and Blue, and one 4-bit wide LUT with 16 entries, named Ext. The Red, Green and Blue LUTs each drive their respective DACs, and the Ext LUT is normally configured to drive the **ED[3:0]** output port, except when Hires mode or LCD mode is selected. These bits may be used outside the chip for a variety of purposes such as supremacy, fading, Hi-Res and LCD driving. The **ED[7:4]** output port is normally driven from the Ext register, ereg[7:4], which may be written at any time, so these bits can be used as a DC control port.

The mapping of the logical colours through the LUTs is dependent on the mode in use, as follows:

- In 1,2,4 bits/pixel modes, the logical data is fed simultaneously to all 4 LUTs. This gives a fully flexible palette with any logical colour being mapped to any physical colour, and any **ED[3:0]** value. The palette will give 16 colours from a selection of 2^{24}.
- In 8-bits/pixel modes, the logical data is fed simultaneously to all 4 LUTs. This gives a fully flexible palette with any logical colour being mapped to any physical colour. Logical colours 0-15 access the Ext LUT, and logical colours 16-255 access location 0 of the Ext LUT. The Ext LUT again drives **ED[3:0]**. The palette will give 256 colours from a selection of 2^{24}.
- In the 16-bits/pixel mode, a patented technique has been developed. This approach is highly flexible and allows many different addressing modes e.g. 5-5-5, 5-6-5 etc. In this mode 2^{16} colours are available from a selection of 2^{24}.
- In the 32-bits/pixel mode, 24 bits from the logical field will drive the 256 entries in each of the colour LUTs (8 bits to each LUT) and 4 bits will drive the Ext LUT. The upper 4 bits are discarded.The palette will give the full range of 2^{24} colours.

Note that where a logical field does not drive all the palette entries (such as in 4 bits/pixel mode) only the lower part of the palette is used. Unused sections need not be programmed.

When HiRes mode or LCD mode is selected, the palette must be set up in a predetermined configuration. This is explained in the chapters on hi-res support and LCDs.

11.2.1 Palette updating

A signal FLYBK exists within ARM7500 as an output from the video and sound macrocell. FLYBK goes HIGH at the start of the first raster which is not displayed, and goes LOW at the start of the first raster which is displayed. The rising edge of this signal can cause an interrupt via the ARM7500 IRQA interrupt registers, and the palette should be updated at this time for flicker-free updating.

11.3 Cursor

ARM7500 has a hardware cursor 32 pixels wide and any number of pixels high. Its 2 bits per pixel allow 4 colours, which include “transparent” plus three other colours from a selection of 2^{24}. It is possible to display the cursor in the horizontal border, but not in the vertical border.

The cursor has a 3 entry palette which is 28 bits wide, allowing each cursor logical colour to be any physical colour. In addition, there is a 28 bit wide border colour register.

At the start of every frame, 16 bytes of cursor data are transferred to the video subsystem during the horizontal retrace period. This is enough data for two raster's worth of cursor. After they have been displayed, a request is made for another 16 bytes. Thus, in normal mode, requests are made on every other raster on which there is cursor, and enough data is transferred for two rasters each. In Hi-Res mode, a request is made every raster. Note that the cursor data is always transferred in bursts of four words.

11.3.1 Cursor in Hi-Res mode

In order to allow micro-pixel resolution of the cursor in Hi-Res mode when operating at 4 micro-pixels per normal pixel, it is necessary to define 2 bits per micro-pixel, or 8 bits per normal pixel. The 16 bytes of cursor data available for each raster can thus generate 64μ-pixels of cursor. In Hi-Res mode the cursor palette is not used (though the border may be programmed). Refer to the chapter on Hi-Res support.

The cursor is always positioned to align with a normal pixel. In order to position the cursor to a μ-pixel horizontally, four different copies of the cursor are required: each copy defines the cursor offset by a single μ-pixel. It is possible to define transparency to a resolution of a μ-pixel, so by selecting the correct cursor image, the required position can be achieved.

11.3.2 Cursor in LCD mode

The video subsystem is capable of displaying the hardware cursor in LCD mode. However, because of the split-screen nature of duplex LCDs, the cursor needs special attention. If the cursor is entirely in the upper or lower half-screen, then the cursor should be programmed as normal, but VCSR[14:13] should be programmed accordingly (0x10 = upper half-screen; 0x01 = lower half-screen). If the cursor “straddles” the split screen, then the cursor image in memory must start at the top of the lower half-screen, and end with the bottom of the upper half screen. Hence two contiguous images of the cursor image are required, and the start pointer moved accordingly. In practice, four images of the cursor are required, to ensure that a resolution of one raster is maintained across the boundary. As the cursor moves from one panel to the other, the pointer to the cursor image in memory must be moved. For more details, refer to *Appendix B: Dual Panel Liquid Crystal Displays*.

In the case where the cursor straddles the split screen, the meaning of the VCSR and VCER registers are changed. The VCER register now defines the start of cursor in the upper half-screen, and the VCSR defines the end of the cursor in the lower half-screen. Thus the cursor is actually displayed in the lower half-screen from the start of display until VCSR, and then again in the upper half-screen from VCER until the end of display. This mode is selected by programming VCSR[14:13] = 0x11. Further details of how to use ARM7500 with dual panel LCD screens are given in *Appendix B: Dual Panel Liquid Crystal Displays*.

11.4 Hi-Res support

ARM7500 is able to support colour screens with resolutions above 1024 by 768 pixels. For higher resolutions, externally serialising the data is required to produce monochrome (or grey-level) pictures. In this scheme one 16ns-pixel could theoretically be serialised to make eight 2ns-pixels, ie about 500MHz. However, this is dependent on the availability of external hardware capable of generating a serial bitstream at this frequency.

11.4.1 ARM7500 support for Hi-Res mode

When the hrm bit in the Ext register is set, and EREG[1:0] is set to value 0x10, ARM7500 outputs 8 bits of data for every normal pixel on the **ED[7:0]** port. These bits can then be serialised to form a high frequency monochrome pixel stream; alternatively they can be serialised to 2 or 4 bits, which could then drive a high-speed monochrome DAC for grey level displays. With the pixel clock running at a fundamental frequency of about 100MHz, the external serial clock could be running at up to several hundred MHz. In order for the external circuit to be able to synchronise to the ARM7500 output data, ARM7500 also outputs a pixel clock synchronous to the data stream when the hrm bit is set.

In this mode, with EREG[1:0] set to value 0x10, the video data is driven from the Blue LUT, which outputs data BPD[7:0]. Depending on how the external serializer circuit is arranged, the LUT must be set up to give a one-one correlation between the logical address and the physical data value. So, for example, if 4 bits are externally serialised into a single bit stream, then 4 bits/pixel mode should be selected, and **ED[6,4,2,0]** should be used. The lower 16 words of the Blue LUT should be programmed to give all 16 combinations of BPD[6,4,2,0]. If 8 bits are externally serialised to give a single bit-stream, then 8 bits/ pixel mode should be selected, and all 256 values of the Blue LUT should be programmed as a one-one mapping.

Hardware cursor support is provided as follows. The cursor palette is not used, though the Blue border may be programmed. Eight bits of cursor data (CD[7:0]) are defined for each normal pixel. The 8 bits are divided into 4 pairs, with the lsb (least significant bit) of each pair defining whether the video data (BPD) or the msb (most significant bit) of the cursor pair is displayed. Each cursor bit-pair operates on 2 bits of the video data (BPD) according to the following tables:

CD[7]	CD[6]	ED[7]	ED[6]
0	0	BPD[7]	BPD[6]
0	1	0	0

Table 11-2: Deriving high-speed 2-bit cursor data from the normal 8-bit output—CD[6&7]

CD[7]	CD[6]	ED[7]	ED[6]
1	0	BPD[7]	BPD[6]
1	1	1	1

Table 11-2: Deriving high-speed 2-bit cursor data from the normal 8-bit output—CD[6&7]

CD[5]	CD[4]	ED[5]	ED[4]
0	0	BPD[5]	BPD[4]
0	1	0	0
1	0	BPD[5]	BPD[4]
1	1	1	1

Table 11-3: Deriving high speed 2-bit cursor data from the normal 8-bit output - CD[4&5]

CD[3]	CD[2]	ED[3]	ED[2]
0	0	BPD[3]	BPD[2]
0	1	0	0
1	0	BPD[3]	BPD[2]
1	1	1	1

Table 11-4: Deriving high-speed 2-bit cursor data from the normal 8-bit output—CD[2&3]

CD[1]	CD[0]	ED[1]	ED[0]
0	0	BPD[1]	BPD[0]
0	1	0	0
1	0	BPD[1]	BPD[0]
1	1	1	1

Table 11-5: Deriving high speed 2 bit cursor data from the normal 8 bit output - CD[0&1]

So if the external circuit serialises **ED[6,4,2,0]** into a single bit stream, or **ED[7:0]** into a 2-bit data stream then the cursor can be positioned and defined to any micro-pixel: in each case the cursor can be transparent, black or white. If all 8 bits are serialised into a single very high frequency bit stream, then the cursor can only be positioned and defined to units of 2 micro-pixels.

11.5 Liquid Crystal Displays

ARM7500 is capable of driving single panel Liquid Crystal Displays at 1, 2, 4, 8, 16 or 32 bits per pixel, and dual panel LCDs at 1, 2 or 4 bits per pixel. Grey-scaling is provided at up to 16 shades. ARM7500 is also capable of driving single panel colour LCDs with no grey scaling in its normal (video) mode. Two control bits are provided for LCD operation:

lcd	(bit 13 in the Ext register) configures the external data port **ED[7:0]** for LCD operation, and enables the grey-scaling logic (EREG[1:0] must be set to 0x01);
dup	(bit 13 in the control register) enables duplex mode, and should be set for dual-panel LCDs.

11.5.1 LCD grey-scaling

To obtain a grey-scaled output from ARM7500, the lcd bit (bit 13 in the Ext register) must be set. This configures the External port for LCD operation. The DACS should be disabled to save power since ARM7500 can not drive both CRT and LCD displays simultaneously. In order to get this data out of the **ED[7:0]** port, EREG[1:0] must be set to value 0x01.

ARM7500 provides a grey-scaling algorithm which modulates the data output. Grey-scaling is possible at 1, 2 or 4 bits per pixel. The data is output from the chip as one or two 4-bit quantities, depending on whether single or dual panel LCDs are used, at one quarter of the pixel rate. The lower 4 bits of the Green LUT control the upper panel (**ED[7:4]**), and the 4 bits of the Ext LUT control the lower panel (**ED[3:0]**). Thus, the palette can still be used to provide a mapping of logical to physical colour. The cursor palette is used similarly, though the programming of the cursor position needs special treatment - refer to Appendix B. If a single panel LCD is used, then **ED[7:4]** should be used, and the Green LUT programmed accordingly (**ED[3:0]** are held low in this mode). The grey-scaling logic lies between the output of the video multiplexer and the external port and works as described below.

There are effectively 16 physical grey levels available, and in 1,2, or 4 bits per pixel mode the palettes are programmed to give a mapping of the logical colour to physical shade. The resultant 4 bit pixel value out of the video multiplexer is modulated according to its value and the raster number and the point on the raster at which it is generated. The result is a single bit which on average is HIGH for a time equal to the actual 4 bit value. For a single panel screen, 4 of these bits are then collected together and output as a nibble at one quarter of the pixel rate on **ED[7:4]**. **ED[4]** represents the 4th pixel, and **ED[7]** represents the 1st pixel.

If duplex mode is selected, then the pixel stream for the upper half screen is obtained from the Green LUT and that for the lower half screen is obtained from the Ext LUT. Both these pixel streams are passed through the grey-scale logic simultaneously and output as two nibbles on **ED[7:4]** (upper half screen) and **ED[3:0]** (lower half screen).

11.5.2 Dual panel LCDs (duplex mode)

Duplex mode is configured by setting the dup control bit as well as the lcd control bit. The screen parameters are set up according to the requirements of the LCD panel.

Note: *Since the upper and lower panels are driven simultaneously, ARM7500 only produces data for half the total number of lines on the dual panel. Thus the vertical registers must be programmed as if there were only one panel.*

ARM7500 requests data in units of two quad-words. The first quad word the memory controller delivers is for the upper half-screen, and the second quad-word is for the lower half-screen. ARM processor then serialises the data into two simultaneous bit-streams as described above. 1, 2 or 4 bits/pixel may be selected. For details of the ARM7500 register programming requirements for duplex DMA, see *Chapter 13: Memory and I/O Programmer's Model* .

11.5.3 Single panel colour LCDs

If neither dup nor lcd control bits are set, then the **ED[7:0]** port may be used to gain access to all of the physical bits out of the video multiplexer. This would allow many other types of display to be driven.

11.6 External support

ARM7500 has an 8-bit output port, **ED[7:0]** and a synchronous clock, **ECLK**, which have different functions in different modes. The port is controlled by the 2 bits, EREG[1:0], in the control register that essentially select which of the bytes from the video multiplexer are chosen. Additionally, an ARM7500 register bit (bit 1 of the VIDMUX register) can be used to cause the data selection for the **ED** port to be modified according to the state of the **ECLK** output. This feature is intended to be used to increase the bandwidth for driving colour LCD screens. When this control bit is set LOW, the behaviour of the **ED** port is as shown below. The bit is intended to be used with 'LCD' set LOW. When the VIDMUX bit is HIGH, and EREG[1:0] is set LOW, if **ECLK** is LOW, the Red LUT is output on **ED[7:0]**. If **ECLK** is high, the Green LUT is output on **ED[7:0]**.

When EREG[1:0] = 0:

the Red LUT is output on **ED[7:0]**.

When EREG[1:0] = 1:

if lcd = 0 the Green LUT is output on **ED[7:0]**.

If lcd = 1, then the grey-scaled LCD signals are output. **ED[7:4]** carries the data for the upper half screen from the Green LUT, and **ED[3:0]** carries the data for the lower half screen from the Ext LUT. Note that if lcd = 1, data is output at one-quarter of the ARM processor pixel rate, since the data output actually represents 4 pixels for each half-screen.

When EREG[1:0] = 2:

if hrm = 0, the Blue LUT is output on **ED[7:0]**.

If hrm = 1, the multiplexed Blue LUT and HiRes cursor data is output on **ED[7:0]**. See *11.4 Hi-Res support* on page 11-6.

Also, **ED[7:0]** is retimed, and delayed by one extra pixel.

When EREG[1:0] = 3:

if dac = 0, **ED[3:0]** are driven by the Ext LUT, and **ED[7:4]** are driven by the value of the Ext Register,EREG[7:4], which is intended as a DC control port in this mode.

If dac= 1, **ED[3:0]** are delayed by one pixel, so that they are exported from the chip in the same pixel as the analog data to which they correspond. In this configuration **ED[3:0]** bits may be used for supremacy, for overlaying pictures on a pixel-by-pixel basis. Because several bits are output, analog fading and mixing on a pixel basis is possible.

11.6.1 ECLK

ECLK is output along with the data **ED[7:0]**, so that the data can be externally latched and multiplexed. **ECLK** is controlled by lcd and EREG[2]. If EREG[2] = 0, then **ECLK** is output as logic 0. This should be configured whenever **ECLK** is not required, in order to save power. If EREG[2] = 1, then if lcd = 0, **ECLK** is the pixclk, output synchronously with the data stream. If lcd = 1, then **ECLK** is the LCD clock, which runs at a quarter of the pixel rate. The lcd clock is only enabled whilst horizontal display data is being output and is synchronous to the data stream. The timing diagrams below show the relationship between **ED** and **ECLK.**

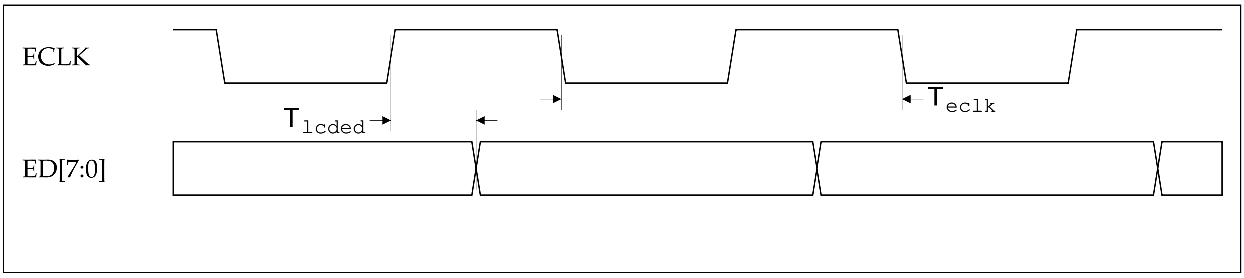


Figure 11-2: Timing relationship between ECLK and ED in LCD greyscale mode

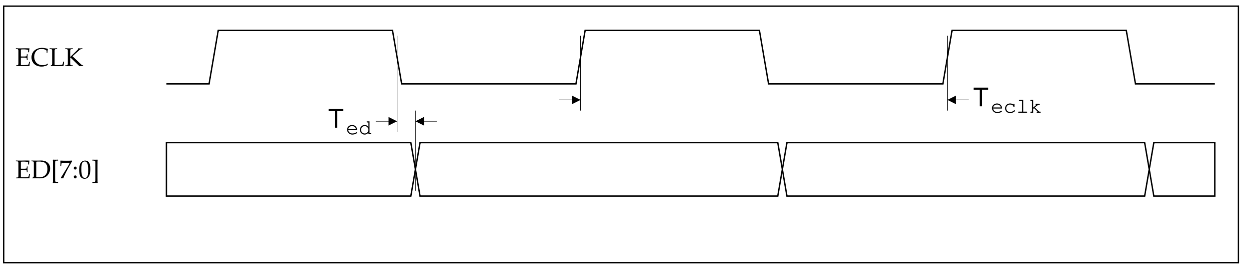


Figure 11-3: Timing relationship between ECLK and ED in all other modes

Symbol	Parameters	Min	Typ	Max	Units	Notes
Teclk	**ECLK** period					
Ted	**ECLK** to **ED** delay					
Tlcded	**ECLK** to **ED** delay—LCD mode					

Table 11-6: ARM7500 ECLK and ED timing

11.6.2 Power saving considerations

The External Port can consume a lot of power, but steps may be taken to minimise power usage. In particular, it is very important not to load the signals heavily, especially **ECLK** which can clock at the pixel rate. When it is not in use, it should not be putting out the raw pixel data, but should be outputting static signals. This is done by selecting EREG[1:0] = 3, and setting all entries of the Ext LUT to be all one value. **ECLK** should be turned off by setting EREG[2] = 0.

If an LCD is fitted, but not operated, it may be necessary to power down the input signals to it. This can be achieved by setting bit 13 low, which disables the grey scaler, and by disabling the external port as described above.

11.6.3 Vertical and horizontal synchronisation

Software control over the polarities of the synchronisation pulses is provided. Two types of Composite Sync may be output, each of either polarity. The logical OR of Hsync and Vsync may be output on the Horizontal Sync (**HSYNC**) pin, and the XOR of Hsync and Vsync may be output on the Vertical Sync (**VSYNC**) pin. Equalisation pulses in the composite synchronisation signal are supported for interlace mode. When LCD mode has been selected, the external **HSYNC** and **VSYNC** pulses are modified in accordance to the requirements of an LCD screen.

The **HSYNC** and **VSYNC** pins are programmed with the Ext Register, EREG[19:16].

11.6.4 Genlocking

Genlocking is supported by ARM7500. A pin is provided to reset the vertical counter to the first raster (**SYNC**).

11.7 Analog outputs

ARM7500 outputs analog R, G, and B signals. It is designed to drive doubly-terminated 75Ω lines directly.

11.7.1 DAC control

There are 4 control bits in the Ext Register associated with the DACs. These are dac and ped[2:0].

Power-save mode

When dac is HIGH, the DACs are all enabled and will generate a current proportional to the digital values from the video multiplexer. When dac is LOW, the reference current into all three DACs is turned off, so the DACs generate no output current, and hence consume much less power. This is useful when operating in LCD mode, or at any time when the screen should be blanked.

Pedestal current

The DACs may be programmed to generate a pedestal offset of 20 lsb equivalent currents. These are controlled individually by pedon[2:0], though they will typically all be programmed on or off together, depending on the monitor characteristics. pedon[0] controls the red pedestal, pedon[1] the green pedestal, and pedon[2] the blue pedestal. If pedon[n] is HIGH, the pedestal current is switched on as the border starts, and is turned off as the border ends.

11.7.2 Video DAC currents

The DACs are each 8 bit resolution, so they source 256 units of current according to the digital value from the video multiplexer. The current step is set by a common reference current, **VIREF**. The recommended reference current is 0.56mA which gives a DAC step of 69µA. Hence digital value 0 gives 0 current and digital value 0xFF gives an output current of (255 * 69)=17.6mA. If pedon is set, then during display time, digital value 0 will generate (20 * 69)=1.38mA, and digital value 0xFF will generate (275* 69)=18.98mA. A 4.3kΩ resistor connected between **VIREF** and **VDD** will provide the desired 0.56mA at 2.6V.

DAC accuracy

At 120MHz the DACs are accurate to 8 bits absolute resolution. They will always be monotonic.

11.7.3 Monochrome output

ARM7500 does not generate a separate composite monochrome signal. This can be generated by resistively mixing the R,G and B externally, if required.

12 Sound Features

This chapter details the sound capabilities available with the ARM7500.

12.1 Sound

The video and sound macrocell has two sound systems built into it. These are an 8-bit analog stereo system, and a 32-bit serial sound interface suitable for driving external CD DACs. Only one of these systems should be used at once, as they share common circuitry.

12.2 The sound FIFO

At the core of the sound system is a 4-word FIFO and a byte wide latch. When empty, the FIFO fills completely by a DMA request. Data is then clocked out of the FIFO, one byte at a time through the latch.

12.3 Analog stereo sound

This mode can work with 1, 2, 4 or 8 stereo channels. The 8-bit sound data is fed through a DAC to produce the sound signal. The first quarter of each sample is muted to allow for DAC settling. During this time, the external, digital **SDO_MUTE** signal goes HIGH. The stereo image is synthesised by time division multiplexing the sound signal between the four analog output pins LM, LP, RM, and RP. The sound DAC characteristic is shown in *Figure 12-1: The sound DAC characteristic* on page 12-3. The sign bit, D0, selects which of the plus or minus pins of the appropriate channel are driven by the DAC. When D0 = 1, LP or RP will be driven, and when D0 = 0, LM or RM will be driven. A digital output, **WS_LnR**, is provided which denotes when the output data is for the left (**WS_LnR** = 1) or right stereo channel. The stereo position for each channel is held in the stereo image registers, and can be in one of eight positions as described below.

SIR value	Stereo Position
0	Undefined
1	100% Left
2	83% Left
3	67% Left
4	Centre
5	67% Right
6	83% Right
7	100% Right

Table 12-1: Stereo position values

In 8 channel mode, the channels are sampled sequentially starting with channel 0. In 4 channel mode, the fifth byte sampled is channel 0 again, and so stereo image register 4 must be programmed to the same value as that of register 0, and so on. In

2 channel mode, registers 0, 2, 4 and 6 correspond to channel 0, and registers 1, 3, 5 and 7 correspond to channel 1. In single channel mode, all eight registers must be programmed to the same value.

The sample rate is programmable via the Sound Frequency Register (SFR). The SFR is programmable in units of 1µS, and the minimum value is 3µS. In eight channel mode, each channel is sampled at a rate of one eighth the value of the SFR.

The DAC transfer characteristic consists of 8 linear segments (chords). Each chord has 16 steps, and the step size in one chord is twice that of the preceding chord. This gives an approximation to the “µ255 law”. The characteristic is shown in the figure below, for the positive and negative halves. Note that bit 0 of the sound data is used as the sign bit.

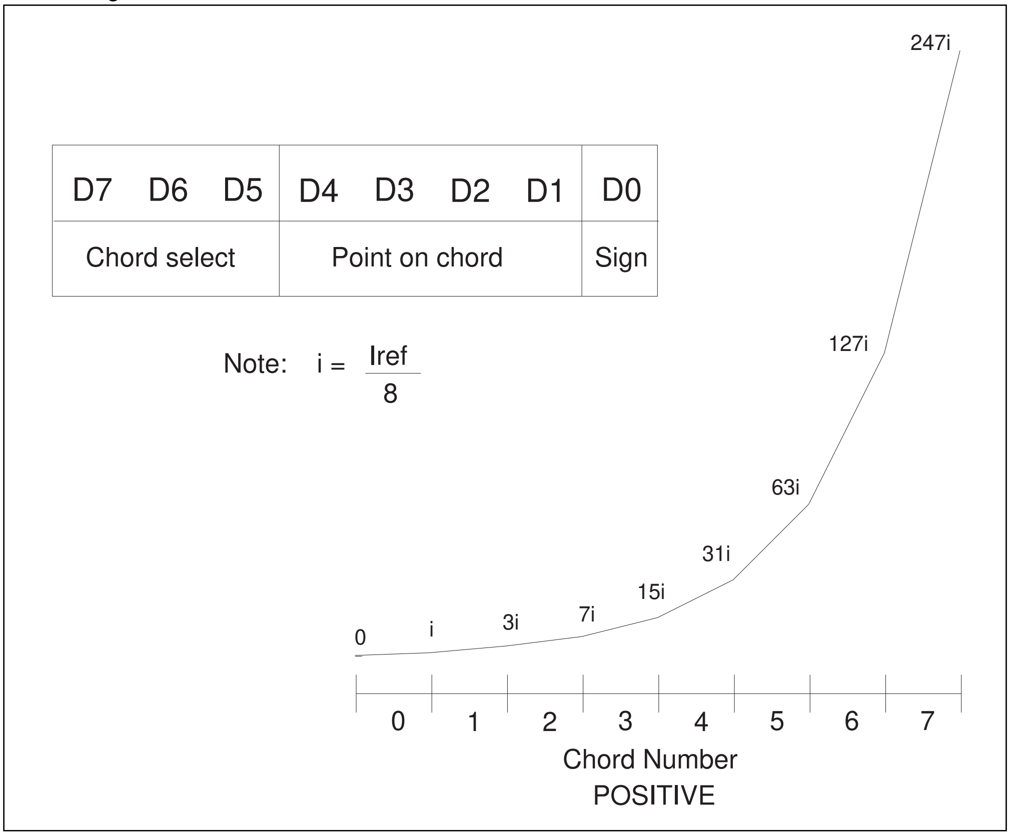


Figure 12-1: The sound DAC characteristic

The outputs are current sinks. The magnitude of the output is a function of the sound reference current. The reference current is equal to the step size of the highest chord. It is recommended that the reference current **SIREF** is 32µA, which may be provided by a 88.5kΩ resistor to **VDD**. The digital outputs, **SDO_MUTE** and **WS_LnR** are provided for advanced external audio systems but for basic operation they can be ignored.

When connected to VDD by a 1k pullup resistor the analog stereo outputs **LM**, **LP**, **RM** and **RP** will have a voltage range of 4.5 0.5V with a 2mA maximum current.

12.4 The Digital Serial Sound Interface

The serial sound interface offers far superior 32-bit stereo sound, at the expense of some additional external circuitry. The serial sound system consists of a three pin serial interface. **SDCLK** is the Serial Data Clock output, **SDO_MUTE** is the Serial Data output, and **WS_LnR** is the Word Select output. When no sound is required, (sctl[2:1]=0), these outputs are stable (**SDCLK**=0, **SDO_MUTE**=0, **WS_LnR**=1).

When in this mode, bytes from the sound FIFO are output in most-significant first order. This is because the serial sound output must go msb first to be compatible with other serial sound devices. Each byte of data is loaded into a parallel-in, serial-out register, and clocked out under control of the bit clock.

There are two timing formats available for the interface: *normal* and *Japanese* formats. The selection of these is controlled by bit 0 of the VIDMUX register in the main part of ARM7500.

When configured for normal mode (VIDMUX bit 0=LOW), each 32-bits sample consists of 16 bits for the left hand channel, and 16 bits for the right hand channel. To distinguish between them, a 'word select' (**WS_LnR**) signal is produced. This signal changes when the lsb of the previous word is output. When **WS_LnR** is HIGH, the right hand channel is being output. This timing relationship is shown below.

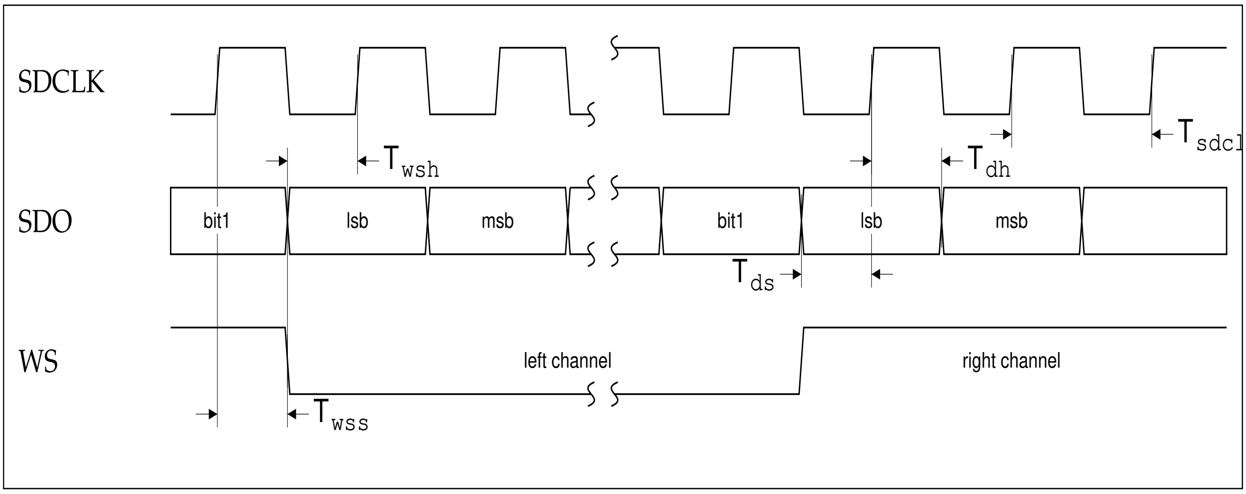


Figure 12-2: Serial sound output timing

In Japanese format, the **WS_LnR** signal changes when the msb of the new word is output. In addition, the polarity of **WS_LnR** is reversed. This is shown in the diagram below.

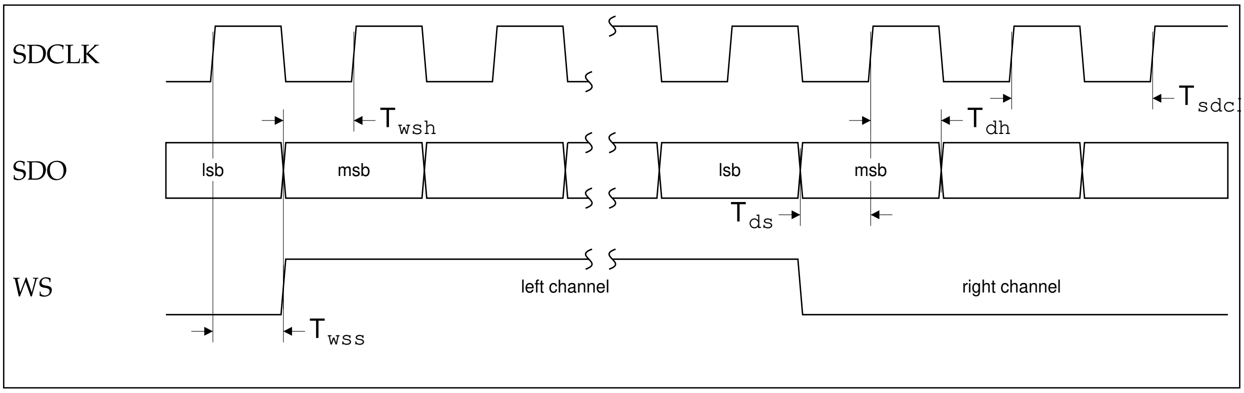


Figure 12-3: Serial sound interface—Japanese timing

The serial sound output can be used with any DAC with a serial sound input. Many DACs require a 11.2896MHz input clock, and to reduce the number of on board crystals required, the video and sound macrocell can cope with this frequency on the **SCLK** input. When using this, the following parameters need to be programmed in the registers.

serial sound (SCTL Register bit 2) = 1

clksel (SCTL Register bit 0) = 1

Sound Frequency Register = 2

The sound system is not limited to operating with this frequency alone, however the Sound Frequency Register must be set to produce the necessary bit rate accordingly.

12.5 Analog sound outputs

When the sound system is in analog mode, analog sound data is output from the sound DACs. There are four outputs, a positive and negative for both the left and the right channels. Also, two digital outputs are produced. On the **SDO_MUTE** pin, the mute signal is output. This goes HIGH for a period between samples to allow for DAC settling. On the **WS_LnR** pin, the Left-not-Right signal is output. This goes HIGH when the analog output is to the left hand channel, and goes LOW when it is for the right hand channel. These two digital outputs are intended to ease the connection of an external audio processing system, and can be ignored for normal use.

13 Memory and I/O Programmer's Model

This chapter details the programmable registers for the memory and I/O subsytem.

13.1 Introduction

ARM7500 contains over 100 programmable registers (in addition to those in the ARM Processor and the 256 video palette entries), which are grouped into three sets. Those inside the ARM Processor are described fully in Chapters 3 to 7. Those inside the video and sound macrocell are all programmed by writing to memory locations 0x03400000 to 0x034FFFFF, with the upper bits of the programmed data determining which video/sound register is to be programmed. All these registers are write only, and are described in the video and sound chapters. The remaining ARM7500 registers are programmed by writing a full 32-bit data word to an address between 0x03200000 and 0x032001F8. Although most of these registers are only 8 or 16 bits wide, all the register addresses are word aligned. All the ARM7500 registers which do not form part of the ARM Processor or the video and sound macrocell are described in the following section.

13.2 Summary of registers

All addresses are in hex and are relative to the base address 0x03200000.

Key

✓	can write or read
✗	do not write or read

Name	Address	Size	Read	Write	Function
IOCR	00	8	✓	✓	I/O control
KBDDAT	04	8	✓	✓	Keyboard data
KBDCR	08	8	✓	✓	Keyboard control
IOLINES	0C	8	✓	✓	general purpose I/O lines
IRQSTA	10	8	✓	✗	IRQA status
IRQRQA	14	8	✓	✓	IRQA request/clear
IRQMSKA	18	8	✓	✓	IRQA mask
SUSMODE	1C	8	✓	SUSPEND	Enter SUSPEND mode
IRQSTB	20	8	✓	✗	IRQB status
IRQRQB	24	8	✓	✗	IRQB request
IRQNSKB	28	8	✓	✓	IRQB mask
STOPMODE	2C	8	✗	STOP	Enter STOP mode
FIQST	30	8	✓	✗	FIQ status

Table 13-1: ARM7500 registers

Name	Address	Size	Read	Write	Function
FIQRQ	34	8	✓	✗	FIQ request
FIQMSK	38	8	✓	✓	FIQ mask
CLKCTL	3C	8	✓	✓	Clock divider control
T0LOW	40	8	✓	✓	Timer 0 LOW bits
T0HIGH	44	8	✓	✓	Timer 0 HIGH bits
T0GO	48	8	✗	GO	Timer 0 go command
T0LAT	4C	8	✗	LATCH	Timer 0 latch command
T1LOW	50	8	✓	✓	Timer 1 LOW bits
T1HIGH	54	8	✓	✓	Timer 1 HIGH bits
T1GO	58	8	✗	GO	Timer 1 go command
T1LAT	5C	8	✗	LATCH	Timer 1 latch command
IRQSTC	60	8	✓	✗	IRQC status
IRQRQC	64	8	✓	✗	IRQC request
IRQMSKC	68	8	✓	✓	IRQC mask
VIDMUX	6C	8	✓	✓	LCD and IIS control bits
IRQSTD	70	8	✓	✗	IRQD status
IRQRQD	74	8	✓	✗	IRQD request
IRQMSKD	78	8	✓	✓	IRQD mask
ROMCR0	80	8	✓	✓	ROM control bank 0
ROMCR1	84	8	✓	✓	ROM control bank 1
REFCR	8C	8	✓	✓	Refresh period
ID0	94	8	✓	✗	Chip ID number LOW byte
ID1	98	8	✓	✗	Chip ID number HIGH byte
VERSION	9C	8	✓	✗	Chip version number
MSEDAT	A8	8	✓	✓	Mouse data
MSECR	AC	8	✓	✓	Mouse control

Table 13-1: ARM7500 registers (Continued)

Name	Address	Size	Read	Write	Function
IOTCR	C4	8	✓	✓	I/O timing control register
ECTCR	C8	8	✓	✓	Expansion card timing control register
ASTCR	CC	8	✓	✓	Asynchronous timing control
DRAMWID	D0	8	✓	✓	DRAM width control 16/32 bit
SELFREF	D4	8	✓	✓	Force CAS/RAS lines LOW individually
ATODICR	E0	8	✓	✓	A to D interrupt control register
ATODSR	E4	8	✓	✗	A to D status register
ATODCC	E8	8	✓	✓	A to D convertor control register
ATODCNT1	EC	16	✓	✗	A to D counter 1
ATODCNT2	F0	16	✓	✗	A to D counter 2
ATODCNT3	F4	16	✓	✗	A to D counter 3
ATODCNT4	F8	16	✓	✗	A to D counter 4
SD0CURA	180	32	✓	✓	Sound DMA 0 CurA
SD0ENDA	184	32	✓	✓	Sound DMA 0 EndA
SD0CURB	188	32	✓	✓	Sound DMA 0 CurB
SD0ENDB	18C	32	✓	✓	Sound DMA 0 EndB
SD0CR	190	8	✓	✓	Sound DMA Control
SD0ST	194	8	✓	✗	Sound DMA Status
CURSCUR	1C0	32	✓	✓	cursor DMA current
CURSINIT	1C4	32	✓	✓	Cursor DMA Init
VIDCURB	1C8	32	✓	✓	Duplex LCD current register B
VIDCURA	1D0	32	✓	✓	Video DMA current A
VIDEND	1D4	32	✓	✓	Video DMA End
VIDSTART	1D8	32	✓	✓	Video DMA start
VIDINITA	1DC	32	✓	✓	Video DMA Init

Table 13-1: ARM7500 registers (Continued)

Name	Address	Size	Read	Write	Function
VIDCR	1E0	8	✓	✓	Video cursor DMA control
VIDINITB	1E8	32	✓	✓	Duplex LCD init register B
DMAST	1F0	8	✓	✗	DMA interrupt status
DMARQ	1F4	8	✓	✗	DMA interrupt request
DMASK	1F8	8	✓	✓	DMA interrupt mask

Table 13-1: ARM7500 registers (Continued)

13.3 Register description

13.3.1 IOCR (0x00) - I/O control

This register is used to control various I/O functions. The value of the FLYBACK signal from the video subsystem can be examined by reading bit 7 of this register, this would be important for genlocking as FLYBACK will provide information about the vertical timing of the display. The FLYBACK bit also gives information about when the video palette registers can safely be reprogrammed. This should only be done during the FLYBACK period, when this bit has been set HIGH. Control of the open drain **OD[1:0]** and **ID** pins is provided from this register. It is also possible to read the status of the **nINT1** pin.

7	6	5	4	3	2	1	0
F	N	1	1	I	1	C	D

F = FLYBACK value

N = **nINT1** value

I = **ID** open drain pin control

C = **OD[1]** open drain pin control

D = **OD[0]** open drain pin control

Write	bits[7:4,2] ignored bit[3,1:0] open drain pin controls, 0: force pin LOW, 1: pin is input only
Read	bit[7] reads current FLYBACK value from video and sound macrocell bit[6] reads current **nINT1** pin value bits[5:4,2] read one bit[3] reads current **ID** pin value bit[1] reads current **OD[1]** pin value bit[0] reads current **OD[0]** pin value
Reset	bits[3,1:0] set as inputs (HIGH)

13.3.2 KBDDAT (0x04) - Keyboard data

7	6	5	4	3	2	1	0
D	D	D	D	D	D	D	D

D = keyboard data

Write	next byte to be sent over serial interface to keyboard
Read	last byte of data received from keyboard

13.3.3 KBDCR (0x08) - Keyboard control

7	6	5	4	3	2	1	0
T	T	R	R	E	P	D	C

T = transmit status

R = receive status

E = enable

P = received parity

D = data pin status

C = clock pin status

Write bits[7:4,2] ignored
bit[3] enable, 0: state machine cleared, 1: state machine enabled
bit[1] force **KBDATA** pin LOW, 0: don't force LOW, 1: force LOW
bit[0] force **KBCLK** pin LOW, 0: don't force LOW; 1: force LOW

Read bit[7] TXE, shift register empty, 0: not ready, 1: enabled and ready to transmit
bit[6] TXB, transmitter busy, 0: not busy, 1: currently sending data
bit[5] RXF, receive shift register full, 0: not full, 1: ready to read
bit[4] RXB, receiver busy, 0: not busy, 1: currently receiving data
bit[3] ENA, state machine enable, 0: disabled, 1: enabled
bit[2] RXP, receive parity bit, odd parity bit for last received data
bit[1] SKD, **KBDATA** pin value after synchronisation
bit[0] SKC, **KBCLK** pin value after synchronisation

13.3.4 IOLINES (0x0C) - IOP[7:0] port control

This register is the control for the 8-bit I/O port included in the ARM7500. Each bit independently controls the state of one of the open drain I/O pins **IOP[7:0]**. On reset all the bits are configured to be inputs.

7	6	5	4	3	2	1	0
I	I	I	I	I	I	I	I

I = **IOP** open drain pin

Write zero force corresponding pin LOW

Write one corresponding pin becomes an input

Read read value on corresponding pin

Reset	set all as inputs

13.3.5 IRQSTA (0x10) - IRQ A interrupts status

There are four sets of IRQ interrupt control, masking and status registers in ARM7500, of which this is the first. Not all the bits in each register are used. Note that this status register contains a bit (7) which is always active, and this can be used to force an interrupt from software by programming the corresponding bit in the IRQA mask register HIGH.

7	6	5	4	3	2	1	0
1	T	U	R	F	N	0	P

1 = always active bit

T = 2MHz timer 1, rising edge triggered

U = 2MHz timer 0, rising edge triggered

R = power on reset

F = FLYBACK, rising edge triggered

N = **nINT1**, falling edge triggered

P = **INT2**, rising edge triggered

Write	ignored
Read	status bit[7] is always 1 bits[6:2,0] 0: not triggered since last cleared, 1: triggered since last cleared bit[1] is always 0
Reset	clear bits[6:5,3:2,0] to zero power on reset sets bit[4] to 1, push button reset maintains the current bit[4] value

13.3.6 IRQRQA (0x14) - IRQ A interrupts request/clear

7	6	5	4	3	2	1	0
1	T	U	R	F	N	X	P

1 = always active bit

T = 2MHz timer 1, rising edge triggered

U = 2MHz timer 0, rising edge triggered

R = power on reset

F = FLYBACK, rising edge triggered

N = **nINT1**, falling edge triggered

P = **INT2**, rising edge triggered

Write	clear triggered interrupts 0: don't clear interrupt 1: clear interrupt
Read	requests, as status, but bitwise ANDed with mask

13.3.7 IRQMSKA (0x18) - IRQ A interrupts mask

7	6	5	4	3	2	1	0
1	T	U	R	F	N	0	P

1 = always active bit

T = 2MHz timer 1, rising edge triggered

U = 2MHz timer 0, rising edge triggered

R = power on reset

F = FLYBACK, rising edge triggered

N = **nINT1**, falling edge triggered

P = **INT2**, rising edge triggered

Write	set mask for each interrupt source 0: don't form part of nIRQ 1: form part of nIRQ
Read	value set by write
Reset	set all zeros (none affect nIRQ)

13.3.8 SUSMODE (0x1C) - SUSPEND mode

This register allows the CPU to set the ARM7500 into SUSPEND mode. Only one bit (0) is used, and writing to this bit will cause SUSPEND mode to be entered. The value written to bit 0 determines whether the external I/O clocks, normally output from the chip, are also disabled during SUSPEND mode. The value programmed will depend on the nature of the peripherals being driven by those clocks.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	S

S = SUSPEND mode control of external I/O clocks

Write	turn off external I/O clocks when in this mode
	0: turn off, 1: don't turn off
	Enter SUSPEND mode with MCLK,FCLK,I/O clocks and some internal clocks stopped. DMA continues and the write to this location completes on either wakeup event, nIRQ or nFIQ or reset.
Read	return above value
Reset	set to zero

13.3.9 IRQSTB (0x20) - IRQ B interrupts status

7	6	5	4	3	2	1	0
K	J	P	T	I	S	C	F

K = keyboard receive interrupt

J = keyboard transmit interrupt

P = **nINT3**, active LOW

T = **nINT4**, active LOW

I = **INT5**, active HIGH

S = **nINT6**, active LOW

C = **INT7**, active HIGH

F = **nINT8**, active LOW

Write	ignored
Read	status 0: inactive, 1: active

13.3.10 IRQRQB (0x24) - IRQ B interrupts request

7	6	5	4	3	2	1	0
K	J	P	T	I	S	C	F

K = keyboard receive interrupt

J = keyboard transmit interrupt

P = **nINT3**, active LOW

T = **nINT4**, active LOW

I = **INT5**, active HIGH

S = **nINT6**, active LOW

C = **INT7**, active HIGH

F = **nINT8**, active LOW

Write	ignored
Read	request, status bitwise ANDed with mask

13.3.11 IRQMSKB (0x28) - IRQ B interrupts mask

7	6	5	4	3	2	1	0
K	J	P	T	I	S	C	F

K = keyboard receive interrupt

J = keyboard transmit interrupt

P = **nINT3**, active LOW

T = **nINT4**, active LOW

I = **INT5**, active HIGH

S = **nINT6**, active LOW

C = **INT7**, active HIGH

F = **nINT8**, active LOW

Write	set mask for each interrupt source 0: don't form part of nIRQ 1: form part of nIRQ
Read	value set by write
Reset	set all zeros (none affect nIRQ)

13.3.12 STOPMODE (0x2C) - STOP mode

This register exists only as an address decode and is used to enter STOP mode. It is very important that DMA activity is stopped before this register is written to. The value written to the register will be permanently forced out on the main data bus during STOP mode, and for most systems it will be desirable to ensure that this value is 0xFFFFFFFF. The address bus will automatically be forced HIGH during STOP mode.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X

Write	(any value), enter STOP mode with OSCPOWER set low. The write to this register completes on either wakeup event, **nEVENT**, **nEVENT2**, or reset

Read ignored

13.3.13 FIQST (0x30) - FIQ interrupts status

The FIQ control registers take a similar form to the IRQ registers previously described. Again, bit 7 is always active so that a FIQ interrupt can be forced via software

7	6	5	4	3	2	1	0
1	F	0	S	0	0	I	D

1 = always active

F = **nINT8**, active LOW

S = **nINT6**, active LOW

I = **INT5**, active HIGH

D = **INT9**, active HIGH

Write ignored

Read status
0: inactive, 1: active

13.3.14 FIQRQ (0x34) - FIQ interrupts request

7	6	5	4	3	2	1	0
1	F	0	S	0	0	I	D

1 = always active

F = **nINT8**, active LOW

S = **nINT6**, active LOW

I = **INT5**, active HIGH

D = **INT9**, active HIGH

Write ignored

Read request, status bitwise ANDed with mask

13.3.15 FIQMSK (0x38) - FIQ interrupts mask

7	6	5	4	3	2	1	0
1	F	0	S	0	0	I	D

1 = always active

F = **nINT8**, active LOW

S = **nINT6**, active LOW

I = **INT5**, active HIGH

D = **INT9**, active HIGH

Write	set mask for each interrupt source 0: don't form part of nFIQ, 1: form part of nFIQ
Read	value set by write
Reset	set all zeros (none affect nFIQ)

13.3.16 CLKCTL (0x3C) - Clock control

On system power up, the clock control register will be reset such that all three main clocks have a divide by 2 prescale at the inputs to the chip. This register will sometimes need to be reprogrammed as part of the initial tasks of the operating system, to set the prescalers into divide by 1 mode. Divide-by-2 mode allows faster oscillators to be used with less rigorous mark-space requirements.

7	6	5	4	3	2	1	0
X	X	X	X	X	F	M	I

F = FCLK divide control

M = MEMRFCK divide control

I = I/O clock divide control

Write	bit[2] 0: FCLK x 2 = **CPUCLK**, 1: FCLK = **CPUCLK** bit[1] 0: MEMRFCK x 2 = **MEMCLK**, 1: MEMRFCK = **MEMCLK** bit[0] 0: IOCK32 x 2 = **I_OCLK**, 1: IOCK32 = **I_OCLK**
Read	return above value

Power On Reset only

set all to zero, i.e. divide by 2 clocks

Push button reset does not affect this register

13.3.17 T0LOW (0x40) - Timer 0 LOW bits

There are eight registers associated with the two 16-bit timers in ARM7500.

7	6	5	4	3	2	1	0
L	L	L	L	L	L	L	L

L = LOW byte of timer

Write	set LOW byte latch value which is loaded into timer when it reaches end count
Read	read value of LOW count latched by the 'Latch' command T0LAT

13.3.18 T0HIGH (0x44) - Timer 0 HIGH bits

7	6	5	4	3	2	1	0
H	H	H	H	H	H	H	H

H = High byte of timer

Write	set HIGH byte latch value which is loaded into timer when it reaches end count
Read	read value of HIGH count latched by the 'Latch' command T0LAT

13.3.19 T0GO (0x48) - Timer 0 Go command

Write	(value ignored) load counter with HIGH and LOW latch values and start decrementing
Read	ignored

13.3.20 T0LAT (0x4C) - Timer 0 Latch command

Write	(value ignored) latch timer value in HIGH and LOW count latches
Read	ignored

13.3.21 T1LOW (0x50) - Timer 1 LOW bits

7	6	5	4	3	2	1	0
L	L	L	L	L	L	L	L

L = LOW byte of timer

Write	set LOW byte latch value which is loaded into timer when it reaches end count
Read	read value of LOW count latched by the 'Latch' command T1LAT

13.3.22 T1HIGH (0x54) - Timer 1 HIGH bits

7	6	5	4	3	2	1	0
H	H	H	H	H	H	H	H

H = HIGH byte of timer

Write	set HIGH byte latch value which is loaded into timer when it reaches end count
Read	read value of HIGH count latched by the 'Latch' command T1LAT

13.3.23 T1GO (0x58) - Timer 1 Go command

Write	(value ignored) load counter with HIGH and LOW latch values and start decrementing
Read	ignored

13.3.24 T1LAT (0x5C) - Timer 1 Latch command

Write	(value ignored) latch timer value in HIGH and LOW count latches
Read	ignored

13.3.25 IRQSTC (0x60) - IRQ C interrupts status

The IRQC set of control registers control the effect of the **IOP[7:0**] I/O port bits on the main interrupts. Their functionality is identical to that described for the previous IRQB set

7	6	5	4	3	2	1	0
I	I	I	I	I	I	I	I

I = **IOP[7:0]** pins, active LOW

Write	ignored
Read	status 0: inactive, 1: active

13.3.26 IRQRQC (0x64) - IRQ C interrupts request

7	6	5	4	3	2	1	0
I	I	I	I	I	I	I	I

I = **IOP[7:0]** pins, active LOW

Write	ignored
Read	request, status bitwise ANDed with mask

13.3.27 IRQMSKC (0x68) - IRQ C interrupts mask

7	6	5	4	3	2	1	0
I	I	I	I	I	I	I	I

I = **IOP[7:0]** pins, active LOW

Write	set mask for each interrupt source 0: don't form part of nIRQ, 1: form part of nIRQ
Read	value set by write
Reset	set all zeros (none affect nIRQ)

13.3.28 VIDMUX (0x6C) - Video LCD and Serial Sound Mux control

This register has two functions. Bit 0 allows selection of the type of serial sound interface to be supported. The timing of the two possibilities is shown in the Sound Features chapter. Bit 1 is used to control the colour LCD multiplexer which is used in conjunction with the video pixel clock to double the available bandwidth of colour LCD data provided by the ARM7500. Further details of how to use this feature can be found in the video and sound macrocell chapters.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	I	L

L = colour LCD support Mux control

I = Serial Sound Format selection

Write	bit[0] 0: ESEL[0] = EREG[0], 1: ESEL[0] = **ECLK** bit[1] 0: normal format, 1: Japanese format
Read	return above value
Reset	set to zero (normal)

13.3.29 IRQSTD (0x70) - IRQ D interrupts status

The IRQD control registers are used in an identical way to the IRQB and C registers which have already been described.

7	6	5	4	3	2	1	0
X	X	X	2	1	A	T	R

2 = **nEVENT2**, active LOW wakeup event 2

1 = **nEVENT1**, active LOW wakeup event 1

A= A to D, active HIGH

T = mouse transmit active HIGH

R = mouse receive active HIGH

Write	ignored
Read	status bits[7:5] unused

bits[4:0] 0: inactive, 1: active

13.3.30 IRQRQD (0x74) - IRQ D interrupts request

7	6	5	4	3	2	1	0
X	X	X	2	1	A	T	R

2 = **nEVENT2**, active LOW wakeup event 2

1 = **nEVENT1**, active LOW wakeup event 1

A = A to D, active HIGH

T = mouse transmit active HIGH

R = mouse receive active HIGH

Write ignored

Read request, status bitwise ANDed with mask

13.3.31 IRQMSKD (0x78) - IRQ D interrupts mask

7	6	5	4	3	2	1	0
X	X	X	2	1	A	T	R

2 = **nEVENT2**, active LOW wakeup event 2

1 = **nEVENT1**, active LOW wakeup event 1

A = A to D, active HIGH

T = mouse transmit active HIGH

R = mouse receive active HIGH

Write set mask for each interrupt source
0: don't form part of nIRQ, 1: form part of nIRQ

Read value set by write

Reset set all zeros (none affect nIRQ)

13.3.32 ROMCR0,1 (0x80,0x84) - ROM control

The ROM interface is very flexible, allowing the length of non sequential and burst cycles to be programmed. These two registers allow this programming to take place. The half speed select bit is included to enable the interface to be used with slow ROMs when fast DRAM is being used and the memory system clock is running at a higher frequency. In this case the half speed bit will double the length of all the timings and will allow the ROM interface to still function correctly with slower ROMs. Each register also contains a bit (6) which when set will allow a 16-bit wide ROM device to be used for that bank, by performing two 16-bit fetches to form the 32-bit word required by the ARM7500.

7	6	5	4	3	2	1	0
X	S	H	B	B	N	N	N

N = non-sequential access time:

000 = 7 **MEMCLK** cycles

001 = 6 **MEMCLK** cycles

010 = 5 **MEMCLK** cycles

011 = 4 **MEMCLK** cycles

100 = 3 **MEMCLK** cycles

101 = 2 **MEMCLK** cycles

B = burst mode access time

00 = Burst Off

01 = 4 **MEMCLK** cycles

10 = 3 **MEMCLK** cycles

11 = 2 **MEMCLK** cycles

H = half speed select, i.e. double above delays when H=1

S = 16/32-bit mode

Write	bit[6] 0: 32-bit, 1: 16-bit bit[5] 0: half speed mode, 1: normal speed
Read	return above values
Reset	set to 0x40, i.e. 16-bit, slowest access time to ensure all systems can always be started up

13.3.33 REFCR (0x8C) - Refresh period

This register programs the DRAM refresh period. It is set to the fastest available rate during reset, as refresh continues during reset to ensure that the requirements of DRAM specification can be fully met.

7	6	5	4	3	2	1	0
X	X	X	X	R	R	R	R

R = refresh period

Write	bit[3:0] 0000: refresh off 0001: 16us

	0010: 32us
	0100: 64us
	1000: 128us
	all others undefined
Read	return above values
Reset	set to 0001 (fastest available refresh rate)

13.3.34 ID0 (0x94) - Chip ID number LOW byte

The ID registers and the version register read back the ARM7500 ID and version numbers. These registers are read only and must NOT be written to, as they are used to set the ARM7500 into special modes during production test.

7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0

Write	do not write to this location
Read	LOW byte of chip ID: 0x98

13.3.35 ID1 (0x98) - Chip ID number HIGH byte

7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	1

Write	do not write to this location
Read	HIGH byte of chip ID: 0x5B

13.3.36 VERSION (0x9C) - Chip version number

Write	ignored
Read	chip version number byte

13.3.37 MSEDAT (0xA8) - Mouse data

The Mouse data and control registers are identical to the keyboard data and control registers, and are written to and read from in exactly the same way.

13.3.38 MSECR (0xAC) - Mouse control

As KBDCR register

13.3.39 IOTCR (0xC4) - I/O timing control

This register sets up the cycle types for two areas of I/O space.

7	6	5	4	3	2	1	0
X	X	X	X	C	C	S	S

C = combo area access speed

S = **NPCCS1/2** area access speed

Write	bits[7:4] reserved bits[3:2] 00: Type A (slowest), 01: Type B, 10: Type C, 11: Type D (fastest) bits[1:0] 00: Type A (slowest), 01: Type B, 10: Type C, 11: Type D (fastest).
Read	read back above values

13.3.40 ECTCR (0xC8) - I/O Expansion card timing control

This register sets up the access speed for eight portions of extended address space within the area of I/O space from 08FFFFFF to 0FFFFFFF. Only types A and C are available.

7	6	5	4	3	2	1	0
E	E	E	E	E	E	E	E

E = expansion card area access speed

Write	bit[7] (0F00 0000 -> 0FFF FFFF), 0: Type A, 1: Type C bit[0] (0800 0000 -> 08FF FFFF), 0: Type A, 1: Type C
Read	read back above values

13.3.41 ASTCR (0xCC) - I/O Asynchronous timing control

This register is used for the situation where I/O is being used with a very fast memory system clock. Normally it will always be programmed to zero to give the minimum delay for these cycles, however in some configurations it may be necessary to program the register bit to one to slow down the internal synchronisation between I/O clocks and memory clocks and thus ensure sufficient address hold time for the I/O address.See *Appendix C: Using the ASTCR register at high MEMCLK frequencies*.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	A

A = asynchronous timing control

0: minimal delay to I/O cycles

1: wait states to ensure address hold time

13.3.42 DRAMWID (0xD0) - DRAM width control

This register selects between 16 and 32-bit modes of operation for each of the four available banks of DRAM. Each bank can be individually selected for 16 or 32-bit operation. This would allow a mixed 16/32-bit wide system to be built.

7	6	5	4	3	2	1	0
X	X	X	X	S	S	S	S

S = 16/32-bit mode select, one for each bank

Write	bit[3] bank 3 DRAM width, 0: 32-bit, 1: 16-bit bit[2] bank 2 DRAM width, 0: 32-bit, 1: 16-bit bit[1] bank 1 DRAM width, 0: 32-bit, 1: 16-bit bit[0] bank 0 DRAM width, 0: 32-bit, 1: 16-bit
Read	reads above values
Reset	set bits to zero (32-bit)

13.3.43 SELFREF (0xD4) - DRAM self-refresh control

Direct software control of the external NRAS[3:0] and NCAS[3:0] lines is provided by this register. This is intended for use with self refresh DRAM, so that before the ARM7500 is forced into STOP mode, the banks of DRAM can be set into a self refresh state from software by forcing the NRAS and NCAS lines as specified in the DRAM data sheet.

7	6	5	4	3	2	1	0
C	C	C	C	R	R	R	R

C = force NCAS's LOW

R = force NRAS's LOW

Write	bits[7:4] 0: normal, 1: force to zero bits[3:0] 0: normal, 1: force to zero
Read	reads above values
Reset	set bits to zero (normal)

13.3.44 ATODICR (0xE0) - A to D interrupt control

The A to D convertor interface is designed such that various combination of interrupts from the channels can be used to generate an interrupt request in the IRQD interrupt request register. It should be noted that the logical OR of all four basic enables is used to power up the comparators. As the comparators consume static current, they must be powered down by disabling all the A to D channels using this register before STOP mode is entered.

7	6	5	4	3	2	1	0
S	F	A	C	4	3	2	1

1 = channel 1 interrupt enable

2 = channel 2 interrupt enable

3 = channel 3 interrupt enable

4 = channel 4 interrupt enable

C = any combination of channels generates nIRQ

A = only all channels enabled generates nIRQ

F = first pair enabled generates nIRQ

S = second pair enabled generates nIRQ

Write	bit[7:0] 0: disabled, 1: enabled
Read	return above values
Reset	reset to 0x0F

Note: *The OR of bit[3:0] is used to power up all the comparators. Thus they reset to the powered up state.*

13.3.45 ATODSR (0xE4) - A TO D status

This register shows which of the A TO D channels have been triggered and can have their counters read to ascertain the analog value at the input to the channel. The interrupt request status bits are generated from the stop flags logically ANDed with the interrupt enables from the interrupt control register.

7	6	5	4	3	2	1	0
R	R	R	R	S	S	S	S

R[3:0] = interrupt request state for channels 4 to 1

S[3:0] = stop flag for channels 4 to 1

Write	ignored
Read	bit[7:4] 0: not requesting, 1: requesting
	bit [3:0] 0: not stopped, 1: stopped
Reset	set all zero (not requesting or stopped)

13.3.46 ATODCC (0xE8) - A to D convertor control

The lower 4 bits of this register directly reset each of the four counters, so that they can be set back to zero before a new analog to digital conversion cycle takes place. The counter will start counting as soon as the relevant clear bit is set back to zero. The discharge transistor controls causes the channel comparator input to be pulled firmly down to Vss, thus discharging an external capacitor and ensuring zero volts across the capacitor until the discharge bit is programmed LOW again. With the system connected as it is expected to be used, the external capacitor will begin charging as soon as the discharge bit is reset, so it is expected that the discharge bit would be reset at the same time as the counter clear bit for that channel is re-enabled.

7	6	5	4	3	2	1	0
D	D	D	D	C	C	C	C

D[3:0] = discharge transistor control for channels 4 to 1

C[3:0] = clear counter for channels 4 to 1

Write	bit[7:4] 0: transistor off, 1: transistor on (discharge) bit[3:0] 0: clear counter, 1: enable counter
Read	return above values
Reset	set all zero (clear counters and don't discharge)

13.3.47 ATODCNT1 (0xEC) - A to D counter 1

Write	ignored
Read	returns 16-bit counter value

13.3.48 ATODCNT2 (0xF0) - A to D counter 2

Write	ignored
Read	returns 16-bit counter value

13.3.49 ATODCNT3 (0xF4) - A to D counter 3

Write	ignored
Read	returns 16-bit counter value

13.3.50 ATODCNT4 (0xF8) - A to D counter 4

Write	ignored
Read	returns 16-bit counter value

13.3.51 SDCURA (0x180) - Sound DMA Current A

The operation of the sound DMA channel is described in the Memory Subsystems chapter. The sound current registers are programmed with a page address and the offset within that page to describe the precise location of the first DMA fetch. The value in the register is then increased by 16 following each DMA access.

31 – 29	28 – 12	11 – 4	3 – 0
X X X	P P P P P P P P P P P P P P P P P	F F F F F F F F	0 0 0 0

P = page[16:0]

F = offset[11:0]

Write	bits[31:29] unused
bits[28:12] page of next DMA fetch
bits[11:4] offset within page of next DMA fetch
bits[3:0] ignored

Read	bits[31:29] undefined
bits[28:4] current DMA fetch location
bits[3:0] always zero

13.3.52 SDENDA (0x184) - Sound DMA End A

This register should be programmed with the offset within the page of the final quad word. Bit 30 should always be programmed to zero unless the channel is being initialised for a single transfer in which case it must be programmed HIGH.

31	30	29 – 12	11 – 4	3 – 0
S	L	X X X X X X X X X X X X X X X X X X	E E E E E E E E	0 0 0 0

S = stop bit

L = last bit

E = end[11:0]

Write	bit[31] stop bit, 0: don't stop after reaching End, 1: do stop
bit[30] last bit, 0: not last transfer, 1: last quad word transfer
bits[11:4] last DMA location within page selected
bits[3:0] ignored

Read	bits[31:30,11:4] value written
bits[3:0] always zero

13.3.53 SDCURB (0x188) - Sound DMA Current B

The 'B' pair of registers for the sound DMA channel are used in exactly the same way as the 'A' pair, to enable DMA to continue from the page addressed by one set of registers while the other set are being reprogrammed.

31 29	28 12	11 4	3 0
X X X	P P P P P P P P P P P P P P P P P	F F F F F F F F	0 0 0 0

P = page[16:0]

F = offset[11:0]

Write	bits[31:29] unused bits[28:12] page of next DMA fetch bits[11:4] offset within page of next DMA fetch bits[3:0] ignored
Read	bits[31:29] undefined bits[28:4] current DMA fetch location bits[3:0] always zero

13.3.54 SDENDB (0x18C) - Sound DMA End B

This register is used in the same way as the SDENDA register.

31	30	29 12	11 4	3 0
S	L	X X X X X X X X X X X X X X X X X X	E E E E E E E E	0 0 0 0

S = stop bit

L = last bit

E = end[11:0]

Write	bit[31] stop bit, 0: don't stop after reaching end, 1: do stop bit[30] last bit, 0: not last transfer, 1: last quad word transfer bits[11:4] last DMA location within page selected bits[3:0] ignored
Read	bits[31:30,11:4] value written bits[3:0] always zero

13.3.55 SDCR (0x190) - Sound DMA control

This register controls the sound DMA channel and its state machine. Only two bits can be written to: bit 7 clears the state machine into a state where it has overrun and is requesting an interrupt. Bit 6 enables the sound DMA channel.

7	6	5	4	3	2	1	0
C	0	E	1	0	0	0	0

C = clear

E = enable

Write	bit[7] clear, 0: don't clear state machine, 1: clear state machine. Self clearing.
	bit[6] not used
	bit[5] enable, 0: disabled, 1: enabled
	bits[4:0] not used
Read	bit[7] always reads zero
	bit[6] always reads zero
	bit[5] enable, 0: disabled, 1: enabled
	bits[4:0] read as 10000 (binary), historically signifying a quad word transfer
Reset	enable set to zero

13.3.56 SDST (0x194) - Sound DMA status

The sound DMA status register shows the status of the state machine used to control sound DMA accesses. It cannot be written to.

7	6	5	4	3	2	1	0
X	X	X	X	X	O	I	W

O = overrun

I = interrupt request

W = A or B buffer indication

Write	ignored
Read	bits[7:3] unused
	bits[2:0] direct state machine state
Reset	set to 110 (binary)

13.3.57 CURSCUR (0x1C0) - Cursor DMA current

The cursor current register need not normally be written to as the value in the init register is transferred into it during the FLYBACK period. It is then updated automatically in quad word increments during DMA.

31 29	28 4	3 0
X X X	C C C C C C C C C C C C C C C C C C C C C C C C C	0 0 0 0

C = Current fetch location

Write bits[31:29] unused
bits[28:4] cursor current DMA fetch location
bits[3:0] ignored

Read bits[31:29] undefined
bits[28:4] cursor current DMA fetch location
bits[3:0] always zero

13.3.58 CURSINIT (0x1C4) - Cursor DMA init

This register is written with the initial location of the cursor data buffer.

31 29	28 4	3 0
X X X	I I I I I I I I I I I I I I I I I I I I I I I I I	0 0 0 0

I = initial fetch location

Write bits[31:29] unused
bits[28:4] cursor initial DMA fetch location
bits[3:0] ignored

Read bit[31:29] undefined
bits[28:4] cursor initial DMA fetch location
bits[3:0] always zero

13.3.59 VIDCURB (0x1C8) - Duplex LCD video DMA current B

The 'B' video DMA address registers are for use with dual panel LCDs. The current registers do not normally need to be programmed as the value in the relevant INIT register is loaded into the current register during the FLYBACK period. This register gives the current location of the DMA data for the lower panel.

31 29	28 4	3 0
X X X	C C C C C C C C C C C C C C C C C C C C C C C C C	0 0 0 0

C = current fetch location B

Write bits[31:29] unused
bits[28:4] video current B DMA fetch location
bits[3:0] ignored

Read bits[31:29] undefined
bits[28:4] video current B DMA fetch location
bits[3:0] always zero

13.3.60 VIDCURA (0x1D0) - Video DMA current A

31 29	28 4	3 0
X X X	C C C C C C C C C C C C C C C C C C C C C C C C C	0 0 0 0

C = current fetch location A

Write bits[31:29] unused
bits[28:4] video current A DMA fetch location
bits[3:0] ignored

Read bits[31:29] undefined
bits[28:4] video current A DMA fetch location
bits[3:0] always zero

13.3.61 VIDEND (0x1D4) - Video DMA End

The video END register should be loaded with the address of the final quad word of the video frame buffer within memory

31 29	28 4	3 0
X X X	E E E E E E E E E E E E E E E E E E E E E E E E E	0 0 0 0

E = end location

Write bits[31:29] unused
bits[28:4] video end location
bits[3:0] ignored

Read bits[31:24] undefined
bits[23:4] video end location
bits[3:0] always zero

Note: *Although bits [28:4] can be written, only bits [23:4] are used internally.*

13.3.62 VIDSTART (0x1D8) - Video DMA Start

The video start register should be loaded with the location of the first quad word at the start of the video frame buffer. All the DMA control registers can only be loaded with quad word aligned values.

31–29	28–4	3–0
X X X	S S S S S S S S S S S S S S S S S S S S S S S S S	0 0 0 0

S = start location

Write bits[31:29] unused
bits[28:4] video DMA start fetch location
bits[3:0] ignored

Read bit[31:29] undefined
bits[28:4] video DMA start fetch location
bits[3:0] always zero

13.3.63 VIDINITA (0x1DC) - Video DMA init A

For normal CRT displays and single panel LCD data only the 'A' registers are used. The init register should be loaded with the address within the frame buffer of the first quad word to be displayed in the first raster at the top of the screen. In the case of dual panel displays, this register should be loaded with the address of the first quad word in the frame buffer to be displayed at the top left of the upper panel. The last bit (30) should only be set if the init A register has been programmed to the same value as the VIDEND register. The use of an init register allows hardware scrolling to be implemented by moving the position of the init register within the frame buffer.

31	30	29	28–4	3–0
X	L	E	I I I I I I I I I I I I I I I I I I I I I I I I I	0 0 0 0

I = initial fetch location A

Write bits[31,29] unused
bit[30] last bit, 0: not last fetch location, 1: last fetch location
bits[28:4] video initial A DMA fetch location
bits[3:0] ignored

Read bit[31] zero
bit[30] last bit, 0: not last fetch location, 1: last fetch location
bit[29] 'equal' - output of comparator
bits[28:4] video initial A DMA fetch location
bits[3:0] always zero

13.3.64 VIDCR (0x1E0) - Video DMA control

This register gives overall control for video DMA. Bit 7 selects between dual and single panel modes for LCD driving, and bit 5 enables video DMA. Note that for driving normal CRT displays, bit 7 should be set to zero.

7	6	5	4	3	2	1	0
D	1	E	1	0	0	0	0

D = dual panel mode

E = enable video/cursor DMA

Write	bit[7] 0: normal, 1: dual panel mode
	bit[6] ignored
	bit[5] 0: disable, 1: enable DMA
	bits[4:0] ignored
Read	bits[7,5] return above values
	bit[6] always read back one, DRAM mode
	bits[4:0] read as 10000 (binary), historically meaning quad word transfer
Reset	set to zero (disabled, normal mode)

13.3.65 VIDINITB (0x1E8) - Duplex LCD Video DMA init B

For normal CRT displays and single panel LCD data only the 'A' registers are used, and this register should be programmed with all zeros. In the case of dual panel displays, this register should be loaded with the address of the first quad word in the frame buffer to be displayed at the top left of the lower panel. The last bit (30) should only be set if the init B register has been programmed to the same value as the VIDEND register.

31	30	29	28 ... 4	3 ... 0
X	L	E	I I I I I I I I I I I I I I I I I I I I I I I I I	0 0 0 0

I = initial fetch location B

Write	bits[31,29] unused
	bit[30] last bit, 0: not last fetch location, 1: last fetch location
	bits[28:4] video initial B DMA fetch location
	bits[3:0] ignored
Read	bit[31] zero
	bit[30] last bit, 0: not last fetch location, 1: last fetch location
	bit[29] 'equal' - output of comparator
	bits[28:4] video initial B DMA fetch location
	bits[3:0] always zero

13.3.66 DMAST/DMARQ/DMAMSK (0x1F0,0x1F4,0x1F8) - DMA interrupt control

These three registers each contain only one bit relating to the status of the interrupt generated from the sound DMA state machine.

DMAST (0x1F0) - Sound DMA interrupt status.

7	6	5	4	3	2	1	0
X	X	X	S	X	X	X	X

S = sound interrupt status

Write	ignored
Read	status bits[7:5,3:0] unused bit[4] 0: inactive, 1: active

DMARQ (0x1F4) - Sound interrupt request.

7	6	5	4	3	2	1	0
X	X	X	S	X	X	X	X

S = sound interrupt request

Write	ignored
Read	request, status ANDed with mask bits[7:5,3:0] unused bit[4] 0: inactive, 1: active

DMAMSK (0x1F8) - Sound interrupt mask.

7	6	5	4	3	2	1	0
X	X	X	S	X	X	X	X

S = sound interrupt mask

Write	bits[7:5,3:0] unused bit[4] 0: don't affect nIRQ, 1: affect nIRQ
Read	mask bits[7:5,3:0] unused bit[4] read value written above

14 Memory Subsystems

This chapter describes the ROM and DRAM interfaces, and the DMA channels.

14.1 ROM interface

The ARM7500 ROM interface supports both non sequential and burst mode read cycles, with a range of programmable timings for each type. A single chip select signal **nROMCS** is generated for addresses between 0x00000000 and 0x01FFFFFF, which can be externally split to give separate chip selects for two 16MB banks of ROM. Each bank of ROM can be 16 or 32-bits wide. The ROM access time depends on the **MEMCLK** frequency, and to enable slow ROMs to be used with a high frequency **MEMCLK**, there is a half speed bit available which causes all ROM timings to take twice as many **MEMCLK** cycles.

Assuming a **MEMCLK** frequency of 32MHz, the access time for a non-sequential cycle can be varied from 220ns to 62.5ns in steps of 31.25ns. For burst mode cycles, the two LSBs of the latched address from ARM7500 are incremented to allow up to four sequential reads. The access time for burst mode cycles can be programmed from 125ns down to 62.5ns, again in steps of 31.25ns. If a frequency other than 32MHz is used for **MEMCLK**, these timings will scale accordingly.

Support for 16-bit wide ROMs is provided via a programmable bit in each of the ROM control registers. If a 16-bit wide device is selected, then two memory system cycles will be required to fetch the full 32-bit word required by the ARM. If burst mode is disabled for that bank, then ARM7500 will perform two non-sequential fetches using the programmed non-sequential timing, latch the intermediate 16-bit value, and present the full 32-bit word to the ARM processor macrocell.

If the burst mode timing bits are programmed into an enabled state, then the first 16-bit read will be a standard non-sequential cycle, but the second will be a burst mode cycle to minimise the total access time.

When a 16 bit wide ROM bank is being addressesd, the ROM address is shifted up by one bit such that the LSB appears on LA[2], thus allowing the same PCB layout to be used for 16 bit or 32 bit ROM banks.

When using a 16-bit wide ROM device, data must be stored such that the least significant bytes of a 32-bit word are stored at the lower memory address:

Contents

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0 0 0 0 0 0 0 1

When this is read, the ARM will see:

MSB ... LSB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

There are two identical registers which control the configuration and timing of the two ROM banks. Both registers default to 16-bit mode and the slowest possible non-sequential timings on reset, which means that one of the first actions when using 32-bit wide ROM must be to reprogram these registers for 32-bit wide operation. A detailed description of how to boot up an ARM7500 system using 32-bit wide ROM is contained in *Appendix A: Initialisation and Boot Sequence*.

To program these registers, write a byte to 0x03200080 for the ROMCR0 register (address range 0x00000000 to 0x00FFFFFF) or to 0x03200084 for the ROMCR1 register (address range 0x01000000 to 0x0FFFFFFF). The details of these registers are shown below.

7	6	5	4	3	2	1	0
X	S	H	B	B	N	N	N

N = non-sequential access time:

000 = 7 **MEMCLK** cycles
001 = 6 **MEMCLK** cycles
010 = 5 **MEMCLK** cycles
011 = 4 **MEMCLK** cycles
100 = 3 **MEMCLK** cycles
101 = 2 **MEMCLK** cycles

B = burst mode access time

00 = Burst Off
01 = 4 **MEMCLK** cycles
10 = 3 **MEMCLK** cycles
11 = 2 **MEMCLK** cycles

H = half speed select, i.e. double above cycle time when H=1

S = 16/32-bit mode

Write	bit[6] 0: 32-bit, 1: 16-bit bit[5] 0: half speed mode, 1: normal speed
Read	return above values
Reset	set to 0x40, ie 16-bit, slowest access time

Figure 14-1: ROM access timing without burst mode (32-bit mode) on page 14-4 shows the timing of non-sequential and sequential 32-bit ROM accesses without burst mode. *Figure 14-2: ROM access timing—burst mode (32 bit)* on page 14-5 shows the timing of non-sequential and sequential 32-bit ROM accesses with burst mode. *Figure 14-3: ROM access timing with burst mode—16-bit mode* on page 14-5 shows the timing of non-sequential and sequential 16-bit ROM accesses with burst mode. Note that all diagrams assume divide by 1 mode for **MEMCLK**.

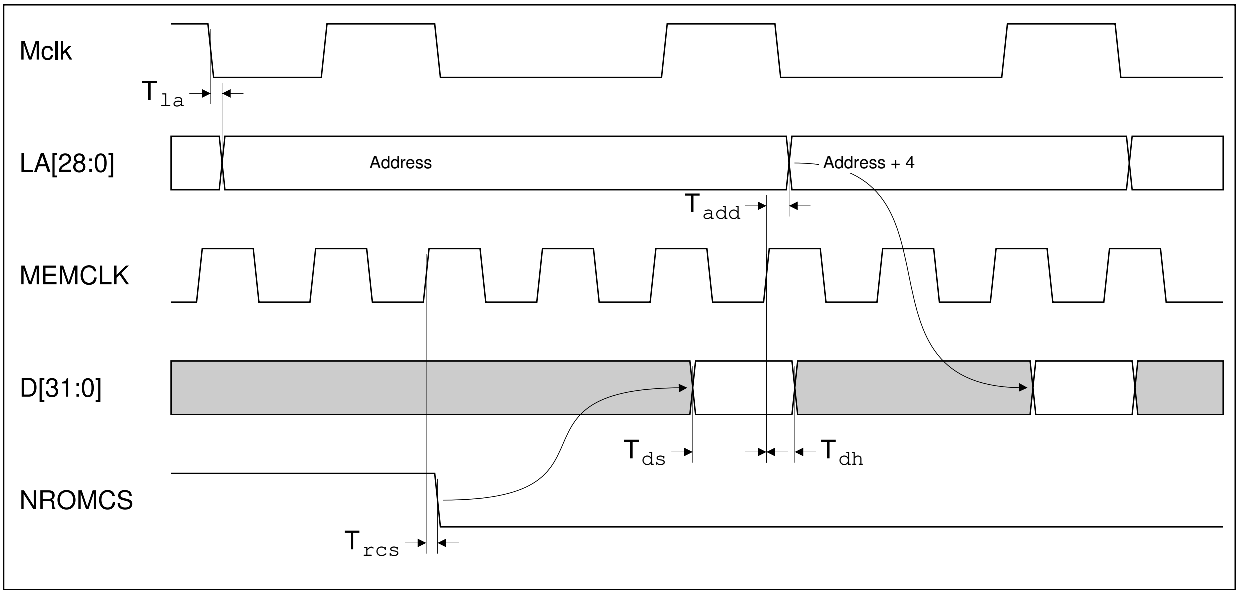


Figure 14-1: ROM access timing without burst mode (32-bit mode)

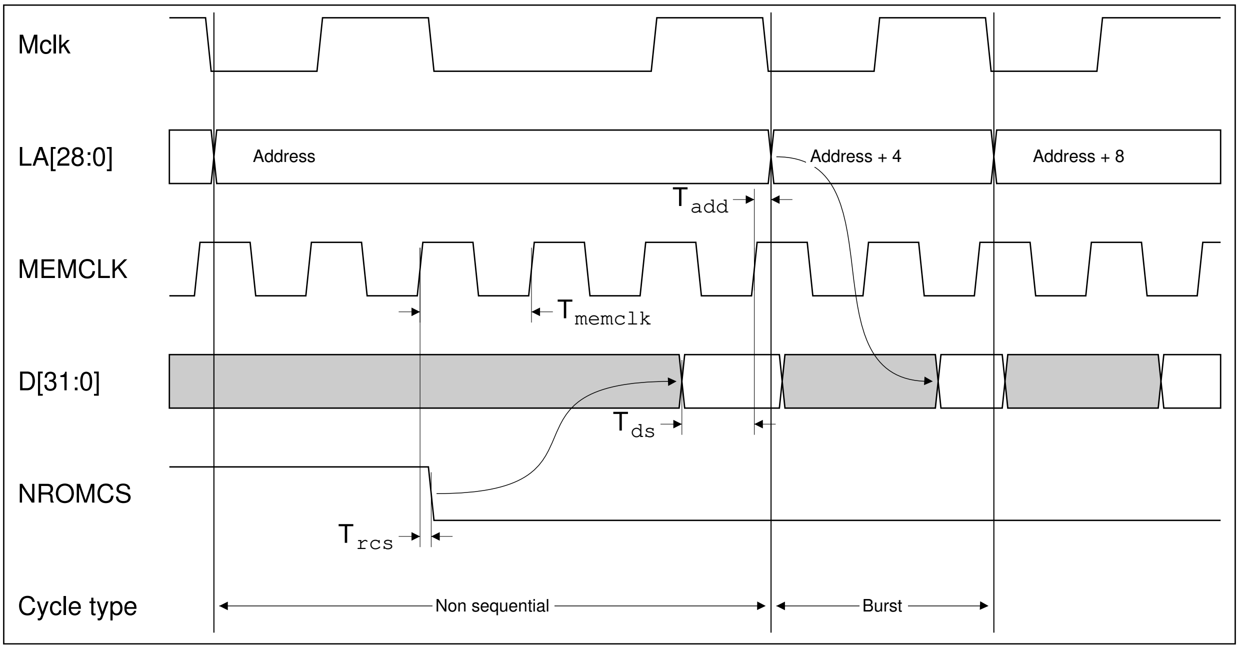


Figure 14-2: ROM access timing—burst mode (32 bit)

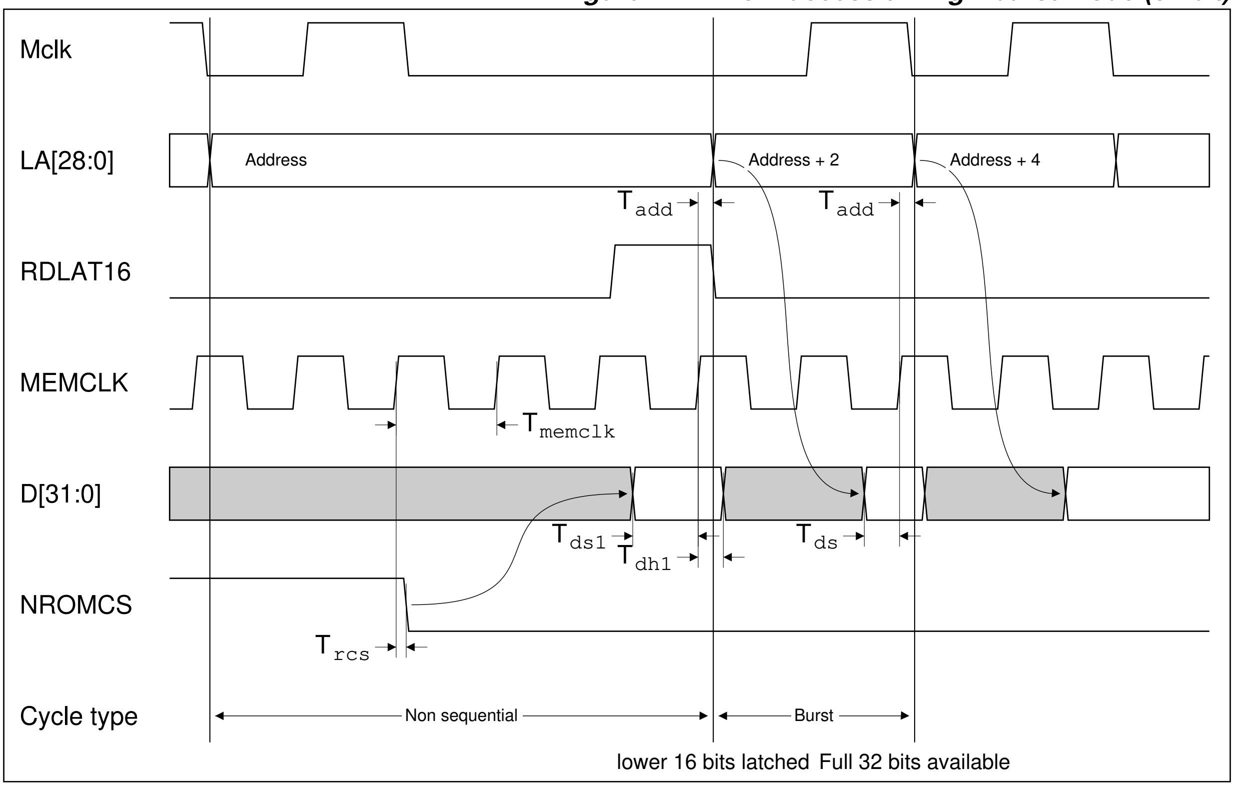


Figure 14-3: ROM access timing with burst mode—16-bit mode

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tadd	**MEMCLK** rising to **LA[28:0]** changing				ns	
Tmemclk	**MEMCLK** period		31.25		ns	assumes divide by 1 mode for **MEMCLK**
Tds	DATA setup to **MEMCLK** rising edge which creates falling **MCLK**				ns	
Tds1	DATA setup to **MEMCLK** rising edge which creates falling **RDLAT16**				ns	
Trcs	**MEMCLK** rising to **NROMCS** falling				ns	
Tdh	DATA hold from **MEMCLK** rising edge which creates falling **MCLK**				ns	
Tdh1	DATA hold from **MEMCLK** rising edge which creates falling **RDLAT16**				ns	
Tla	**LA[ ]** changing from **MEMCLK** rising which creates falling **MCLK**					

Table 14-1: ARM7500 ROM timing

14.2 DRAM interface

The DRAM interface can directly drive four banks of DRAM, using four **nRAS** strobes to select the bank, four **nCAS** strobes to select the byte within the word, and twelve multiplexed row/column address lines **RA[11:0]**, giving a maximum of 64MB in each DRAM bank. The **nRAS** strobes are decoded directly from bits 27 and 26 of the address, which means that the DRAM address space will be non-contiguous if the full 64MB is not used for each bank.

The DRAM controller supports page mode burst cycles with up to 255 sequential accesses in a burst. Each of the four banks can be a 16 or 32-bit wide device. Support is provided for CAS before RAS refresh, and direct programmability of the **nRAS** and **nCAS** outputs via a special register allows software to directly control self refresh DRAM.

DRAM cycle speed is controlled by the frequency of **MEMCLK**. Non-sequential DRAM cycles require five **MEMCLK** cycles, and sequential ones require two.

14.2.1 DRAM control registers

There are three registers associated with DRAM control. The DRAMWID register has four bits, one for each bank, and allows selection between 16 and 32-bit modes of operation for each bank. The SELFREF register allows direct forcing of the **nRAS** and **nCAS** outputs. The default state of each of these bits is zero, which allows normal operation of the **nRAS** and **nCAS** outputs. When a bit is set HIGH, however, the relevant **nCAS** or **nRAS** output is immediately forced active (LOW). The REFCR register controls the refresh rate for CAS before RAS refresh. There are four possible refresh periods from 128µs to 16µs.

14.2.2 DRAM address multiplexing

The multiplexing of the DRAM address onto the **RA[11:0]** outputs is slightly different for 32 and 16-bit modes. The DRAM address requested by the ARM or DMA controller must be shifted up by one bit in 16-bit mode, to enable two locations to be accessed to read or write one 32-bit word. The row/column address multiplexing arrangements are shown below, where the numbers in the table refer to the address bits provided by the ARM or DMA controller:

For a 32-bit wide DRAM bank:

RA[11:0]	11	10	9	8	7	6	5	4	3	2	1	0
Row address	24	22	19	18	17	16	15	14	13	12	11	10
Column address	25	23	21	20	9	8	7	6	5	4	3	2

For a 16-bit wide DRAM bank:

RA[11:0]	11	10	9	8	7	6	5	4	3	2	1	0
Row address	23	21	18	17	16	15	14	13	12	11	10	9
Column address	24	22	20	19	8	7	6	5	4	3	2	*

* This bit generated separately by DRAM controller to access each 16-bit half word in turn.

14.2.3 Selection between 16 and 32-bit DRAM

The DRAMWID register at address 0x032000D0 allows the width of each of the four DRAM banks to be defined for ARM7500. On reset, all banks are defined as 32 bits wide, so if a 16-bit system is being used it is necessary to program this register before any writes to DRAM occur. It is not possible to write to DRAM in 16-bit mode and read back from the same bank in 32-bit mode, or vice versa.

7	6	5	4	3	2	1	0
X	X	X	X	S	S	S	S

S = 16/32-bit mode select, one for each bank

Write	bit[3] bank 3 DRAM width, 0: 32-bit, 1: 16-bit bit[2] bank 2 DRAM width, 0: 32-bit, 1: 16-bit bit[1] bank 1 DRAM width, 0: 32-bit, 1: 16-bit bit[0] bank 0 DRAM width, 0: 32-bit, 1: 16-bit
Read	reads above values
Reset	set bits to zero (32-bit)

14.2.4 DRAM interface timing specification

32-bit mode

Figure 14-4: DRAM read timing (32-bit mode) , below, shows the timing of non sequential and sequential 32-bit DRAM read cycles. *Figure 14-5: DRAM write timing (32-bit mode)* on page 14-10 shows the timing of both types of 32-bit DRAM write cycles. Note that all timing diagrams assume divide by 1 is selected for **MEMCLK**.

In 32-bit mode, byte reads and writes have the same timing as word accesses, but only one **nCAS** output is selected according to the decode of bits 1 and 0 of the address.

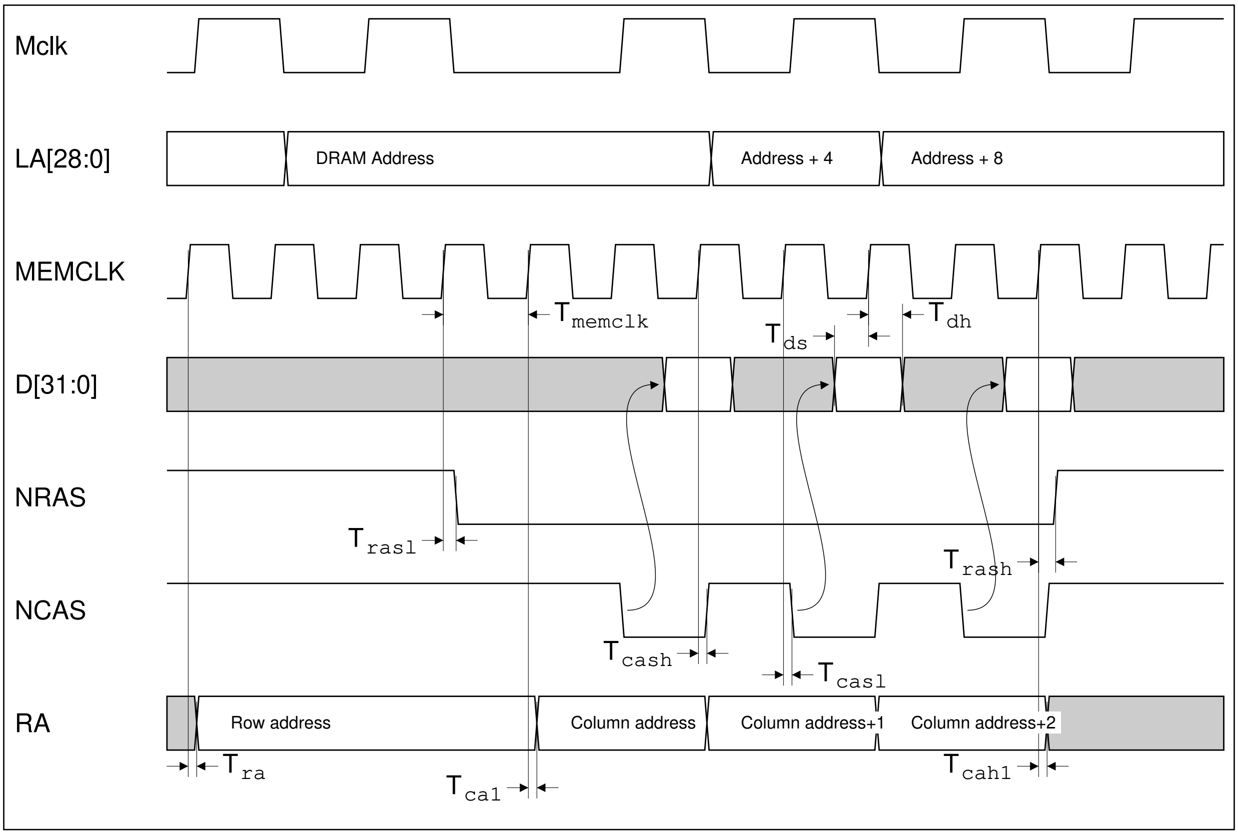


Figure 14-4: DRAM read timing (32-bit mode)

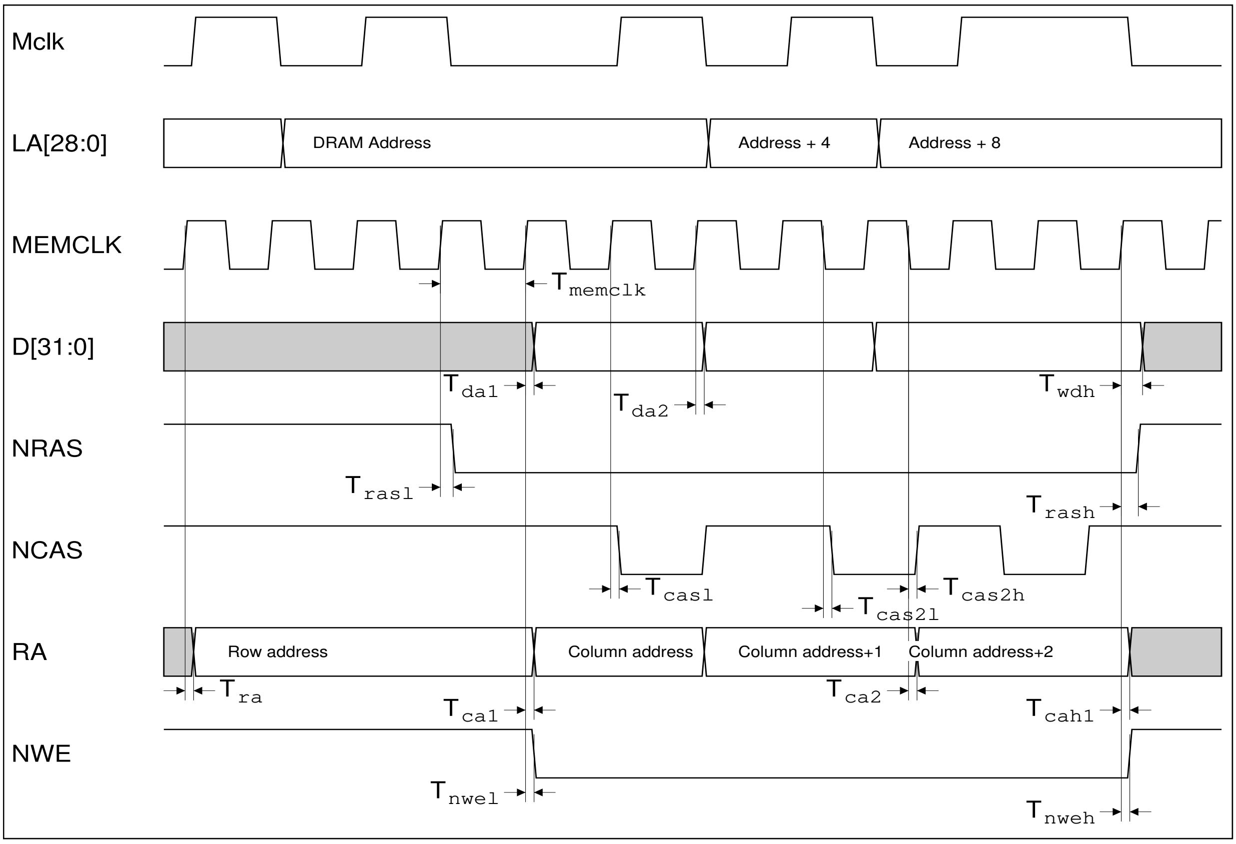


Figure 14-5: DRAM write timing (32-bit mode)

16-bit mode

In 16-bit mode ARM7500 must perform two reads or writes for each 32-bit word DRAM access requested by the ARM processor or the DMA controller. Only **nCAS[1]** and **nCAS[0]** are used, to access the two bytes of each word. **nCAS[3:2]** are held at logic ONE. In 16-bit mode, the same number of physical addresses are available as for 32-bit mode, which means that only 32MB of DRAM is supported per bank. Words are stored in DRAM with the upper half-word at the lower address:

Contents: 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1 0 0 0 0 0 0 0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1 0 0 0 0 0 0 1

When this is read, the ARM will see:

MSB LSB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

In 16-bit mode, byte reads and writes only require a single DRAM access, and the LSB of the column address is decoded in conjunction with the **nCAS[1:0]** outputs to select a single byte from four. Byte reads and writes for 16-bit wide DRAM thus have the same timing as for the non-sequential 32-bit case as shown in Figures 5-4 and 5-5.

16-bit mode word accesses involve a non-sequential access for the upper half word, followed by a sequential access for the lower half word at the next memory location. A non sequential 16-bit mode word access thus requires 7 **MEMCLK** cycles, after which sequential accesses can continue until a page boundary is reached, taking 2 cycles each.

The timing diagram (*Figure 14-6: DRAM read timing (16-bit mode)*) below shows a 16-bit mode read cycle, and *Figure 14-7: DRAM write timing (16-bit mode)* on page 14-13 shows a 16-bit mode write cycle.

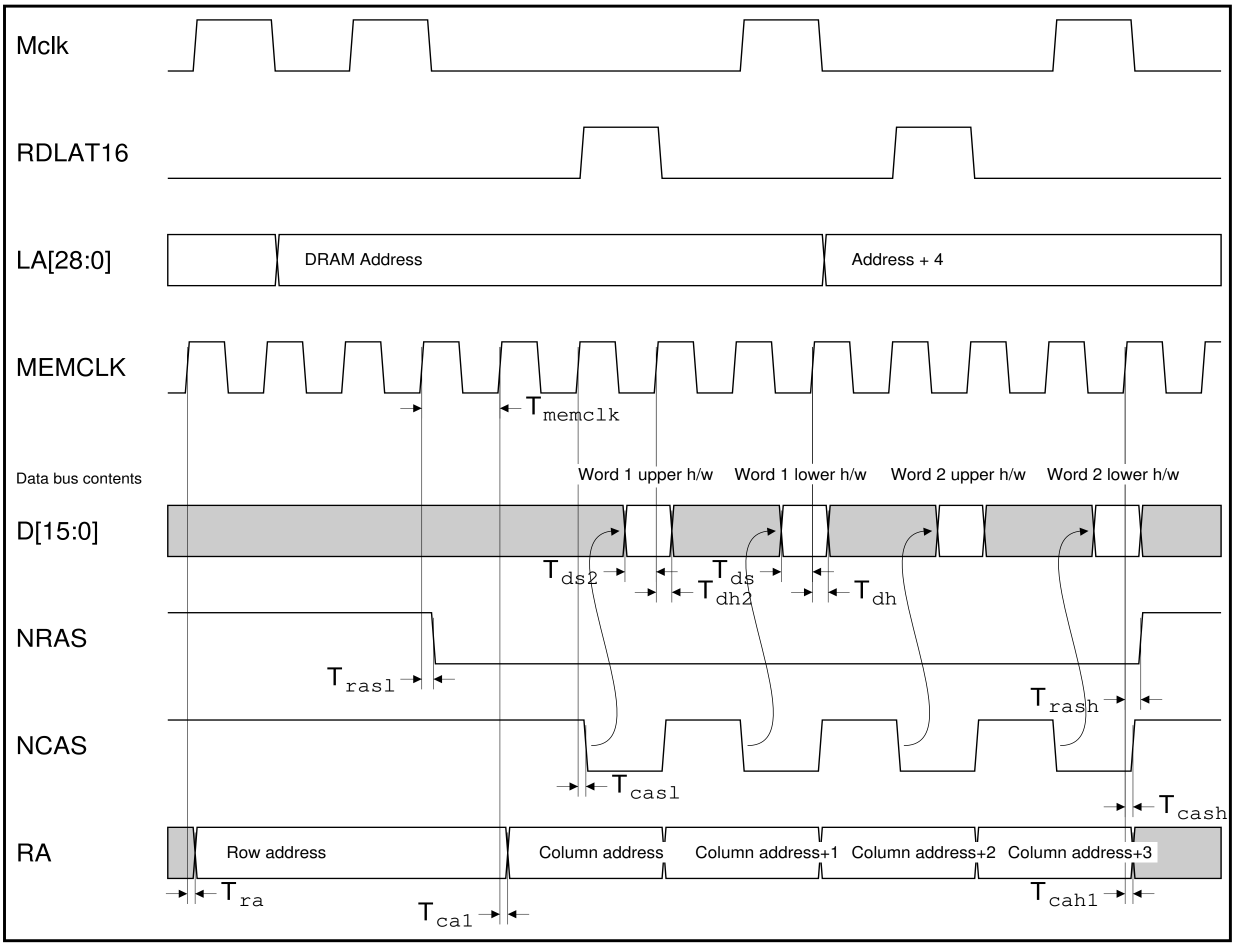


Figure 14-6: DRAM read timing (16-bit mode)

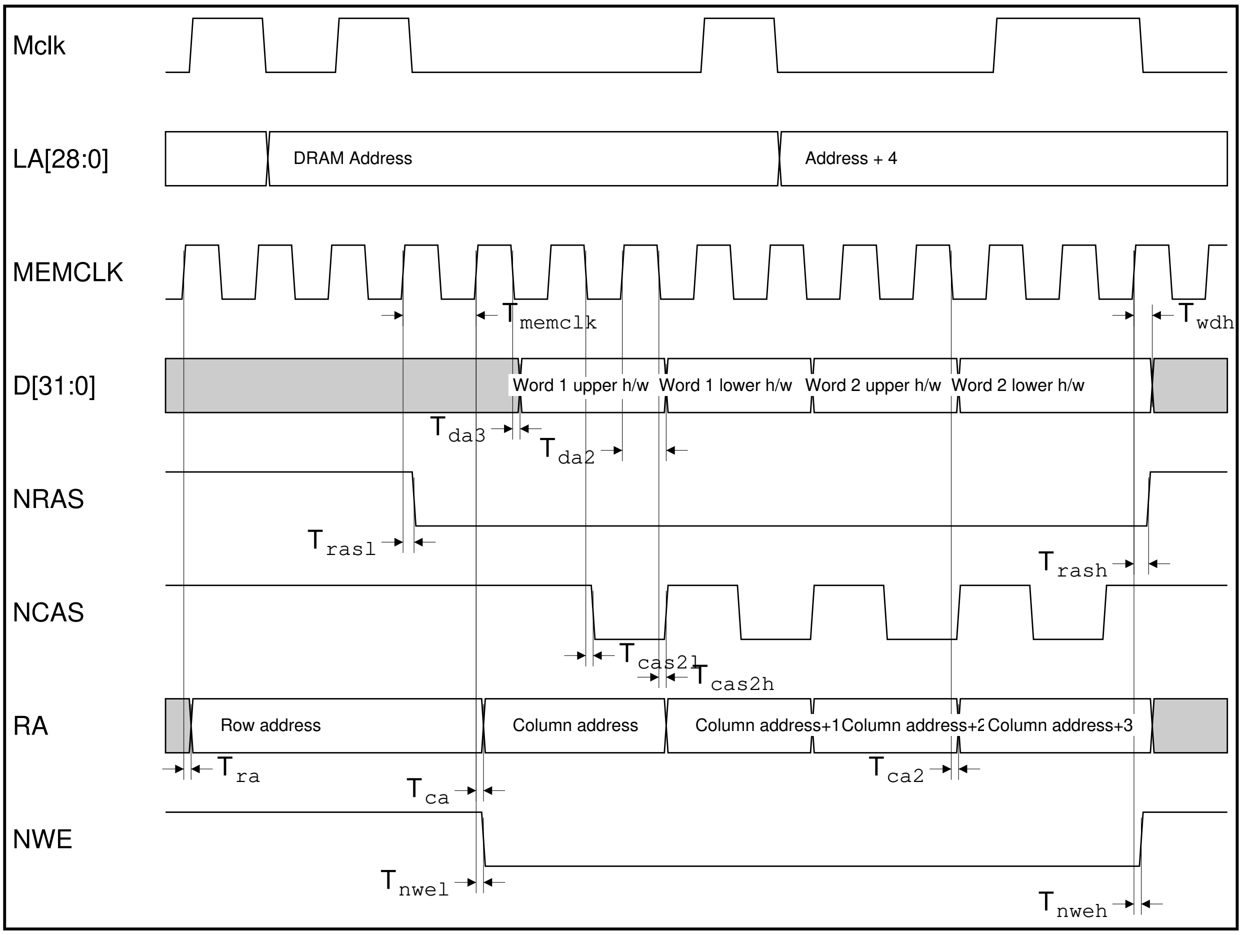


Figure 14-7: DRAM write timing (16-bit mode)

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tdh	DATA hold from **MEMCLK** rising edge which causes **Ncas[ ]** to rise				ns	
Tcasl	**MEMCLK** rising to **Ncas[ ]** falling				ns	
Tcash	**MEMCLK** rising to **Ncas[ ]** rising				ns	
Tmemclk	**MEMCLK** period		31.25		ns	assumes divide by 1 mode for **MEMCLK**
Tds	DATA setup to **MEMCLK** rising edge which causes **Ncas[ ]** to rise				ns	

Table 14-2: ARM7500 DRAM timing

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tda1	**MEMCLK** rising to write DATA valid				ns	
Tda2	**MEMCLK** rising to write DATA valid				ns	
Tda3	**MEMCLK** falling to DATA valid				ns	
Tds2	DATA setup to **MEMCLK** rising which creates RDLAT16 falling				ns	
Tdh2	DATA hold from **MEMCLK** rising which creates RDLAT16 falling				ns	
Twdh	DATA hold from **MEMCLK** rising edge which cases **NRAS[ ]** to rise				ns	
Trash	**MEMCLK** rising to **NRAS[ ]** rising				ns	
Trasl	**MEMCLK** rising to **NRAS[ ]** falling				ns	
Tra	**MEMCLK** rising to **RA[ ]** valid (row address)				ns	
Tca1	**MEMCLK** rising to **RA[ ]** valid (column address)				ns	
Tca2	as Tca1 but **MEMCLK** falling				ns	
Tcah1	column address, **RA[ ]**, hold from **MEMCLK** rising				ns	
Tnwel	**MEMCLK** rising to **NWE** falling				ns	
Tnweh	**MEMCLK** rising to **NWE** rising				ns	
Tcas2l	as Tcasl but **MEMCLK** falling				ns	
Tcas2h	as Tcash but **MEMCLK** falling				ns	

Table 14-2: ARM7500 DRAM timing

14.2.5 DRAM refresh

DRAM refresh is controlled by a small state machine and counter within ARM7500. The refresh interval timer is clocked by a clock derived from the fixed frequency **I_OCLK**, and thus the refresh intervals will remain the same even if the frequency of **MEMCLK** is increased for use with faster DRAM. There are four timings available for refresh, controlled by the REFCR refresh control register at address 0x0320008C. During reset, the refresh timer is reset to the fastest value (16µs), and the counter and state machine are clocked such that refresh continues even during reset.

7	6	5	4	3	2	1	0
X	X	X	X	R	R	R	R

R = refresh period

Write	bit[3:0] 0000: refresh off 0001: 16µs 0010: 32µs 0100: 64µs 1000: 128µs all others undefined
Read	return above values
Reset	set to 0001 (fastest available refresh rate)

The output states for DRAM refresh cycles are shown in *Figure 14-8: Refresh cycle timing* on page 14-15. Note that this assumes divide by 1 mode for **MEMCLK**.

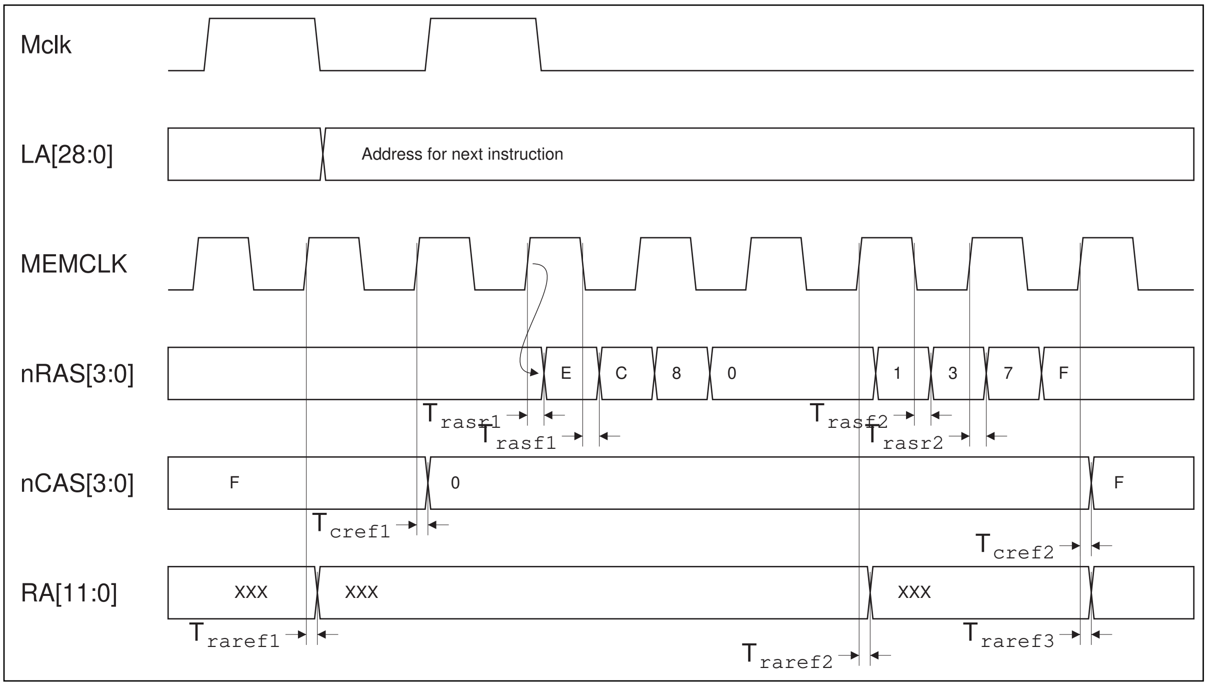


Figure 14-8: Refresh cycle timing

Symbol	Parameters	Min	Typ	Max	Units	Notes
Trasr1	**MEMCLK** rising to **nRAS** falling				ns	
Trasf1	**MEMCLK** falling to **nRAS** falling				ns	
Trasr2	**MEMCLK** rising to **nRAS** rising				ns	
Trasf2	**MEMCLK** falling to **nRAS** rising				ns	
Tcref1	**MEMCLK** rising to **nCAS[3:0]** falling				ns	
Tcref2	**MEMCLK** rising to **nCAS[3:0]** rising				ns	
Traref1	**MEMCLK** rising to **RA[11:0]** changing				ns	
Traref2	**MEMCLK** rising to **RA[11:0]** changing				ns	
Traref3	**MEMCLK** rising to **RA[11:0]** changing				ns	

Table 14-3: ARM7500 refresh cycle timing

14.2.6 DRAM self-refresh

The **nCAS** and **nRAS** lines can be forced active by programming bits in the SELFREF register at address 0x032000D4. This is intended for use with self refresh DRAM, and particularly in conjunction with STOP mode so that DRAM can retain state when all the ARM7500 clocks have been stopped. All DMA must be stopped and the code which writes to this register must be executing from ROM.

7	6	5	4	3	2	1	0
C	C	C	C	R	R	R	R

C = force **nCAS**'s LOW

R = force **nRAS**'s LOW

Write	bits[7:4] 0: normal, 1: force to zero bits[3:0] 0: normal, 1: force to zero
Read	reads above values
Reset	set bits to zero (normal)

14.3 DMA channels

The ARM7500 supports video, cursor and sound DMA to enable direct transfer of quad words of data from DRAM to the video and sound processing interfaces. All DMA is in units of four words (quad words) and data can be read from any of the four banks of DRAM in either 16 or 32-bit mode. ARM7500 contains a DMA Address Generator, which has a number of programmable control registers associated with each channel. Most of these registers contain 28-bit physical addresses. The DMA controller also includes support for DMA to dual panel LCD screens.

All three of the DMA channels have at least one CURRENT register which contains the address in memory of the next data to be fetched from DRAM on that channel. Each channel uses START, INIT and END registers to define the size and location of the buffer in memory from which the DMA will take place. However, all three channels have slightly different methods of using these registers. Exact details of the contents of all these registers can be found in the programmer's model section of the datasheet.

14.3.1 Video DMA

The video DMA channel can be used in two modes. Duplex mode is used for fetching DMA data for use with a dual panel LCD display, and involves fetching a quad word of data for the top half of the display, followed by a quad word of data for the bottom half of the display, then the next quad word for the top half and so on. This is implemented using two parallel sets of registers which must be programmed accordingly. A description of how to use the ARM7500 with a dual panel LCD display can be found in *Appendix B: Dual Panel Liquid Crystal Displays*.

Normal mode is used for standard CRT and LCD displays and data is fetched sequentially from the frame buffer. Selection between normal and duplex mode of operation is achieved via bit 7 of the VIDCR register at location 0x032001E0. Bit 5 of the same register enables the video DMA channel. It should not be enabled until the other address registers have been programmed to sensible values.

The registers associated with video DMA should only be programmed during the FLYBACK period, to avoid corrupting data while DMA is in progress or while the display is half way through a raster. The state of the internal FLYBACK signal is available for polling in the IOCR register, and can create an interrupt by programming the IRQA mask register appropriately.

There is a single VIDSTART register, which should be programmed with the location in memory of the first quad word of video data at the start of the frame buffer. The VIDEND register is programmed with the location in memory of the start of the last quad word in the frame buffer image.

For normal mode operation, the VIDINITA register should be programmed with the address in memory of the data which will be used to create the pixels at the top left corner of the display. This need not necessarily be at the same address as that programmed into the VIDSTART register, thus allowing hardware scrolling by moving the address in the VIDINITA register through the frame buffer. The value in the VIDINITA register is automatically transferred into the VIDCURA register during the FLYBACK period, so there is no need to program the current register separately. For normal operation, the VIDINITB register should be programmed to 0x00000000, so that the value in the VIDCURB register is defined. All video channel registers should be programmed with addresses which are quad word aligned (ie bits 0 to 3 are zero).

There is an extra bit (30) in the VIDINITA register, which must be programmed HIGH if the address in the VIDINITA register is the same as the address in the VIDEND register. At all other times it should be programmed LOW.

Once all bits have been programmed, the enable bit in the VIDCR register can be written to, and the video DMA channel will become operational. The channel is then controlled by a video request signal from the video controller part of ARM7500. When a request for more video data arrives and the current bus cycle finishes, the bus controller will arbitrate in favour of the DMA (which has the highest priority on the bus) to fetch a quad word of data for the video sub system. Immediately after each DMA access, the address in the current register is incremented by 16 (one quad word) and the address is compared with the address in the VIDEND register. If they are the same, the DMA controller knows that the next DMA will be the last one in the buffer, and after the next DMA, the current register will be reloaded from the VIDSTART register. During the FLYBACK period, the current register will be automatically reloaded with the value in the VIDINITA register.

Programming of the DMA and video subsystem for use with dual panel LCDs is described in full in *Appendix B: Dual Panel Liquid Crystal Displays*, and uses identical principles, except there are two current registers and two init registers, one for each panel. On each successive DMA access, the ARM7500 will toggle between the two sets of registers providing data first for the upper panel and then from the lower panel. This means that the two init registers should always be programmed with addresses with are equidistantly spaced through the wrapped-around frame buffer.

14.3.2 Cursor DMA

There are only two registers associated with the cursor channel, the CURSCUR current register and the CURSINIT register. The channel is enabled under the control of the video enable bit in the VIDCR video DMA control register. The operation of the channel is the same for normal or duplex modes, but it is necessary to program the cursor differently depending on which mode is being used. Details of the programming required can be found in *Appendix B: Dual Panel Liquid Crystal Displays*.

The CURSINIT register should be programmed with the address of the first word of cursor data in memory. There is no END register as the width of the cursor is predetermined (32 pixels) and the height of the cursor is defined by programming the VCSR and VCER registers in the video sub system. Each quad word fetch will result in two rasters' worth of cursor data being transferred, except in Hi-Res Mode (see *11.4 Hi-Res support* on page 11-6). At the end of each fetch, the value in the CURSCUR register is increased by 16, to address the start of the next quad word. The value programmed into the CURSINIT register must be quad word aligned.

14.3.3 Sound DMA

The Sound DMA channel provides data for the ARM7500 sound interface. There are two sets of pointer registers so that data transfers can be double buffered to ensure that DMA data is always available even when the data in one buffer is exhausted. One set of registers can be reprogrammed while the others are being used.

Sound DMA transfers are constrained to a single 4KByte page, as only the lowest 12 bits of the DMA address are incremented and compared to check for the end of the buffer. All sound DMA is quad word and must be from quad word aligned addresses, so the lowest four bits of the registers are not used and should be programmed to zero. Bit 30 of each of the END registers is the "last" bit, which must be programmed HIGH if the initial value in the current register is the same as the end register for that buffer, ie for a single transfer.

There is also an interrupt mask and status bit for the sound channel which allows the status of the sound DMA state machine to be monitored. The state machine will generate an interrupt when the end of the current buffer is reached, and it is up to the system software to take appropriate action to reprogram that channel as required while DMA continues from the location pointed to by the other set of buffers.

Sound data is requested by the ARM7500 sound subsystem which asserts a request signal, and the bus controller will arbitrate in favour of the sound DMA when the current bus cycle has completed as long as there is not an outstanding video or cursor DMA request.

14.3.4 The Sound DMA state machine

The sound DMA channel is controlled by a simple state machine. The state machine remains in an idle state when the enable bit in the sound DMA control register has not been set. The state bits of the state machine are directly mapped to the Sound DMA status register, where they are named Overrun, Int and A/B. On reset, the state machine is set to state 110, such that the Overrun and Int bits are set. The Overrun bit indicates when a channel has stopped because it has finished a transfer and the other pointer pair has not been programmed. The Int bit indicates when the channel is requesting an interrupt. The A/B bit indicates which pair of current/end pointers is in use.

The state machine diagram in the figure below shows how the state machine transfers between buffers A and B to allow DMA to continue uninterrupted when both sets of DMA address registers have been programmed. The transitions between states occur either when the ARM processor programs an pointer register pair, or when a buffer is completed. To ensure correct operation, the current pointer must be programmed before the end pointer as it is the action of programming the end pointer which causes the state transition. The “stop” bit in the end register is used to terminate a sequence of DMA, by forcing the state machine back into one of the idle states at the end of the last buffer.

During operation of the state machine, when the end of one buffer is reached, an interrupt will be generated which can be used to signal to the ARM processor that it is time to reprogram that pair of pointers. If one buffer’s address pointers have not been reprogrammed before the other buffer is exhausted, then both the Int and Overrun bits will be set, and DMA cannot continue until the pointers are reprogrammed.

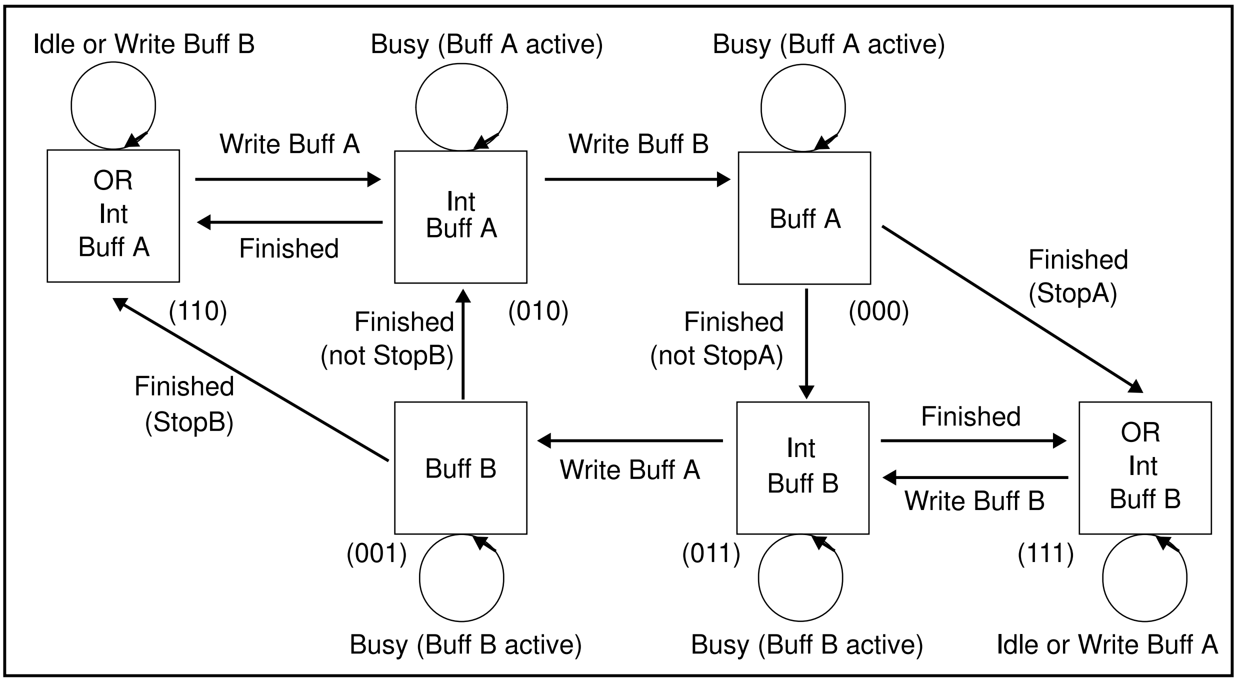


Figure 14-9: Hardware DMA state machine diagram

15 I/O Subsystems

This chapter describes the ARM7500 I/O subsystems.

15.1 Introduction

ARM7500 has a 16-bit wide general I/O port, BD[15:0]. This allows slow I/O access to continue independently of DMA activity on the ARM7500 data bus. There are three types of I/O access supported over the I/O bus, which are 16MHz PC style I/O, 8MHz request/grant based I/O and simple 8MHz based fixed timing I/O. ARM7500 also has a separate 8-bit wide general purpose open drain I/O port, each bit of which can be configured as an interrupt source. There are four analog comparators, each with a 16 bit 2MHz timer which can be used as a four channel analog joystick interface. Two identical PS/2 serial mouse/keyboard ports are included. There are two general purpose 2MHz 16-bit counter timers, which can be programmed to produce interrupts at timed intervals. ARM7500 includes an interrupt handler, with enable and mask bits for each interrupt source, which can process potential interrupts from a number of internal and external sources.

The 16MHz PC style I/O provides all the signals required to interface with a standard PC Combo chip, enabling an industry standard part to be used to complete the I/O interfaces to devices such as a floppy disc.

The facility is available to expand the width of the I/O bus externally by adding latches and buffers to the upper 16 bits of the main external data bus and control signals for these devices are provided from ARM7500. Support is provided for Execute-in-place (XIP) from a 16-bit wide PCMCIA card attached to the I/O bus.

Because the I/O clocks can be completely asynchronous to the memory system clock (which is controlling the main bus arbitration state machine), there will be additional synchronisation penalties at the start and end of the I/O cycle. Clearly the exact additional delay will depend on the actual phase of the clocks at the point in question, and the timing diagrams do not attempt to show this in detail. However the worst case synchronisation delays are indicated.

15.2 I/O address space usage

The main I/O address space is defined as being from address 0x03000000 to 0x03FFFFFF:

I/O address	Contents
0x03000000	Module space - asserts **nMSCS**
0x03010000	16MHz I/O - asserts **nCCS** (Combo chip select)
0x03012000	16MHz I/O - asserts **nCDACK** (Combo DACK)
0x0302A000	16MHz I/O - asserts **nCDACK** and **TC** (Combo DACK and TC)
0x0302B000	16MHz I/O - asserts **nPCCS1**
0x0302B800	16MHz I/O - asserts **nPCCS2**

Table 15-1: I/O address space usage

I/O address	Contents
0x0302C000	Reserved
0x03030000	Module space - asserts **nMSCS**
0x03040000	Reserved
0x03200000	ARM7500 internal I/O and memory control registers
0x03210000	Simple I/O space - asserts **nSIOCS1/2**
0x03400000	ARM7500 internal video and sound control registers
0x03500000	Reserved

Table 15-1: I/O address space usage (Continued)

In addition, there is an extended I/O address space for 16MHz PC style I/O from address 0x08000000 up to 0x0FFFFFFF, divided into eight 16MB areas. The chip select generated throughout this area is **nEASCS**.

15.3 Additional I/O chip select decode logic

The **SETCS** input is used to select between additional decode logic for the some of the main chip select outputs.

The effect of this is that when **SETCS** is LOW:

nMSCS is asserted over the whole of Module space

nEASCS is asserted over the whole of Extended I/O address space

nSIOCS2 is asserted only in the following ranges of simple I/O space:

0x03240000 -> 0x0324FFFF
0x032C0000 -> 0x032CFFFF
0x03340000 -> 0x0334FFFF
0x033C0000 -> 0x033CFFFF

When **SETCS** is HIGH:

nMSCS is asserted only in the following ranges of Module I/O space:

0x03000000 -> 0x03003FFF
0x03030000 -> 0x03033FFF

nEASCS is asserted only in the following range of Extended I/O space:

0x08000000 -> 0x08FFFFFF

nSIOCS2 is asserted only in the following ranges of Simple I/O space:

0x03240000 -> 0x03243FFF
0x032C0000 -> 0x032C3FFF
0x03340000 -> 0x03343FFF
0x033C0000 -> 0x033C3FFF

15.4 Simple 8MHz I/O

The Simple I/O type of access is 16-bit only and has a selection of 4 different cycle speeds selectable by bits 20 and 19 of the address. This type of I/O will be selected for addresses in the range 0x3210000 to 0x32FFFFFF. When writing, the upper half-word of the ARM7500 data bus is written out on the I/O bus. When reading, the I/O bus data is read back onto the lower half-word of the ARM7500 data bus.

During these accesses, the signal **nSIOCS1** is always asserted with a read or write strobe as appropriate based on the CLK8 8MHz clock **nSIOCS2** is asserted according to the decoding in the section above. The read and write strobes are the **nIOR** and **nIOW** output pins respectively. The four timings of the Simple 8MHz I/O accesses are shown below:

Address [20:19]	Name	Minimum CLK8 cycles
0 0	slow	7
0 1	medium	6
1 0	fast	5
1 1	sync	5

Table 15-2: Timings of the Simple 8MHz I/O accesses

The “sync” timing is referenced to the 2MHz CLK2 output, and there will thus be an additional possible synchronisation penalty of up to 3 CLK8 cycles depending on the phase of CLK2 and CLK8 at the commencement of the I/O cycle. This is in addition to synchronisation between the I/O and memory subsystem signals.

The diagrams below show the timing of the four different types of simple I/O cycles. Note that all diagrams assume divide by 1 mode for both **MEMCLK** and **I_OCLK**.

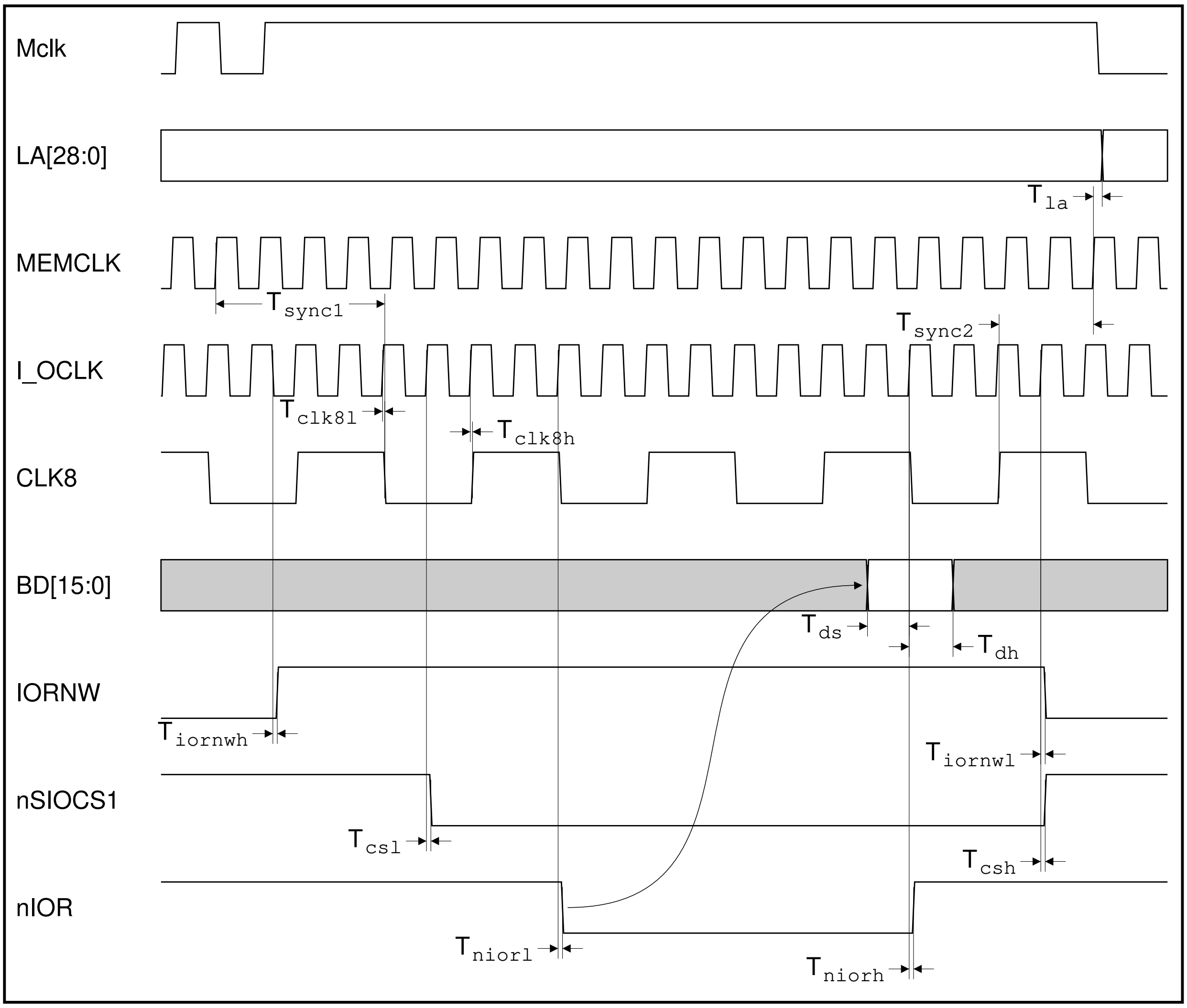


Figure 15-1: ‘Fast’ 8MHz Simple I/O read cycle timing

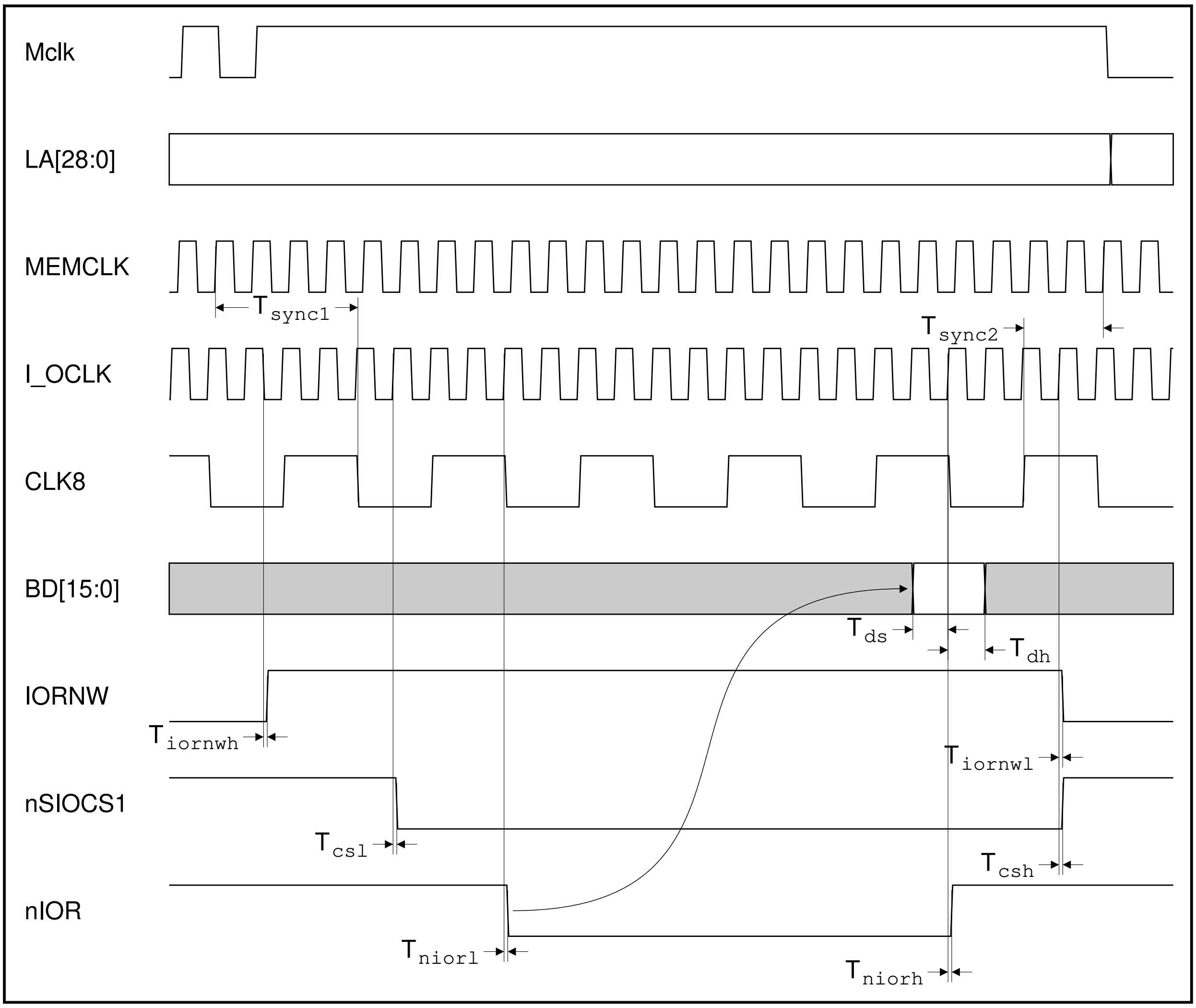


Figure 15-2: 'Medium' 8MHz Simple I/O read cycle timing

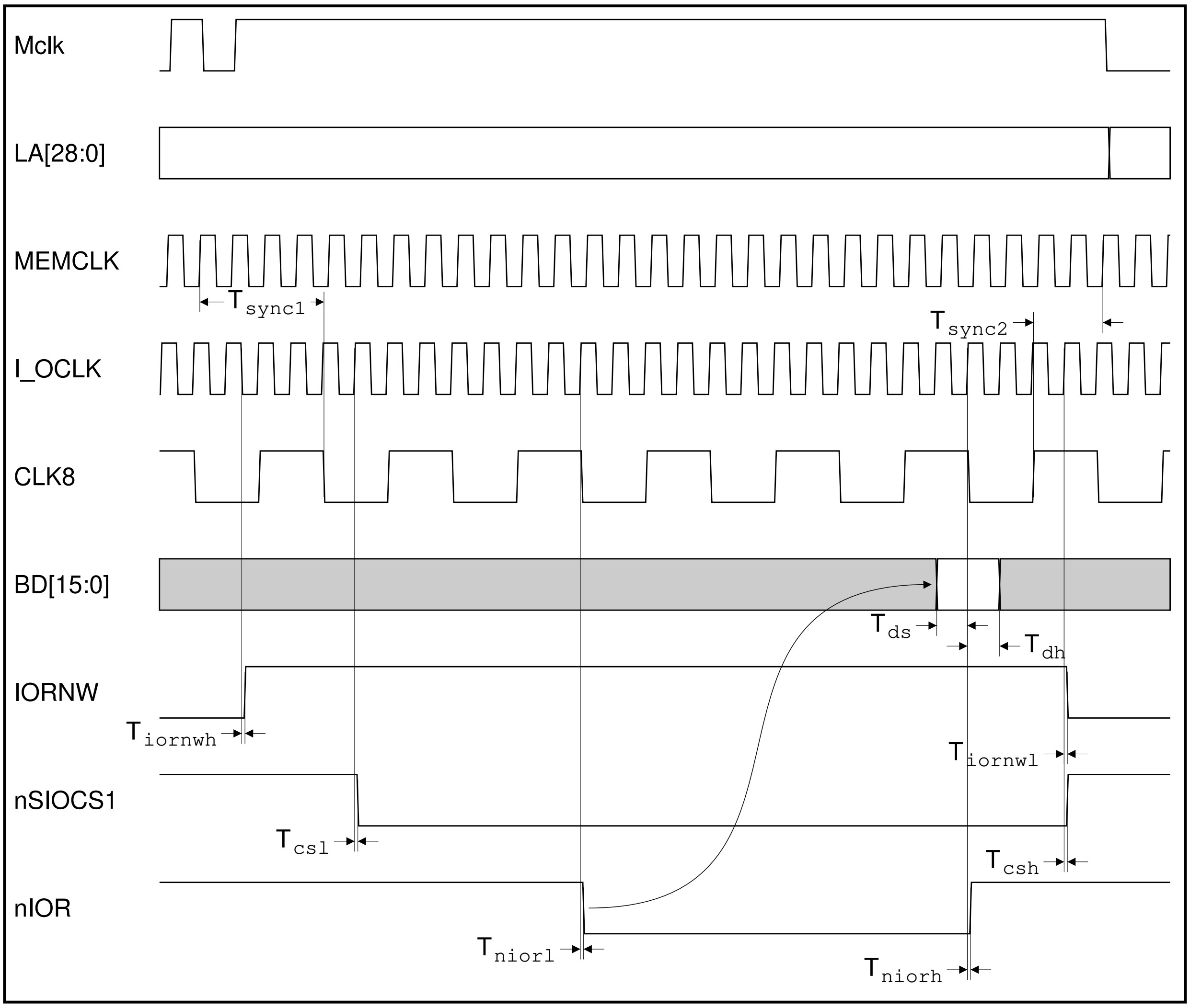


Figure 15-3: 'Slow' 8MHz Simple I/O read cycle timing

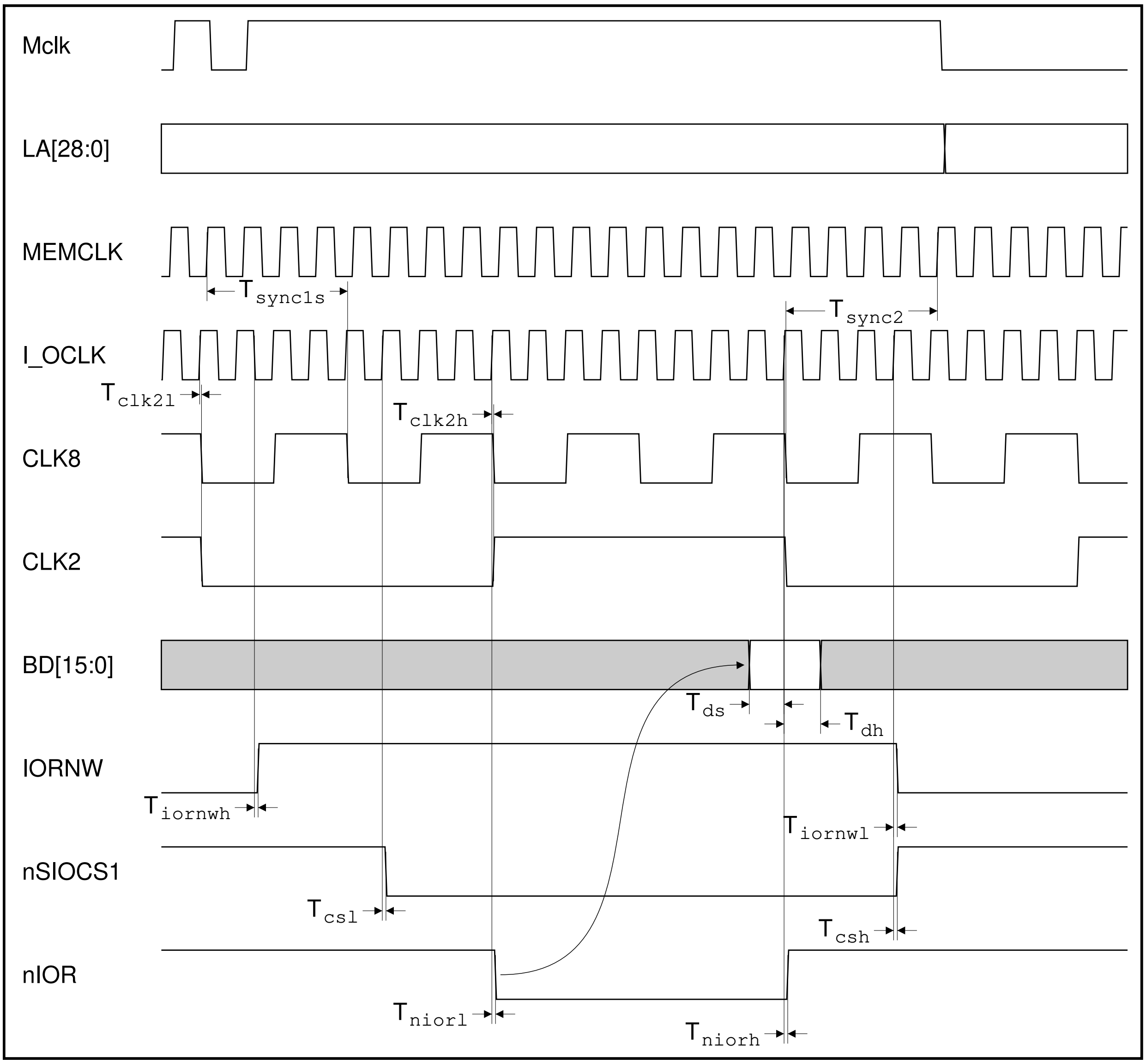


Figure 15-4: 'Sync' 8MHz I/O read cycle timing

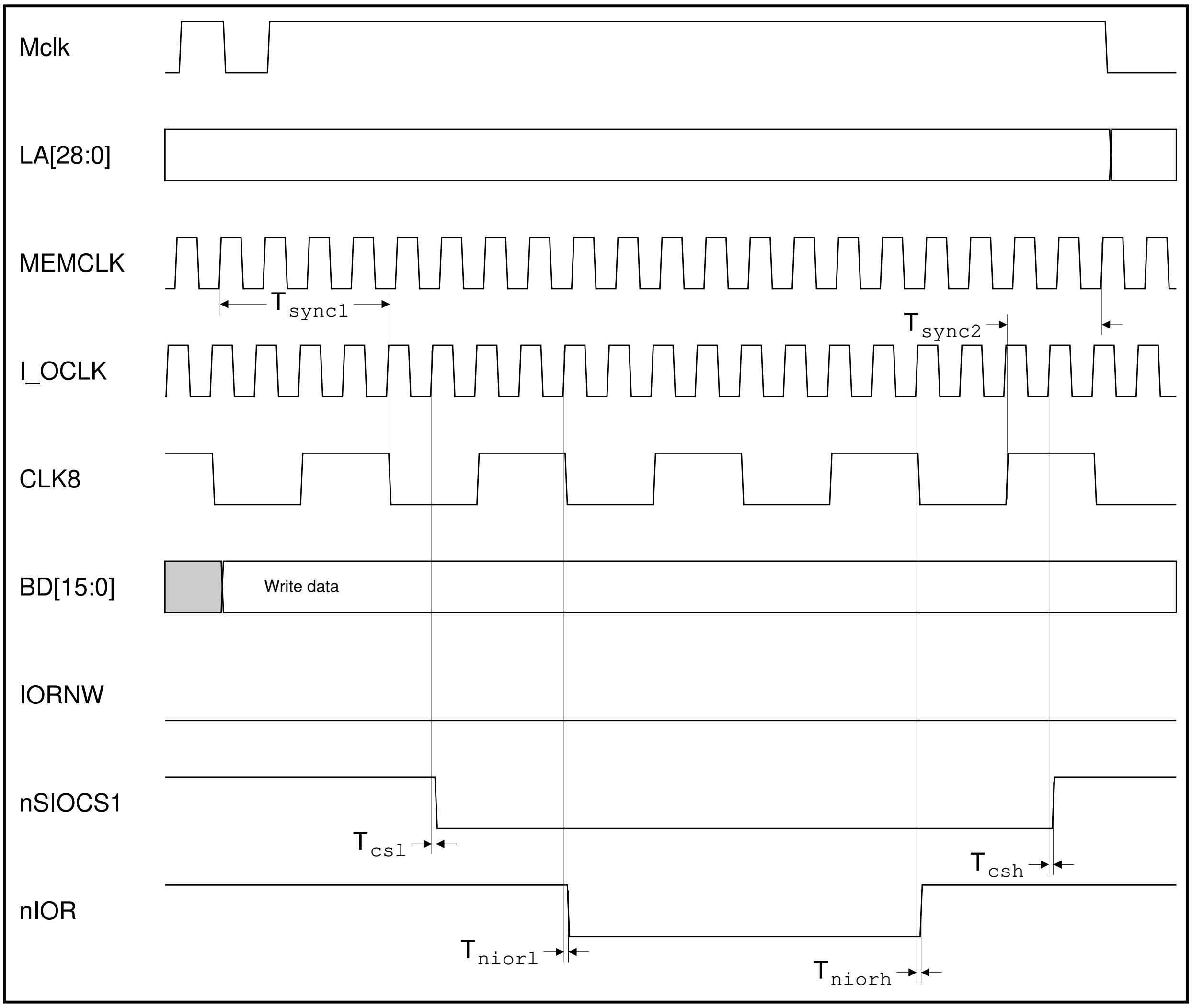


Figure 15-5: 'Fast' 8MHz Simple I/O write cycle timing

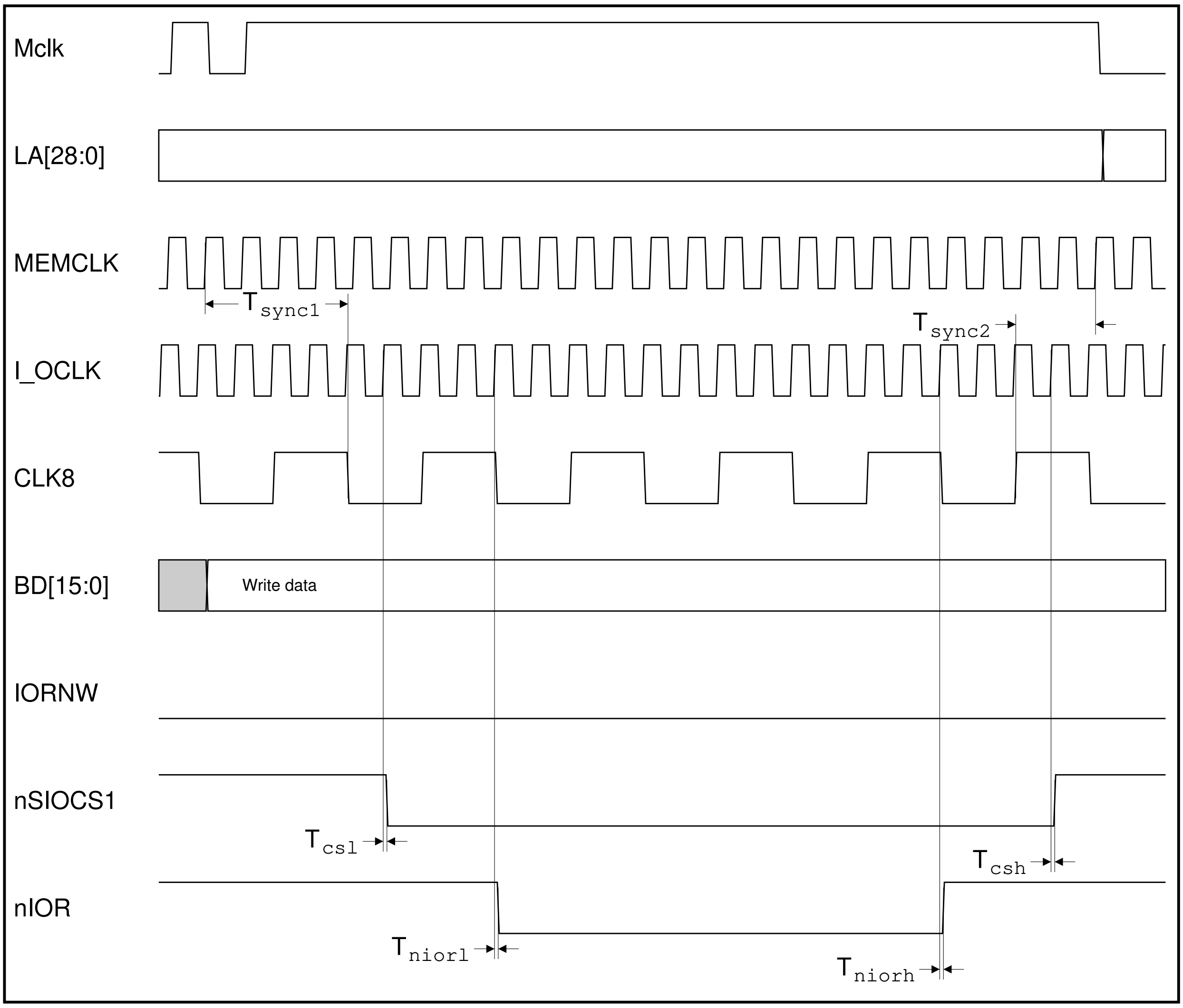


Figure 15-6: 'Medium' 8MHz Simple I/O write cycle timing

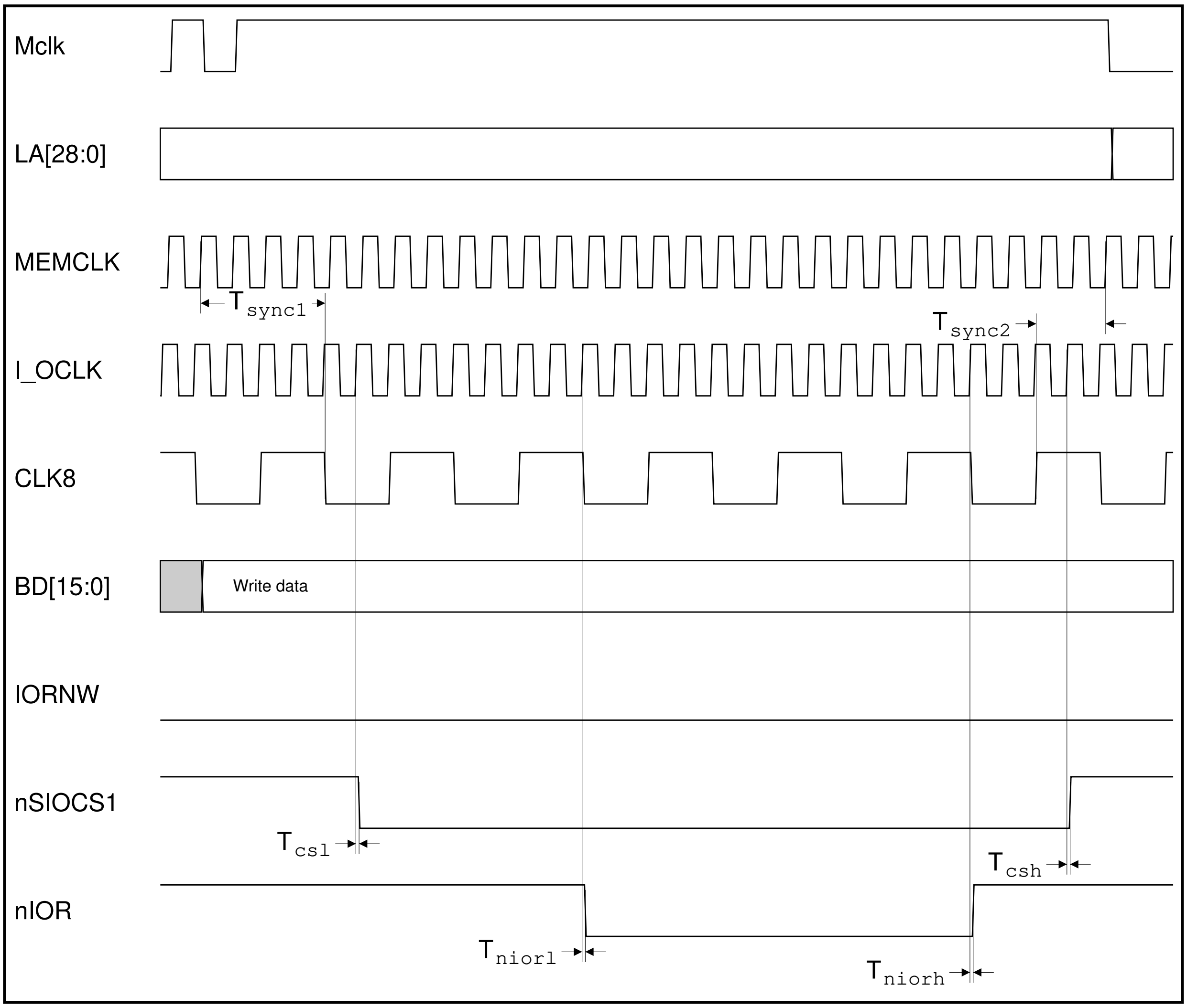


Figure 15-7: 'Slow' 8MHz Simple I/O write cycle timing

Pre-release Draft - Unrestricted

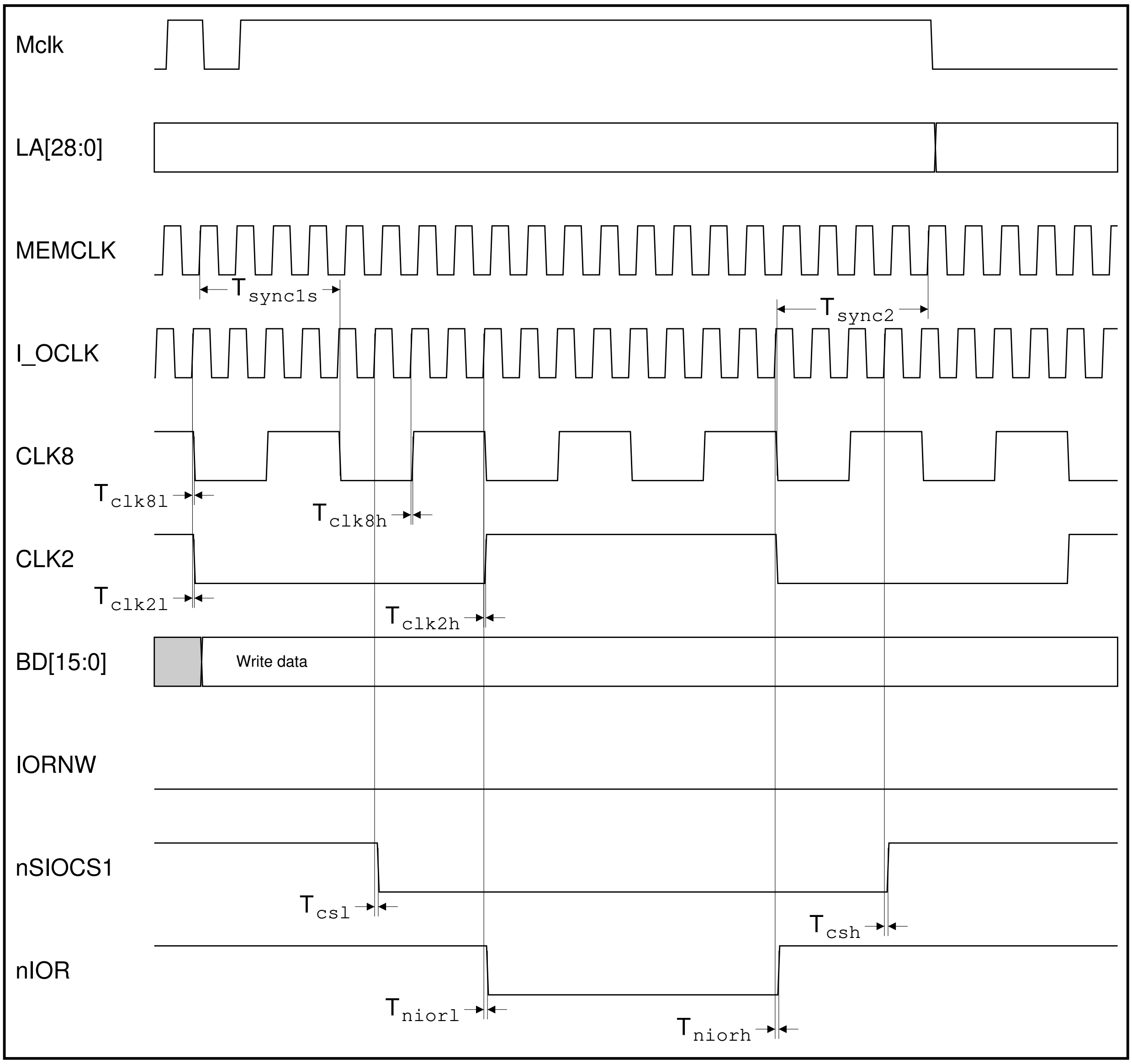


Figure 15-8: 'Sync' 8MHz Simple I/O write cycle timing

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tclk8l	**I_OCLK** rising to **CLK8** falling					
Tclk8h	**I_OCLK** rising to **CLK8** rising					
Tclk2l	**I_OCLK** rising to **CLK2** falling					
Tclk2h	**I_OCLK** rising to **CLK2** rising					
Tcsl	**I_OCLK** rising to **nSIOCS1** falling					
Tcsh	**I_OCLK** rising to **nSIOCS1** rising					
Tdh	DATA hold from **nIOR** rising					
Tds	DATA setup to **nIOR** rising					
Tiornwh	**I_OCLK** falling to **IORNW** rising					
Tiornwl	**I_OCLK** rising to **IORNW** falling					
Tniorl	**I_OCLK** rising to **nIOR** falling					
Tniorh	**I_OCLK** rising to **nIOR** rising					
Tniowl	**I_OCLK** rising to **nIOR** falling					
Tniowh	**I_OCLK** rising to **nIOR** rising					
Tsync1	synchronisation time from **MEMCLK** rising which creates MCLK falling to **I_OCLK** rising which creates next **CLK8** falling					
Tsync1s	synchronisation time from **MEMCLK** rising which creates MCLK falling to **I_OCLK** rising which creates next **CLK8** falling while **CLK2** is low					
Tsync2	synchronisation time from **I_OCLK** which creates **CLK8** rising to **I_OCLK** rising at end of cycle to **MEMCLK** rising which creates MCLK falling					
Tla	**LA[ ]** changing after **MEMCLK** rising which creates MCLK falling					

Table 15-3: Timing diagrams

15.5 Module I/O

The Module I/O type of access is 16-bit only and its speed is controlled by a handshake mechanism with the external hardware. The signals **nIORQ** (output) and **nIOGT** (input) are used for this handshaking. When writing, the upper half-word of the ARM7500 data bus is written out on the I/O bus. When reading, the I/O bus data is read back onto the lower half-word of the ARM7500 data bus. The module type of I/O will be initiated for addresses in the ranges 0x03000000 to 0x0300FFFF and 0x03030000 to 0x0303FFFF.

During these accesses, the signal **nMSCS** is asserted but read and write strobes are not used, although the **IORNW** signal is active.

The **nBLI** is driven by the external hardware to indicate when the read or write data should be latched from the BD I/O bus.

The I/O cycle will terminate when both **nIORQ** and **nIOGT** are LOW at the rising edge of REF8M. The timing diagrams below show the signal relationship for the **nIORQ**/**nIOGT** module I/O type of access.

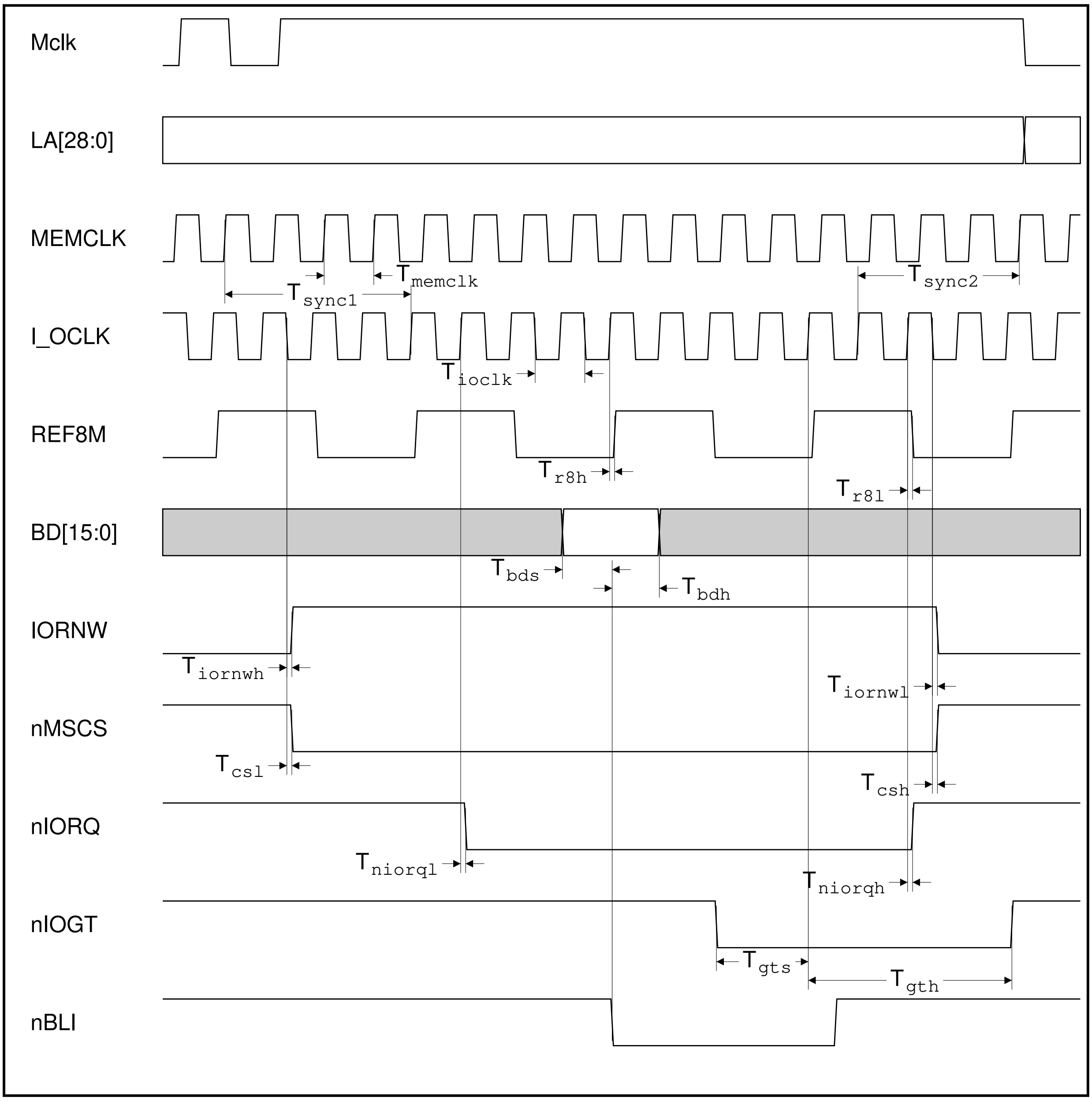


Figure 15-9: 8 MHz Module read I/O cycle

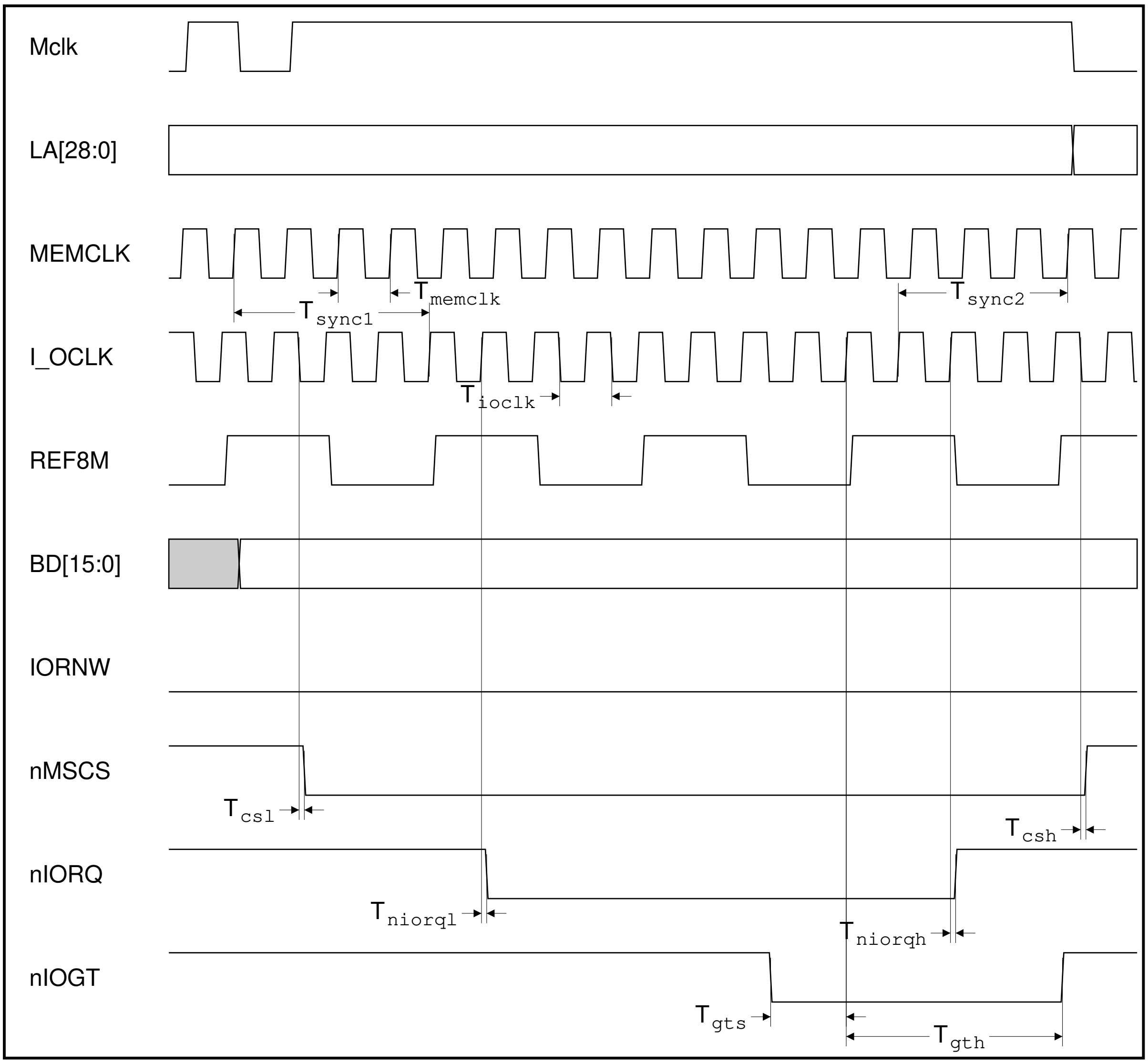


Figure 15-10: 8 MHz Module write I/O cycle

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tbds	Data setup up to **nBLI** falling					
Tbdh	Data hold from **nBLI** falling					
Tcsl	**I_OCLK** falling to **nMSCS** falling					
Tcsh	**I_OCLK** rising to **nMSCS** rising					

Table 15-4: 8 MHz Module read and write I/O cycles

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tiornwh	**I_OCLK** falling to **IORNW** rising					
Tiornwl	**I_OCLK** rising to **IORNW** falling					
Tniorql	**I_OCLK** rising to **nIORQ** falling					
Tniorqh	**I_OCLK** rising to **nIORQ** rising					
Tr8ml	**I_OCLK** rising to **REF8M** falling					
Tr8mh	**I_OCLK** rising to **REF8M** rising					
Tsync1	synchronisation time from **MEMCLK** rising which creates MCLK falling to **I_OCLK** rising which creates next **REF8M** rising					
Tsync2	synchronisation time from **I_OCLK** rising after **REF8M** rising at end of cycle to **MEMCLK** rising which creates MCLK falling					
Tgts	setup of **nIOGT** to **I_OCLK** rising at **REF8M** rising					
Tgth	hold of **nIOGT** from **I_OCLK** rising at **REF8M** rising					

Table 15-4: 8 MHz Module read and write I/O cycles

15.6 PC bus style I/O

This type of I/O is designed to function in conjunction with a standard PC Combo chip, and cycles are generated from a 16MHz clock.

The PC bus style I/O type of access routes the lower halfword of the ARM7500 bus through the device providing a direct 16 bit interface. Additionally, signals are generated to support the addition of external latches/drivers to extend the I/O data by 16 bits. The upper half-word of the ARM7500 data bus is routed through these external devices if present. This type of I/O access is used for the address space from 03010000 to 0302CFFF (five sections), and in the larger extended address space from 0x08000000 to 0x0FFFFFFF (eight sections). There are 4 fixed cycle types based on the 16MHz clock, although the larger extended address area only supports two of these cycle types. Any access may be held up by external circuitry removing the **READY** signal before the end of the cycle.

The signals used to control the external buffers and latches required to implement 32-bit wide I/O are **nWBE**, **nRBE**, and **nBLO**. The timing diagrams in this section (*Figure 15-11: 16 MHz Type D read I/O cycle* and *Figure 15-12: 16 MHz Type D write I/O cycle*) show the timing of these signals relative to the external data bus. For full details of the external circuitry and connections required to implement a 32-bit wide I/O system using the ARM7500, refer to *Appendix D: Expanding PC-style I/O to 32 bit* .

Two additional inputs, **nXIPLATCH** and **nXIPMUX16**, are provided to allow external circuitry to route a full 32 bit data word through the 16 bit I/O bus using multiplexing. This would allow, for example, the execution of ARM code from a 16 bit wide PCMCIA card with a suitable external controller. The **nXIPMUX16** signal directly controls an internal multiplexer which maps either the upper or lower 16 bits of the internal data bus through to the 16 bit wide I/O bus, for writes to an I/O peripheral. When **nXIPMUX16** is LOW, the upper 16 bits of the data bus are passed to **BD[15:0]**, and when **nXIPMUX16** is HIGH, the lower 16 bits of the data bus are passed to **BD[15:0]**. For reads from an I/O peripheral, the falling edge of the **nXIPLATCH** signal causes the first 16 bits provided on the **BD[15:0]** bus to be latched as the upper half word for the main internal data bus, after which the lower 16 bits can be output from the peripheral and the I/O cycle can be allowed to complete normally. If **nXIPLATCH** has been driven low, the upper half word of data is driven to the ARM processor internally and not from the external transceivers if present. *Figure 15-19: 16 MHz Type B read I/O cycle with PCMCIA* and *Figure 15-20: 16 MHz Type B write I/O cycle with PCMCIA* show the relevant timing details. Depending on the cycle timing, it will usually be necessary for the external controller to use the **READY** signal to stretch the I/O access to give sufficient time for both half words to be read or written as appropriate.

If an I/O access is to be stretched, the **READY** signal should be set LOW at least one 16MHz clock cycle before the rising edge of **nIOR** or **nIOW**. This will cause the **nIOR** or **nIOW** strobe and the chip select to be held LOW until **READY** is set back to HIGH again, when the I/O cycle will complete as normal. For details of READY timing see *Figure 15-17: 16 MHz Type A read I/O cycle*.

The four address areas for 16MHz I/O within the main I/O address space can support any of the four available cycle types A to D. The IOTCR register can be programmed (at address 0x032000C4) to determine which type of cycle will be used for each group of addresses. The addresses are grouped such that the **nCCS** and pseudo DMA address spaces form one group, and the **nPCCS1** and **nPCCS2** address area forms another group.

7	6	5	4	3	2	1	0
X	X	X	X	C	C	N	N

C = **nCCS** + pseudo DMA access speed

N = **nPCCS1** and **nPCCS2** area access speed

Write	bits[7:6] unused
	bits[5:4] unused
	bits[3:2] 00: Type A (slowest); 01: Type B; 10: Type C; 11: Type D (fastest).
	bits[1:0] 00: Type A (slowest); 01: Type B; 10: Type C; 11: Type D (fastest).
Read	read back above values
Reset	set to zero (slowest)

The extended address space from address 0x08000000 onwards for 16MHz I/O accesses supports only cycle types A and C, and the ECTCR register should be programmed to specify which cycle type is required for each of the eight 16MB areas within the extended address space. The details of this register, at address 0x032000C8, are shown below:

7	6	5	4	3	2	1	0
E	E	E	E	E	E	E	E

E = expansion card area access speed

Write	bit[7] (0F00 0000 -> 0FFF FFFF), 0: Type A, 1: Type C bit[0] (0800 0000 -> 08FF FFFF), 0: Type A, 1: Type C
Read	read back above values
Reset	set to zero (slowest)

This type of I/O asserts a single chip select according to the area, except in Combo DACK + TC space, where both the **nCDACK** and **TC** outputs are asserted to signal to the PC Combo chip that the end of a pseudo DMA sequence has been reached. In the extended address space the **nEASCS** chip select is asserted.

The timing diagrams in the figures below show the four types of 16 MHz I/O cycle. Note that all diagrams assume divide by 1 mode for both **MEMCLK** and **I_OCLK**.

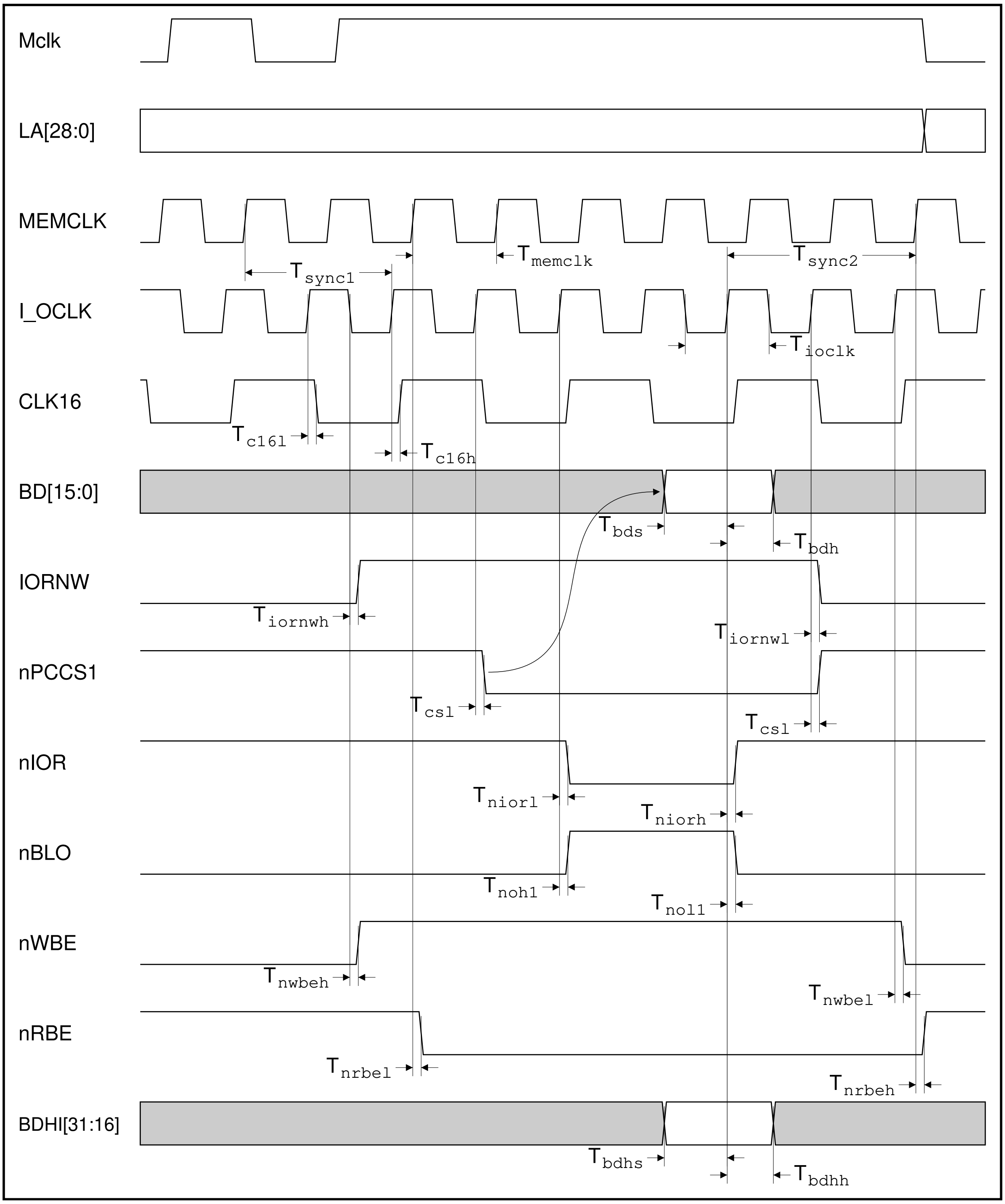


Figure 15-11: 16 MHz Type D read I/O cycle

Pre-release Draft - Unrestricted

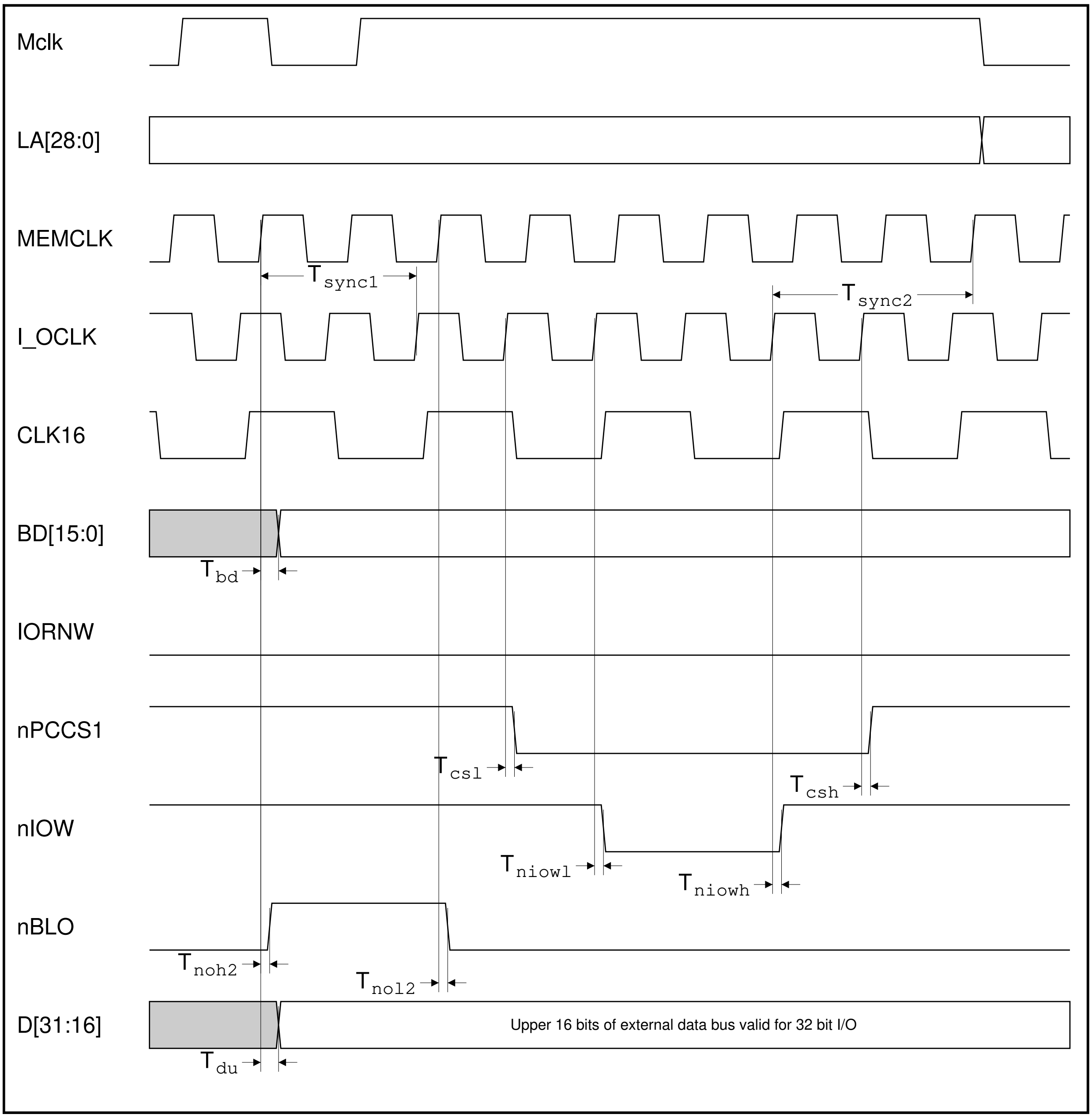


Figure 15-12: 16 MHz Type D write I/O cycle

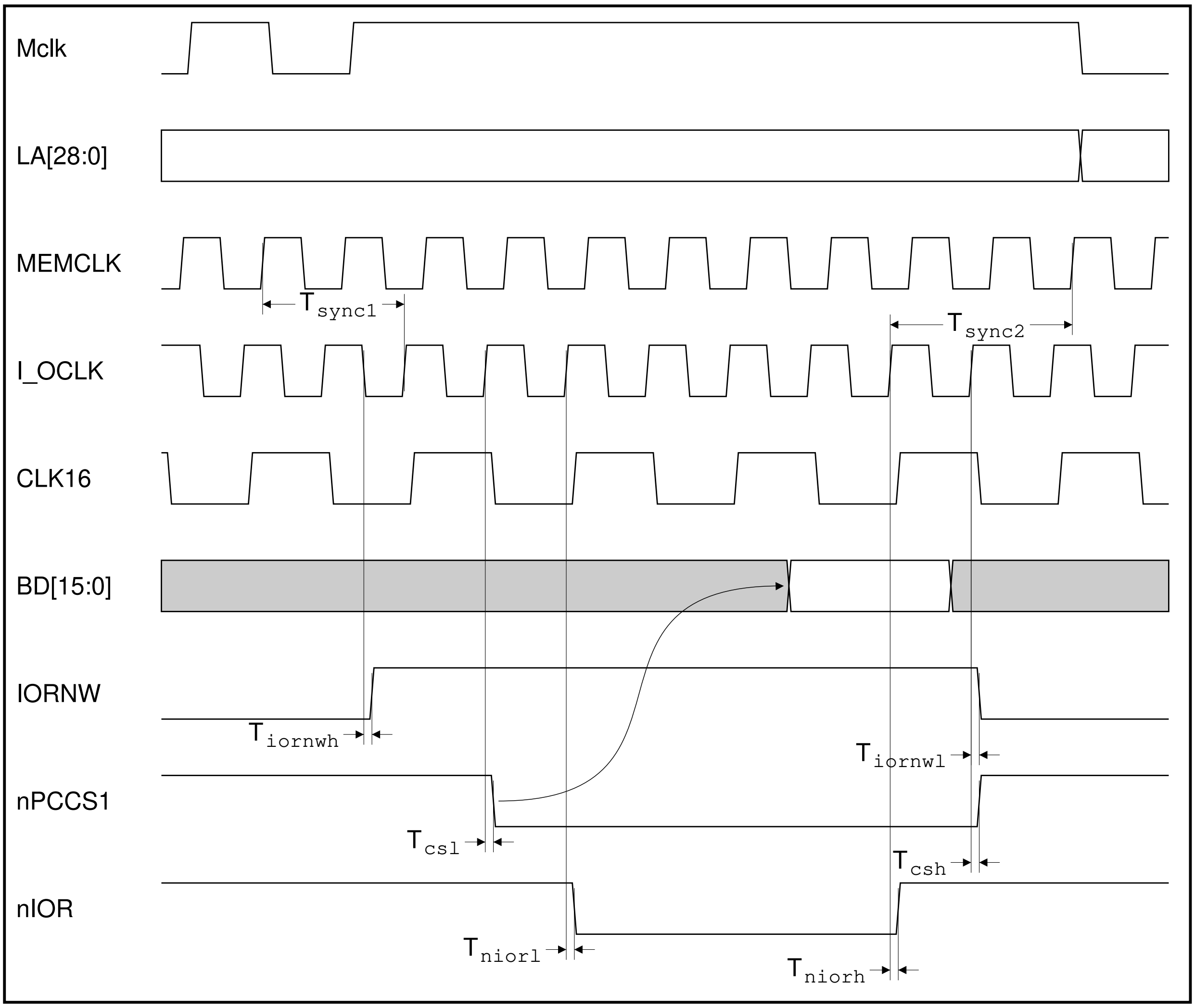


Figure 15-13: 16 MHz Type C read I/O cycle

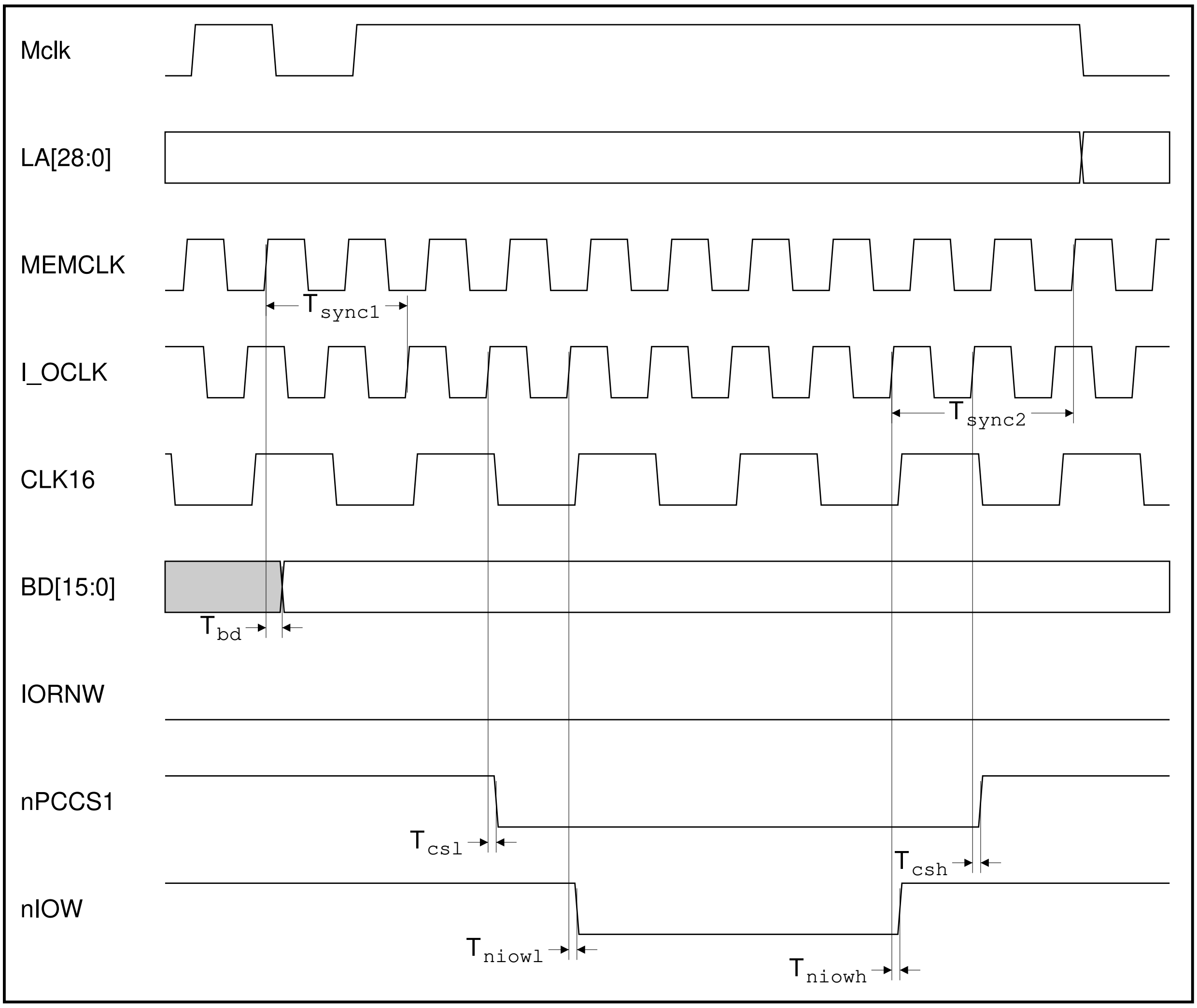


Figure 15-14: 16 MHz Type C write I/O cycle

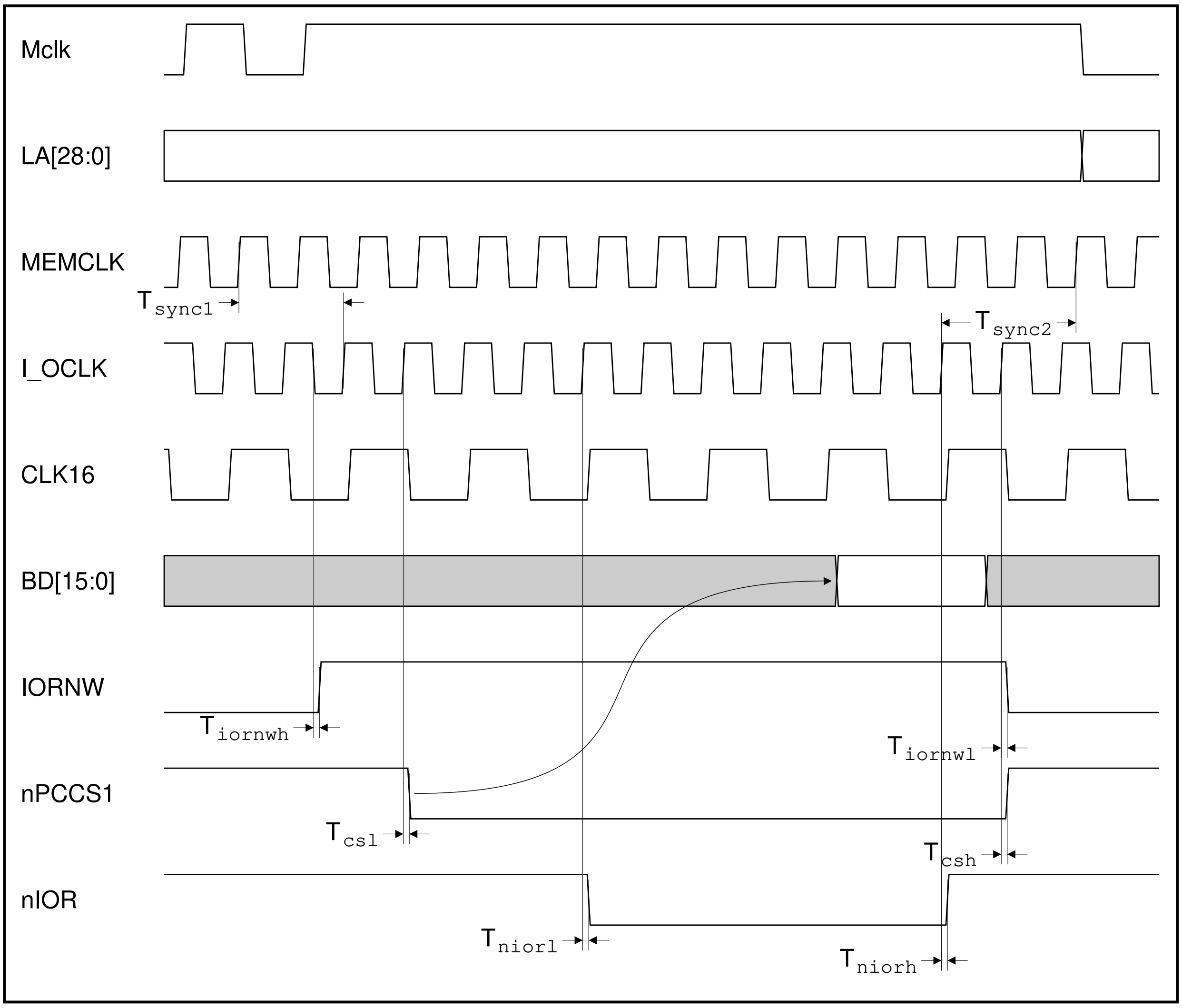


Figure 15-15: 16 MHz Type B read I/O cycle

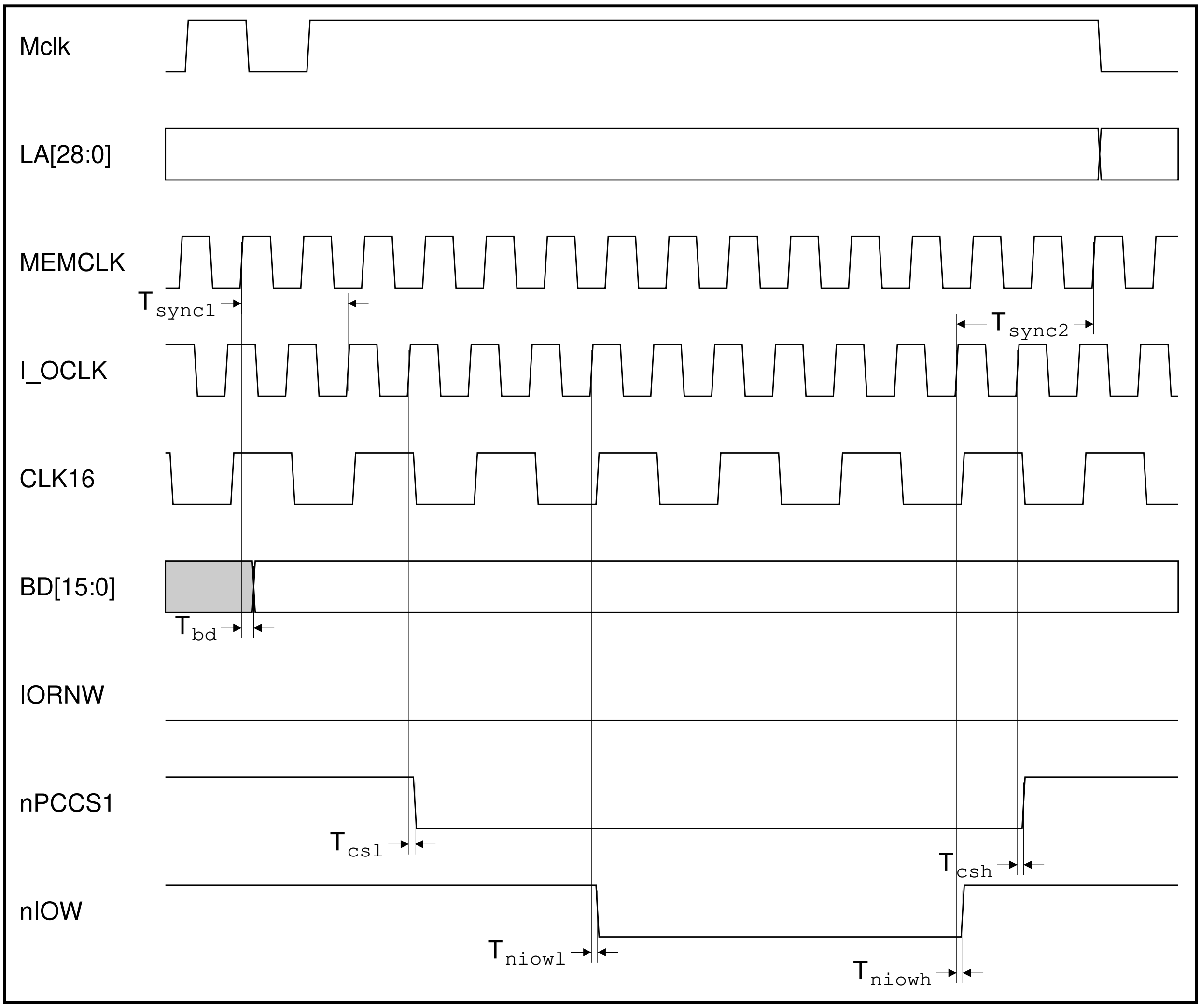


Figure 15-16: 16 MHz Type B write I/O cycle

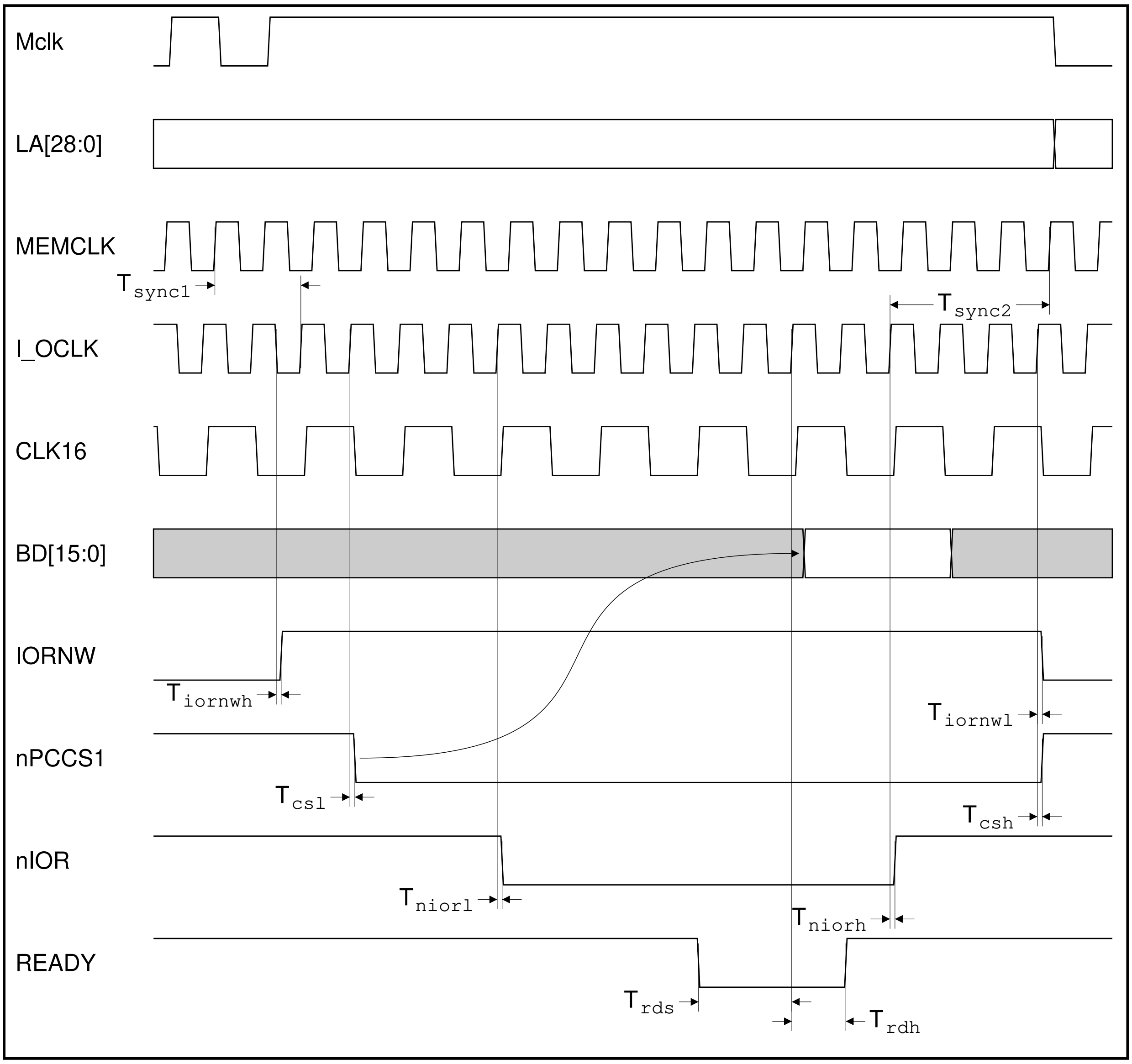


Figure 15-17: 16 MHz Type A read I/O cycle

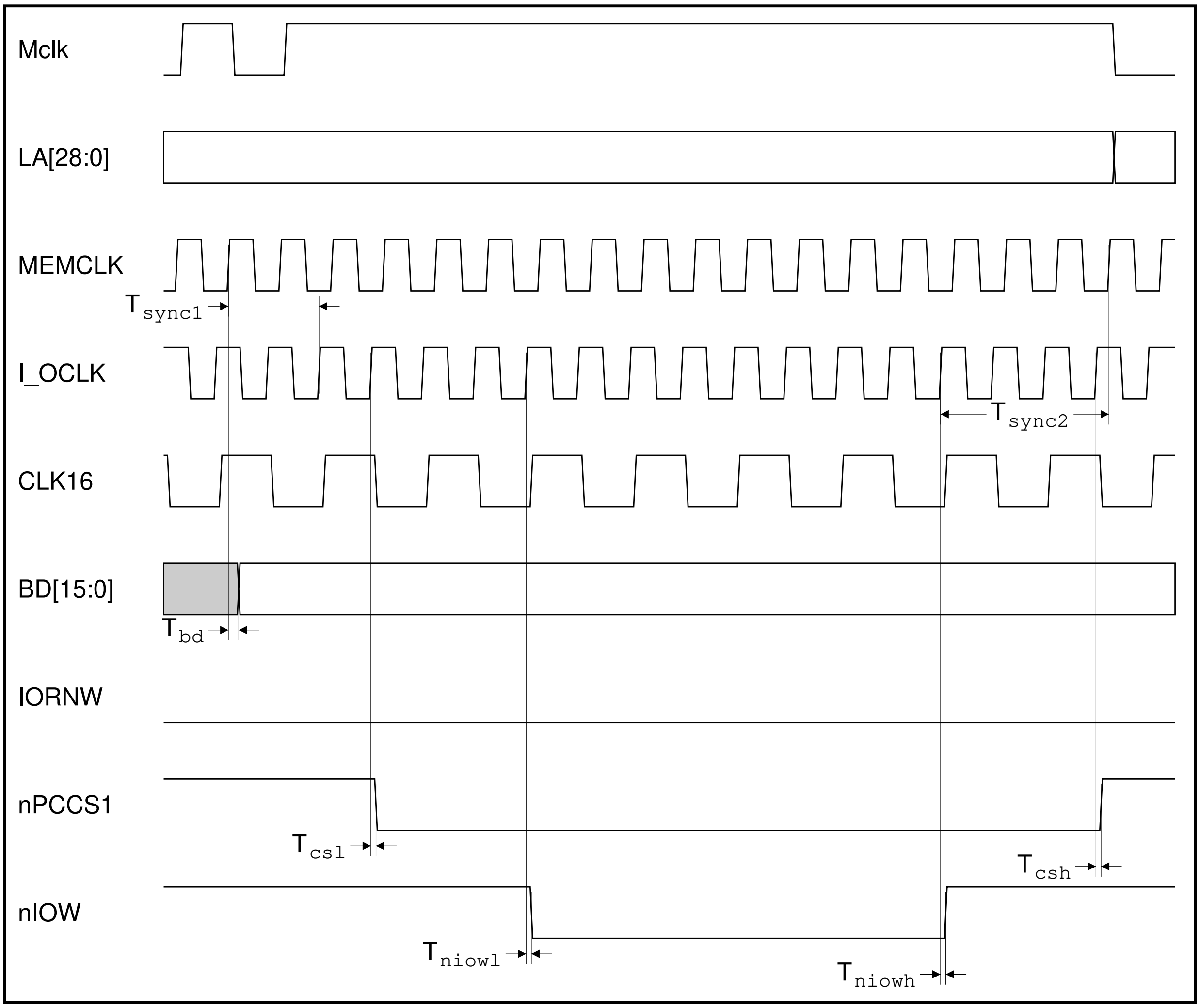


Figure 15-18: 16 MHz Type A write I/O cycle

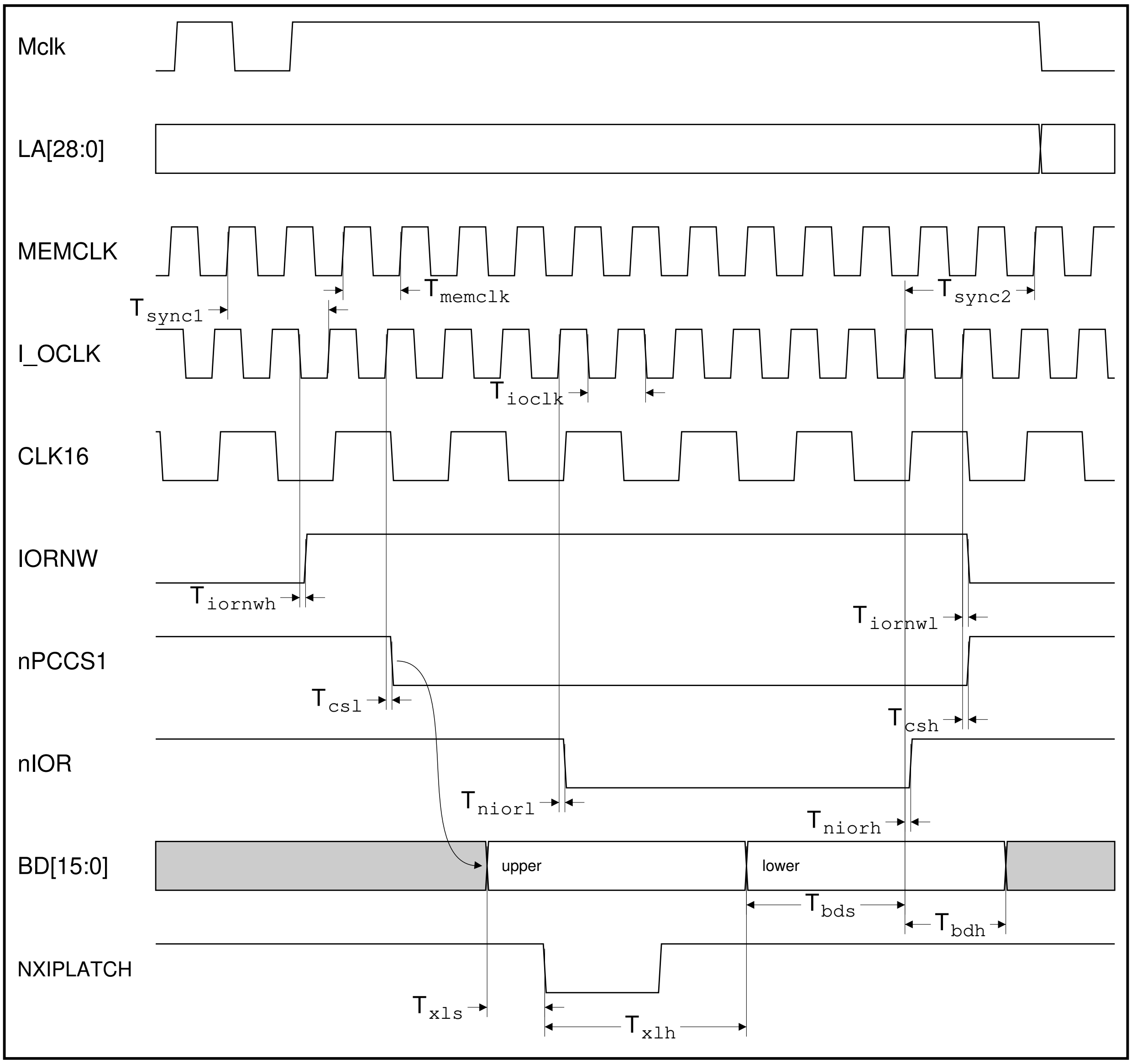


Figure 15-19: 16 MHz Type B read I/O cycle with PCMCIA

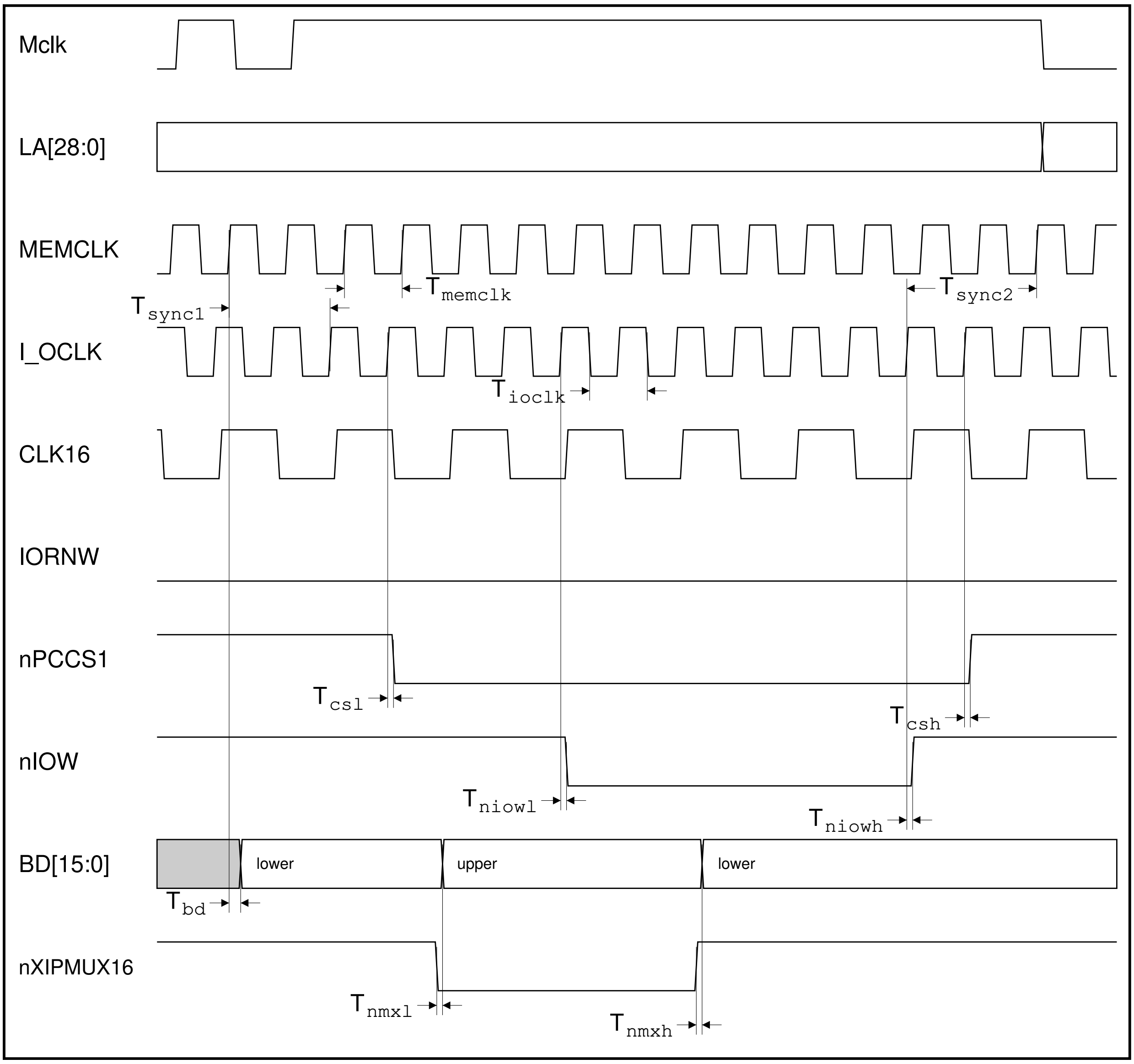


Figure 15-20: 16 MHz Type B write I/O cycle with PCMCIA

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tnmxl	**nXIPMUX16** falling to upper data output on **BD[15:0]**					
Tnmxh	**nXIPMUX16** falling to lower data output on **BD[15:0]**					

Table 15-5: 16 MHz I/O cycles

Symbol	Parameters	Min	Typ	Max	Units	Notes
Txls	DATA setup to **nXIPLATCH** falling					
Txlh	DATA hold from **nXIPLATCH** falling					
Tbdhs	High data hold time from **nIOR** rising					
Tbdhh	High data setup to **nIOR** falling					
Tc16l	**I_OCLK** rising to **CLK16** falling					
Tc16h	**I_OCLK** rising to **CLK16** rising					
Tbdh	Data hold from **nIOR** rising					
Tbds	Data setup to **nIOR** rising					
Tiornwh	**I_OCLK** falling to **IONRW** rising					
Tiornwl	**I_OCLK** rising to **IONRW** falling					
Tcsl	**I_OCLK** rising to **nPCCSI** falling					
Tcsh	**I_OCLK** rising to **nPCCSI** rising					
Tniorl	**I_OCLK** rising to **nIOR** falling					
Tiorh	**I_OCLK** rising to **nIOR** rising					
Tnoh1	**I_OCLK** rising to **nBLO** rising, read					
Tnol1	**I_OCLK** rising to **nBLO** falling, read					
Tnoh2	**MEMCLK** rising to **nBLO** rising, write					
Tnol2	**MEMCLK** rising to **nBLO** falling, write					
Tnwbeh	**I_OCLK** falling to **nWBE** rising					
Tnwbel	**I_OCLK** rising to **nWBE** falling					
Trbel	**MEMCLK** rising to **nRBE** falling					
Trbeh	**MEMCLK** rising to **nRBE** rising					
Tniowl	**I_OCLK** rising to **nIOW** falling					
Tniowh	**I_OCLK** rising to **nIOW** rising					
Tsync1	synchronisation time from **MEMCLK** rising that creates MCLK falling to **I_OCLK** rising which creates next **CLK16** rising					
Tsync2	synchronisation time from **I_OCLK** rising which creates **CLK16** rising at end of cycle to **MEMCLK** rising which creates MCLK falling					

Table 15-5: 16 MHz I/O cycles

15.7 DMA during I/O cycles

DMA to the Video and Sound Macrocell can continue during I/O cycles. Write data from the ARM Processor is latched early, so that the data bus can be used freely for DMA data. Thus only the start of an I/O cycle needs to be added to any DMA latency calculations.

15.8 Clock synchronisation conditions

In a system which uses a **MEMCLK** frequency greater than **I_OCLK**, it may be necessary to insert an extra I/O clock cycle to allow sufficient address hold time before the chip select is taken away. The problem arises because the chip select is generated from the fixed frequency I/O world clock, whereas the address changes according to the memory system clock. When a faster **MEMCLK** is used, it is possible for the synchronisation to the memory clock to occur rapidly at the end of the cycle, and thus for the I/O address to change before the chip select has been removed. This may be a problem for some peripherals. To avoid this, there is a register bit in the ASTCR register, at address 0x032000CC, which is normally set to zero, but can be programmed to one to add an extra I/O clock period to ensure that the address will not change before the chip select has been deasserted.

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	A

A = asynchronous timing control

0: minimal delay

1: wait states to ensure address hold time

See *Appendix C: Using the ASTCR register at high MEMCLK frequencies* .

15.9 Keyboard/mouse interface

The keyboard and mouse interfaces are identical, differing only in the names of the external pins. The interfaces are designed to communicate with a standard PS/2 keyboard or mouse, via a 2 pin serial link. The keyboard interface uses the pins KBDATA, KBCLK, and the mouse interface uses the pins MSDATA and MSCLK, all of which are open drain.

There is an 8-bit control register for each interface, which provides direct access to the CLK and DATA outputs, an enable bit to enable the interface, and five status flags. The KBDCR is programmed at address 0x03200008, and the MSECR (mouse control register) at address 0x032000AC.

7	6	5	4	3	2	1	0
T	T	R	R	E	P	D	C

T = transmit status

R = receive status

E = enable

P = received parity

D = data pin status

C = clock pin status

Write	bits[7:4,2] ignored
	bit[3] enable, 0: state machine cleared, 1: state machine enabled
	bit[1] force **KBDATA**/**MSDATA** pin LOW, 0: don't force LOW, 1: force LOW
	bit[0] force **KBCLK**/**MSCLK** pin LOW, 0: don't force LOW; 1: force LOW
Read	bit[7] TXE, shift register empty, 0: not ready, 1: enabled and ready to transmit
	bit[6] TXB, transmitter busy, 0: not busy, 1: currently sending data
	bit[5] RXF, receive shift register full, 0: not full, 1: ready to read
	bit[4] RXB, receiver busy, 0: not busy, 1: currently receiving data
	bit[3] ENA, state machine enable, 0: disabled, 1: enabled
	bit[2] RXP, receive parity bit, odd parity bit for last received data
	bit[1] **KBDATA**/**MSDATA** pin value after synchronisation
	bit[0] **KBCLK**/**MSCLK** pin value after synchronisation

There is also a data register (KBDAT) which is used both to write bytes to be transmitted across the serial link and to read bytes received. The KBDAT register is programmed at address 0x03200004, and the MSEDAT (Mouse data register) is programmed at address 0x032000A8.

The interfaces generate two interrupts each, one to indicate that the transmit buffer is empty and thus that another byte can be transmitted, and one to indicate that a byte has been received by the interface. These interrupt bits are processed by the IRQB register set (for Keyboard) and the IRQD register set (for Mouse).

The keyboard interface is held in reset until the enable bit in the control register is set. The interface can be controlled on the basis of the interrupts generated, or by polling the status flags in the control register. The Tx interrupt is generated when the transmit buffer has been emptied and the interface is ready to be programmed with another character for transmission. The Rx interrupt is set when a complete character has been received in the receive buffer, and the byte is ready to be read from the register. The received data parity bit, RXP, is available in the control register at bit 2. Odd parity is used. The keyboard and mouse interface state machines are clocked by the 8MHz I/O system clock.

The **KCLK**/**MSCLK** signal is always driven by the keyboard/mouse, unless ARM7500 wishes to prevent the peripheral from transmitting (because it is about to transmit some data itself). When data is received from the peripheral, the **KDATA**/**MSDATA** line is pulled low as a start bit. Each data bit is set up to the falling edge of the clock. Eight data bits are transmitted from the keyboard/mouse, followed by a parity bit (odd parity) and a HIGH stop bit. The diagram below shows the protocol of this transfer.

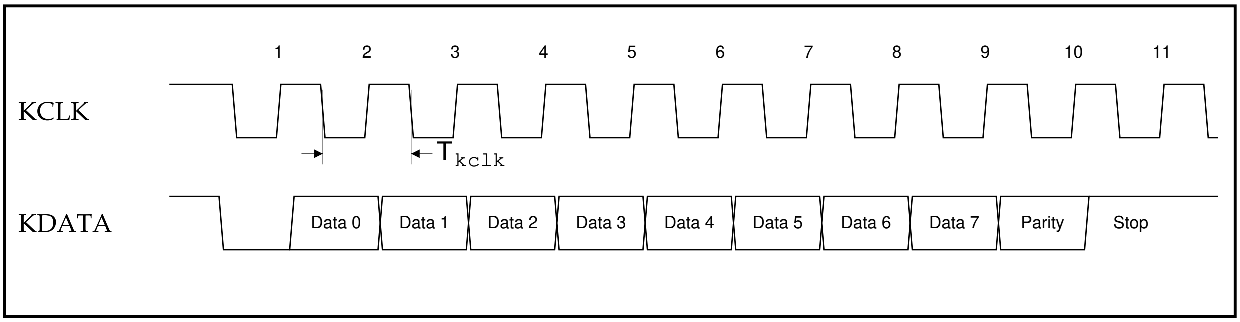


Figure 15-21: ARM7500 Keyboard/mouse controller receive protocol

When ARM7500 transmits a byte to the peripheral, the **KCLK**/**MSCLK** line is pulled LOW, then allowed to float and the **KDATA**/**MSDATA** line is pulled LOW, as a request to send. The keyboard/mouse then drives the clock, causing ARM7500 to put eight bits of serial data out onto the **KDATA**/**MSDATA** line. A parity bit is driven out, followed by a stop bit, and the stop bit may be acknowledged by the peripheral (the ARM7500 does not check on the acknowledge). The timing requirements of the interface are shown in the diagram below.

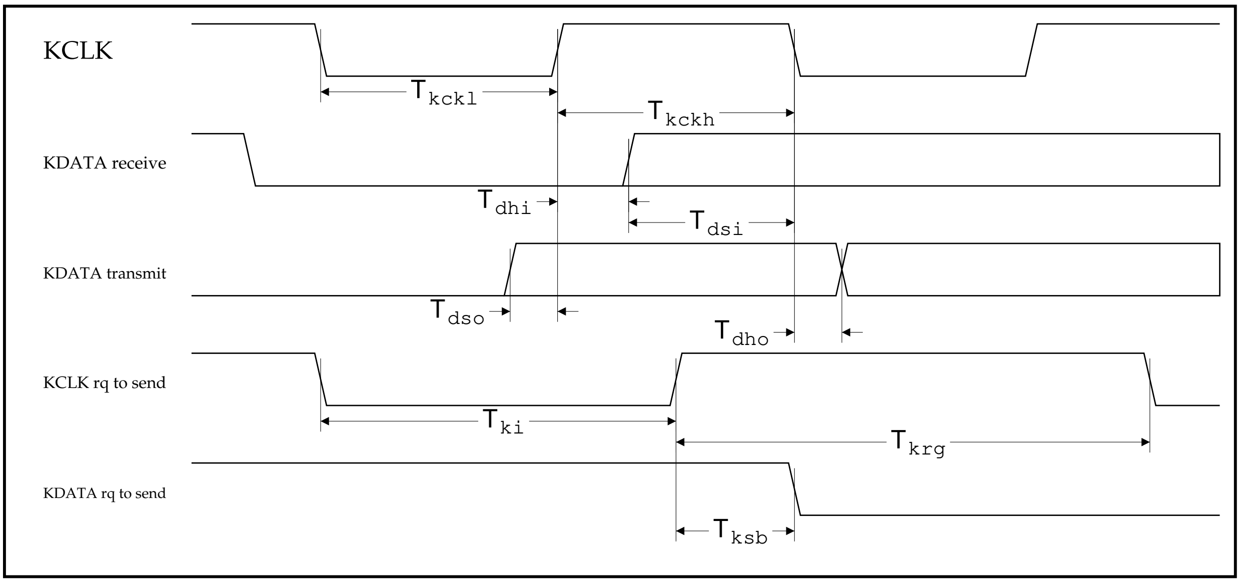


Figure 15-22: Keyboard/mouse interface timing

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tkclk	keyboard clock period	1		100	µs	
Tkckl	keyboard clock low time	0.5		50	µs	
Tkckh	keyboard clock high time	0.5		50	µs	
Tdhi	hold on DATA from CLK rising for Receive	1		Tkckh - 1µs	µs	
Tdsi	setup on DATA to CLK falling for Receive	1		Tkckh - 1µs		
Tdso	setup on DATA to CLK rising for Transmit	Tkckl - 1µs		Tkckl		
Tdho	hold on DATA from CLK falling for Transmit	0ns		1µs		
Tki	time for which CLK is held low to request a send	63.5	64	64.5	µs	
Tkrg	clock low from ARM7500 to clock low from keyboard for request to send	1			µs	
Tksb	clock low to data low hold time for request to send	1			µs	

Table 15-6: Keyboard/mouse cycles

15.10 Analog to digital converter interface

ARM7500 contains four analog comparators with 16 bit timers, which are designed primarily for the implementation of an analog joystick interface. Each converter is of the slope integration type, using an external RC network attached to the appropriate **ATOD[3:0]** pin to generate a variable ramp delay. The time taken for the voltage at the input to the comparator to reach the comparator's threshold is measured by a 16-bit counter which is stopped when the threshold of the comparator is reached. At this point an internal "stop" flag for that channel is set. The value is held in the counter until it has been read and the channel is then reset. Discharge transistors on the analog inputs are used to discharge the external capacitor and to initiate a new integration cycle.

15.10.1 Counters

Each of the four counters can be reset by programming one of four bits in the ATODCR register. The four counters cannot be written to but can be read at addresses as follows:

CNT1 (0x032000EC) - counter 1

CNT2 (0x032000F0) - counter 2

CNT3 (0x032000F4) - counter 3

CNT4 (0x032000F8) - counter 4

The four counters have been implemented as simple asynchronous ripple counters, and it is therefore important that they should not be read until the ‘stop’ flag for that particular channel has been set, as seen in the status register, to indicate that the counter has been stopped and the read back value will be stable.

15.10.2 Interrupt control

There is a single bit in the main ARM7500 interrupt handling registers (bit 2 of the IRQD set) which can accept an interrupt from the A to D converters. Thus some interrupt pre-processing is done to determine how this main interrupt is to be generated. An interrupt control register is provided so that various combinations of channels can generate the final interrupt. There are four possible interrupt sources, one for each channel, and each channel attempts to generate an interrupt when the comparator threshold is reached and the ‘stop’ flag is set internally. Each of these interrupt sources can be individually enabled using the lower four bits of the Interrupt Control register, and the upper four bits determine which combination of bits will create the main interrupt which is passed to the IRQD registers.

Address 0x032000E0 - Interrupt Control

7	6	5	4	3	2	1	0
S	F	A	C	4	3	2	1

1 = channel 1 interrupt enable

2 = channel 2 interrupt enable

3 = channel 3 interrupt enable

4 = channel 4 interrupt enable

C = any combination of channels generates nIRQ

A = only all channels enabled generates nIRQ

F = first pair enabled generates nIRQ

S = second pair enabled generates nIRQ

Write	bit[7:0] 0: disabled, 1: enabled
Read	return above values
Reset	reset to 0x0F

Note: *The OR of bit[3:0] is used to power up all the comparators. Thus they reset to the powered up state.*

15.10.3 Status of interface

The status of the 'stop' flag for each channel can be read directly from bits 0 to 3 of the status register, as can the interrupt status, which is simply the logical AND of the 'stop' flag values and the corresponding channel enables from the interrupt control register. This register should be read by the system software in a polled system to check whether a channel has reached its final count value and is thus waiting to be read before another conversion cycle can be initiated.

Address 0x032000E4 - Status

7	6	5	4	3	2	1	0
R	R	R	R	S	S	S	S

R[3:0] = interrupt request state for channels 4 to 1

S[3:0] = stop flag for channels 4 to 1

Write	ignored
Read	bit[7:4] 0: not requesting, 1: requesting
Reset	set all zero (not requesting)

15.10.4 Control

The converter control register allows the discharge transistors and counters for each channel to be enabled and disabled, to give full control over the resetting of the counter and the timing of the start of a conversion cycle. Before a conversion can be started, the discharge bit and the counter clear bit for the channel in question should be forced one and zero respectively, and then the bits should be returned to zero and one respectively to actually initiate a conversion cycle. This will cause the analog voltage across the external capacitor to begin to ramp up, and simultaneously the 2MHz clock to the counters will be enabled, thus starting the count. Synchronisation between the memory system clock which is used to program the registers, and the 2MHz I/O world clock results in a small extra delay before the counter is really enabled, but this is negligible against the 0.5 s period of the 2MHz clock.

Address 0x032000E8 - Converter control

7	6	5	4	3	2	1	0
D	D	D	D	C	C	C	C

D[3:0] = discharge transistor control for channels 4 to 1

C[3:0] = clear counter for channels 4 to 1

Write	bit[7:4] 0: transistor off, 1: transistor on (discharge) bit[3:0] 0: clear counter, 1: enable counter
Read	return above values
Reset	set all zero (clear counters and don’t discharge)

15.10.5 Comparators

The comparators are accurate to 2.5mV resolution and require a stable reference voltage of less than 2.5V to function correctly. The reference voltage is applied at the **ATODREF** pin. The same reference voltage is routed to all four comparators. In order for the comparators to function correctly, it is essential that the reference current to the Video DACs on the VIREF pin is present, as this current is used to generate the operating current used by the gain stages in the comparator. The comparator reference currents are disabled to save power if all the interrupt enables (bits 0 to 3 of the interrupt control register) are set to zero. Hence at least one channel must be enabled for any of the channels to function correctly.

15.10.6 Converter operation

The values of the capacitance and variable resistance used in the external RC circuit will determine the range of time delays which will be seen from the moment the capacitor begins to charge to the moment that the comparator threshold is crossed. The 16-bit counters are clocked by the 2MHz internal clock (derived from the 32MHz **I_OCLK**), and thus the counter will count for 65536 values over 32.7ms before returning to zero. In order to provide a meaningful reading from the converter, it is important that the capacitor and variable resistor values are such that this time will not be exceeded under the worst case conditions. The A to D converter is effectively providing a digital count directly related to the value of the resistance in the RC circuit.

15.11 Timers

The ARM7500 includes two general purpose timers which can be used as interrupt sources. Each timer is implemented as a 16-bit down counter, and has an input latch and an output latch associated with it. The counter decrements continuously, clocked at 2MHz. When it reaches zero, it is reloaded from the input latch and the down count recommences.

There are four eight bit wide registers associated with the two timers. Each timer has two eight bit registers corresponding to the 16-bits of the timer, and two further write-only registers which cause the GO and LATCH commands to be issued to the appropriate timer when written to. The diagram below shows the timer configuration.

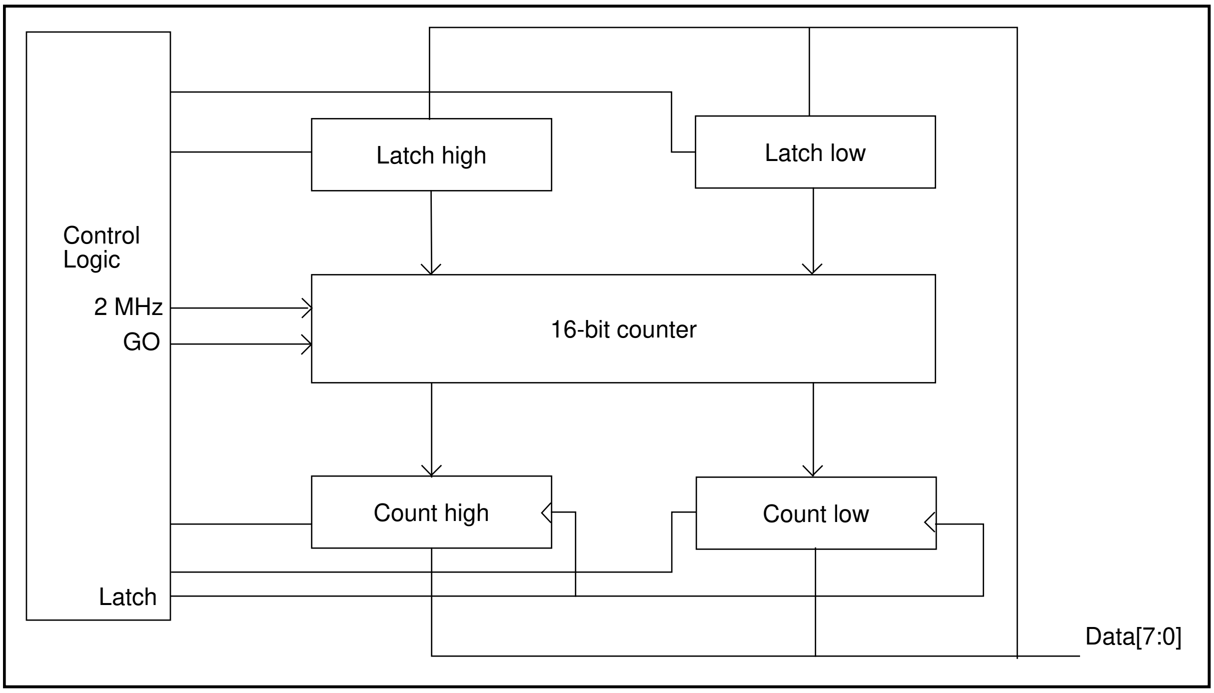


Figure 15-23: Timer configuration

15.11.1 Programming the timers

The locations of the registers can be found in *Chapter 13: Memory and I/O Programmer's Model*. Writing to the T0LOW register causes the value in the lower half of the timer 0 input latch to be updated. Writing to the T0HIGH register causes the value in the upper half of the timer 0 input latch to be updated with the written value. Writing to the T0GO register causes the counters to be loaded immediately with the value programmed into the input latch. If the counter is loaded with zero it will continuously reload.

Writing to the T0LATCH register causes the current count value to be placed in the output latch. Reading the T0HIGH register will then return the upper 8 bits of the count value, and reading the T0LOW register will return the lower 8 bits of the count value.

15.11.2 Timer interrupts

Each timer will generate an interrupt when it reaches zero and is reloaded. These interrupts are handled by the IRQA set of interrupt processing registers (bits 5 and 6). The timers can be used to generate timed interrupts at regular intervals T, where T = (T0LOW + (256 * T0HIGH)) * 0.5 μs.

15.12 General purpose, 8-bit wide, I/O port

A general purpose 8-bit wide I/O port is included in the ARM7500. The eight open drain output pins **IOP[7:0]** can be driven LOW or monitored as inputs by using the IOLINES register at address 0x0320000C. When read, this register will return the current value seen at the **IOP[7:0]** pins. When written to, each bit will control the status of the corresponding **IOP** pin. When a one is written to a bit, that pin's output enable is switched off and it can be driven as an input. When a zero is written to a bit, the corresponding output pin is forced LOW.

There is a complete set of three interrupt control and status registers (IRQD) for the **IOP** pins, which allow any bit to generate a unique interrupt. The interrupt is generated when the corresponding **IOP** bit is LOW.

15.13 ID and OD open drain I/O pins

There are three further open drain I/O pins, **ID** and **OD[1:0]**. These are written to via the IOCR register, and are not capable of generating interrupts. Each pin is forced LOW by programming a zero to the appropriate bit in the IOCR register. Programming a one to any bit causes the corresponding pin to be tristated, and the value of the input level applied to the pin can then be read back from the same bit of the IOCR register. The **OD[1:0]** pins could be used to implement a simple serial link. The **ID** pin is intended to be used with an ID chip, which outputs a unique system ID when the **ID** pin is forced LOW. During Power On Reset the **ID** output is forced LOW, and it then becomes tristate on leaving reset. It should be noted that these three pins do not have pull ups on-chip, and so it is advisable to fit them externally if they are not connected to another device.

15.14 Version and ID registers

The ID register is composed of two 8-bit hardwired registers which are read only. The lower byte is accessed at location 0x03200094, and the upper byte at location 0x03200098. Together they should return the value 0x5B98. The Version register is accessed at location 0x0320009C, and this will read back the version number of the device. Under no condition should either of these registers be written to, as this may cause the chip to enter a test mode.

15.15 Interrupt control

The ARM7500 interrupt handler takes interrupts from a variety of sources and generates the IRQ or FIQ interrupt signals required by the ARM Processor, depending on the settings of the control and enable bits in the five sets of interrupt registers. The five sets are FIQ, IRQA, IRQB, IRQC and IRQD. Each of these has a status, mask and request register associated with it, giving a total of 15 registers.

The table below shows the interrupt sources featuring in each set of registers. The polarity entry refers to the level required at the external pin to set the interrupt. ‘Internal’ means that the interrupt is generated as a result of an internal state change, as opposed to change on an external pin.

Register	Bit	Polarity/Type	Name/Function
FIQ	7		Always active for software generated **FIQ**.
	6	LOW	**nINT8** interrupt pin
	5		
	4	LOW	**nINT6** interrupt pin
	3		
	2		
	1	HIGH	**INT5** interrupt pin
	0	HIGH	**INT9** interrupt pin
IRQA	7		Always active for software generated **IRQ**.
	6	internal	2MHz timer 1
	5	internal	2MHz timer 0
	4	falling edge	**nPOR** power on reset
	3	internal	Flyback from video subsystem
	2	falling edge	**nINT1** interrupt pin
	1		
	0	rising edge	**INT2** interrupt pin
IRQB	7	internal	Keyboard Rx buffer full
	6	internal	Keyboard Tx buffer empty
	5	LOW	**nINT3** interrupt pin
	4	LOW	**nINT4** interrupt pin
	3	HIGH	**INT5** interrupt pin
	2	LOW	**nINT6** interrupt pin
	1	HIGH	**INT7** interrupt pin
	0	LOW	**nINT8** interrupt pin
IRQC	7	LOW	**IOP[7]** interrupt pin
	6	LOW	**IOP[6]** interrupt pin
	5	LOW	**IOP[5]** interrupt pin
	4	LOW	**IOP[4]** interrupt pin

Table 15-7: Interrupt table

Register	Bit	Polarity/Type	Name/Function
	3	LOW	**IOP[3]** interrupt pin
	2	LOW	**IOP[2]** interrupt pin
	1	LOW	**IOP[1]** interrupt pin
	0	LOW	**IOP[0]** interrupt pin
IRQD	7		
	6		
	5		
	4	LOW	**nEVENT2** wakeup event
	3	LOW	**nEVENT1** wakeup event
	2	internal	A to D convertor interrupt
	1	internal	Mouse Tx buffer empty
	0	internal	Mouse Rx buffer full

Table 15-7: Interrupt table (Continued)

When an interrupt signal is received from one of the interrupt sources, it causes the corresponding bit in the status register to go HIGH. This bit is then logically ANDed with the appropriate bit in the mask register, to create a value in the appropriate bit of the request register. If any of the bits in any of the IRQ request registers are HIGH, then the ARM7500 will generate an internal IRQ interrupt to the ARM Processor processor macrocell, causing the IRQ exception to be taken. If any of the bits in the FIQ request register are HIGH, the ARM7500 will generate an internal FIQ interrupt to the ARM Processor, causing the FIQ exception to be taken.

The system software can then read the request registers to determine which sources were requesting an interrupt. Reading the status registers will show which sources were requesting interrupts, even if they were masked.

The IRQA request register is slightly different in that some of the interrupt flags are edge triggered and thus need to be cleared after they have been read. All other request registers are read only, but the IRQRQA register can be written to clear triggered interrupts. Writing a one to a bit clears that interrupt. Writing a zero causes no action to be taken.

16 Clocks, Power Saving, and Reset

This chapter describes clock control, power management, and reset.

16.1 Clock control

ARM7500 has a clocking scheme designed to allow maximum flexibility for the system designer. There are three main clock inputs:

CPUCLK CPU clock, used to generate the ARM Processor's FCLK

MEMCLK Memory subsystem clock, used to generate the memory system clock, and the ARM Processor's MCLK

I_OCLK I/O system clock, which should be fixed at 32MHz, and is used to generate all the fixed frequency I/O clocks and refresh rates.

16.1.1 Video and sound subsystem clocks

The video sub-system has two separate external clock inputs and includes a phase locked loop to enable the control of an external VCO. The pixel clock source can be selected to be **VCLKI** (using an external VCO), **HCLK**, which is driven directly in from the **HCLK** pin, or IOCK32, which is the internal I/O subsystem clock and is generated directly from the main I_OCLK input pin as described below. The sound subsystem can be clocked either from IOCK32 generated internally from **I_OCLK**, or by using an externally generated clock connected to the **SCLK** pin. Selection between these various clock sources is described in the video and sound sub-systems section of this data sheet.

16.1.2 I/O clock outputs

Four fixed frequency I/O clocks are output by the ARM7500, all divided down from the fixed frequency input **I_OCLK** which should be set to 32MHz in divide by 1 mode. These are **CLK16** (16MHz), **REF8M**, and **CLK8** (8MHz and an inverted version), and **CLK2** (2MHz).

16.1.3 Synchronous/asynchronous mode for the ARM Processor

The ARM Processor macrocell can be configured to work in synchronous or asynchronous mode, under the control of the **SnA** pin. Synchronous mode can only be used within the ARM7500 if FCLK and MCLK are of exactly the same fundamental frequency and timing, and in fact when **SnA** is set HIGH, **MEMCLK** will be routed straight through to drive both FCLK and MCLK, with a suitable delay to ensure the required phase relationship between FCLK and MCLK is held correctly, ie CPUCLK is ignored when **SnA** = 1. If the **FCLK** frequency is required to be different from the **MEMCLK** frequency, the **SnA** pin must be held LOW, and a suitable frequency applied to **CPUCLK**.

16.1.4 Clock prescalers

Each of the three main clock inputs **CPUCLK**, **I_OCLK** and **MEMCLK** has a selectable divide by 2 prescaler available within ARM7500 to enable a guaranteed 50:50 mark-space ratio internal clock to be produced using a higher frequency external oscillator. The internal clocks, which will be referred to elsewhere in this data sheet, are called FCLK, IOCK32 and MEMRFCK respectively. On Power On Reset, all the prescalers will be set to divide by 2. The prescaling is controlled by the CLKCTL register at address 0x0320003C, and there is one bit to enable or disable each divide by 2 prescaler as required:

7	6	5	4	3	2	1	0
X	X	X	X	X	C	M	I

C = **CPUCLK** divide control

M = **MEMCLK** divide control

I = **I_OCLK** divide control

Write bit[2] 0: FCLK x 2 = **CPUCLK**, 1: FCLK = **CPUCLK**
bit[1] 0: MEMRFCK x 2 = **MEMCLK**, 1: MEMRFCK = **MEMCLK**
bit[0] 0: IOCK32 x 2 = **I_OCLK**, 1: IOCK32 = **I_OCLK**

Read return above value

Power On Reset

set all to zero, ie divide by 2 clocks

16.1.5 Clocking schemes

The simplest mode of operation of the ARM7500 has all three of the main clocks driven by a single 32MHz oscillator, with the prescalers set to divide by 1 mode. However, it is possible to increase the speed of the memory and CPU clocks, noting that if this causes **CPUCLK** and **MEMCLK** frequencies to be different, the **SnA** input must be set LOW for asynchronous operation. The **I_OCLK** frequency must remain at 32MHz (or 64MHz if the divide by 2 prescalers are enabled).

Increasing the memory clock frequency allows the system designer to take advantage of faster DRAM memory. The ARM7500 includes full synchronisation at the interface between the memory and I/O sub-systems to ensure safe operation under asynchronous conditions.

16.2 Power management

The ARM7500 includes power management circuitry which greatly enhances its suitability for battery powered portable applications where power consumption is of paramount importance. There are three power management modes:

NORMAL mode

This is the default operating condition in which all clocks are running and the chip is functioning normally.

SUSPEND mode:

In this mode, the clock to the CPU (FCLK) is stopped, but all other parts of the chip remain active so DMA can continue and the display can continue to be refreshed. It is also possible to stop some of the external I/O clock outputs to save more power if this can be done safely without causing problems for I/O peripherals connected to these clocks.

STOP mode:

This mode allows all the clocks to the ARM7500 to be stopped, and the whole chip will then draw only leakage currents provided all required registers have been appropriately programmed. Outputs are provided from the ARM7500 to enable the oscillator(s) to be powered down, and circuitry to allow the oscillator(s) to cleanly restart using an external RC delay before the clocks inside the ARM7500 are re-enabled. Before STOP mode is entered, a number of registers need to be programmed appropriately in the video sub-system, and further details of the full sequence of events required to make most effective use of the power management features can be found in ➲*16.2.7 STOP mode*.

16.2.1 SUSPEND mode

Entry into SUSPEND mode is achieved by writing to the register location 0x0320001C. Any value can be used, but the value written to bit 0 will determine whether the external I/O output clocks **CLK16**, **CLK8**, **REF8M** and **CLK2** are stopped. DMA may continue unaffected, allowing the display and DRAM data to remain refreshed.

Exit from SUSPEND mode is achieved by a falling edge on either of the asynchronous input event pins, **nEVENT1** and **nEVENT2**, or by any enabled interrupt source generating a FIQ or IRQ interrupt for the ARM Processor. The assertion of **nRESET** will also cause exit from SUSPEND mode. It is important that the interrupt mask and enable registers are programmed appropriately before SUSPEND mode is entered if it is intended that an interrupt source be used to terminate the power saving mode.

The CPU will merely see SUSPEND mode as a write to a location in the memory and I/O register area. It will be unaware of the duration of this write, as both **MCLK** and **FCLK** are frozen, and it is a fully static device. The careful use of SUSPEND mode when no CPU operations are required will have a significant effect on the device‘s average power consumption. It could be used, for example, between key presses while waiting for more user input. The keyboard controller is still clocked during SUSPEND mode and so will be able to generate interrupts which will cause the termination of the write cycle and then cause the CPU to take the interrupt exception.

Details of the SUSMODE register (address 0x0320001C) are shown below:

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	S

S = SUSPEND mode control of external I/O clocks

Write turn off external I/O clocks when in this mode

0: turn off, 1: don't turn off

Enter Suspend mode with MCLK,FCLK,I/O clocks and some internal clocks stopped. DMA continues and instruction completes on either wakeup event, nIRQ or nFIQ.

Read	return above value
Reset	set to zero

16.2.2 STOP mode

Entry into STOP mode is achieved by writing to the register location 0x0320002C. Any value can be written to the register to enter STOP mode, but the value written will appear on the external data bus of the ARM7500 while the chip is in STOP mode. It is therefore recommended that the value 0xFFFFFFFF be written to this register as this will mean that both **D[31:0]** and **LA[28:0**] are driven HIGH during STOP mode.

It is very important that all DMA activity is stopped, that all I/O activity is completed, and that the video subsystem is powered down correctly before the STOP mode register is written to. The **OSCPOWER** output is controlled by the power management circuitry, and will be forced LOW a short time after the write cycle begins. This output may be used to disable the external oscillator(s).

Exit from STOP mode can only be achieved by the use of the asynchronous wake up event pins **nEVENT1** and **nEVENT2**. When either of these is forced LOW, a sequence of events will be triggered which will cause the oscillator(s) to be restarted cleanly. During STOP mode, a zero is driven out from the **OSCDELAY** pin, which ensures that an external capacitor forming part of an RC network attached to the **OSCDELAY** pin remains discharged. As soon as a wake up event occurs the **OSCPOWER** pin is set HIGH again, and the open drain **OSCDELAY** pin is allowed to float and becomes an input. At this point the external capacitor starts to charge, until the schmitt threshold of the **OSCDELAY** input is exceeded. From this point, a further two rising edges must be seen on the input clock from the oscillator before the clock is allowed through to the internal ARM7500 circuitry. The component values used in the RC circuit should be chosen to ensure that the oscillator has sufficient time to stabilise before the **OSCDELAY** input is triggered.

As the video subsystem is inherently dynamic for performance reasons, it is necessary to set it into a special Powerdown mode before STOP mode is entered. To do this, the video Ext register should be programmed with the data 0xC0000000, the Video Control register should be programmed with the data 0xE00040xx (the last byte will depend on the clock source and configuration), and the Sound Control register should be programmed with the data 0xB1000000 (if the sound system is configured for use with the **SCLK** pin as the clock source. If the sound system is being clocked from the ARM7500's internal 32MHz I/O clock, then the register should be programmed with the value 0xB1000001). These actions will disable the video datapath and ensure the entire macrocell is forced into a static state. To ensure that the comparators in the A to D converters do not consume current, they should be shut down by programming the value 0x00 to the ATODICR register at location 0x032000E0.

ARM7500 includes support for self refresh DRAM, and it is intended that this feature should be used during STOP mode to ensure that DRAM contents are preserved. This DRAM mode is activated by allowing direct software control of the **nCAS** and **nRAS** output pins. The SELFREF register (0x032000D4) can be used to directly force the **nRAS** and **nCAS** output pins according to the protocol required for a particular DRAM, in order to enter self-refresh mode. This programming must be performed by code executing from ROM.

In STOP mode ARM7500 will consume leakage currents only, and can be held indefinitely without corruption of the internal registers, CPU cache, etc.

16.3 Reset

The ARM7500 has three pins associated with reset. The **nPOR** pin is intended for use with an external RC delay to generate a power-on-reset pulse when the chip is switched on. The **nRESET** pin is an open drain I/O pin, which is intended to be used to generate a “soft” reset. Both **nPOR** and **nRESET** are active LOW schmitt inputs. The active HIGH **RESET** pin is a clean reset output, which is created from the synchronised version of the **nRESET** input, and is also forced HIGH during **nPOR**.

A LOW state on the **nPOR** input causes the POR bit in the IRQA status register to be set. This bit can later be examined to show that the reset which occurred was an **nPOR** type rather than **nRESET**. The POR bit in the IRQA status register is not reset until the POR clear bit in the IRQA request register is written to. **nPOR** also causes the prescalers on the clock inputs to be set to divide by 2. The **nPOR** input is passed through a pulse stretcher which ensures that even a short pulse on the input will guarantee a full reset of the whole of ARM7500. See *Figure 16-1: nPOR timing diagram*. During **nPOR** reset, **nCAS** is forced low throughout and the **nRAS** outputs are changed according to the sequence in *Figure 14-8: Refresh cycle timing* on page 14-15. While **nPOR** is LOW, **nRESET** and **ID** (which are both open drain pins) are held LOW, and an incrementing address value will be output on the **LA** address bus.

A LOW state on the **nRESET** input is used to generate a 'soft' reset. This does not set any interrupt flags, and the **nRESET** LOW state must exist for longer than 1us to guarantee that it is seen, as it is passed through a synchroniser before being used by the internal circuitry. *Figure 16-2: nRESET timing diagram* below shows the required timing of **nRESET** to ensure correct operation. At the start of the **nRESET** active period, the whole ARM7500 (including the DRAM refresh state machine and counter) is reset for 1us, and for the remaining duration of the **nRESET** pulse, DRAM refresh takes place at the highest selectable rate. During **nRESET**, the ARM Processor outputs an incrementing address on the **LA** bus.

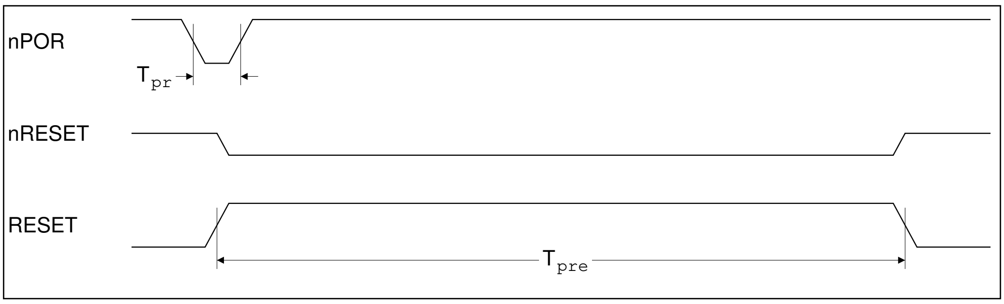


Figure 16-1: nPOR timing diagram

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tpr	time for which nPOR must be held low to guarantee a reset	20			ns	
Tpre	length of internal reset	2			µs	1

Table 16-1: nPOR and nRESET timing

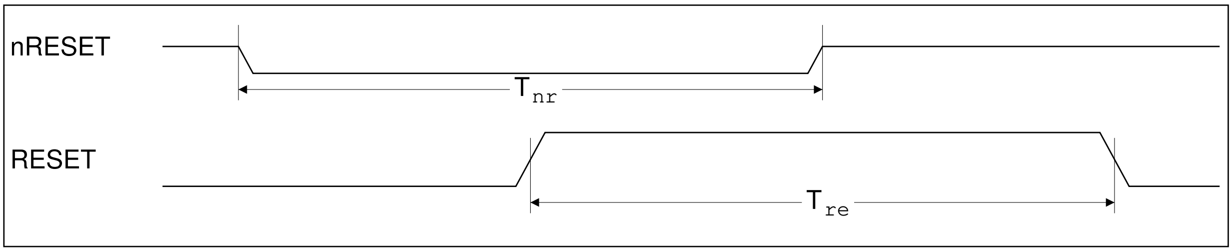


Figure 16-2: nRESET timing diagram

Symbol	Parameters	Min	Typ	Max	Units	Notes
Tnr	time for which nRESET must be held low to guarantee reset	2			µs	2, 3
Tre	length of internal reset	2			µs	3

Table 16-2: nPOR and nRESET timing

1 Assuming I_OCLK is 64MHz. Double this time if I_OCLK is 32 MHz as this reset forces divide by 2 mode on the clock inputs.

2 DMA or writes from the ARM Processor prevent **nRESET** having any effect for their duration. Thus the “soft” reset cannot break write cycles or cause partial DRAM refresh.

3 Assuming IOCK32 is 32MHz.

When in STOP mode, **nRESET** will force the power management control circuitry to revert to normal mode, without necessarily causing a reset sequence to occur.

17 Bus Interface

This chapter describes the ARM7500 bus interface.

17.1 Bus arbitration

Arbitration for the main ARM7500 data bus is carried out with the priorities shown below:

1. Video/cursor DMA
2. Sound DMA
3. DRAM refresh
4. ARM processor memory cycles

As the ARM7500 contains a cached processor, ARM internal cycles can continue while DMA is in progress, but the CPU will stall when it suffers a cache miss and wishes to fill a cache line from memory.

Once an external memory cycle has started, DMA has to wait until it is completed. The exception is for I/O reads or writes and SUSPEND mode, where the write data is latched internally at the start of the cycle, after which DMA requests can be serviced even though the I/O access or SUSPEND mode is under way. The end of an I/O access is held up until the current DMA access is completed. I/O read data is latched internally when available, and is not enabled onto the ARM7500 data bus until any DMA transfers have completed.

17.2 Bus cycle types

There are a large number of different types of cycle which make use of the ARM7500 data bus. Except for DMA accesses, the cycle type is decoded according to the address put out by the ARM processor macrocell, and the detailed timing is controlled by the relevant section of the I/O or memory controller subsystem.

The ARM processor supports two basic types of external cycle: non-sequential and sequential. A non-sequential cycle consists of an Idle cycle followed by a memory cycle, and a sequential cycle consists simply of a memory cycle. The idle cycle allows the memory and I/O controller subsystems time to prepare for a new cycle type. These two cycles are used as the basic building block for the more complex I/O and memory access cycle timings generated by the ARM7500. ARM processor external cycles are clocked by the internal Mclk signal which is generated by the ARM7500's memory controller according to the type of cycle.

Only the latched version of the ARM processor's address is exported from the ARM processor, and this can only change immediately after the falling edge of the internal Mclk signal which clocks the ARM for external accesses. The timing diagrams in this datasheet include Mclk as a reference as it indicates the end of a particular cycle. The ARM7500 internal data bus is not always exported during internal register programming, to save power.

When the ARM processor requests an external memory access, it will do so for one of a number of reasons. A cache linefetch will always consist of memory reads from four sequential addresses. A level 1 translation fetch will consist of a read from memory followed by the address translation such that the next address put out by the ARM will be the translated physical address as generated from the read back section descriptor. A level 2 translation fetch is always preceded by a level 1 fetch, and returns the page table entry which is then used to create the physical address for the next cycle. External buffered and unbuffered write cycles take place with indistinguishable bus timing. When the ARM wishes to read from a location and the data is not in the cache or is uncacheable (eg for I/O), then an external read access is performed.

17.3 Video DMA bandwidth

The maximum video DMA bandwidth is dependent on the **MEMCLK** frequency and the DRAM width (16 or 32-bit), but can be calculated as follows:

Each quad word DMA requires 5+2+2+2 = 11 **MEMCLK** cycles to complete, and it is possible for DMA requests for the video to be serviced sequentially such that the second and subsequent quad word DMA burst takes only 2+2+2+2=8 **MEMCLK** cycles. However, all accesses will be broken up at page boundaries, that is every 256 words. So every 64 DMA bursts, there will be three extra **MEMCLK** periods required.

Therefore, at 32MHz **MEMCLK**, with 32-bit wide DRAM, 64 quad words would be transferred approximately every 16us. The maximum theoretical DMA bandwidth is thus 63.6MBytes/second. If a greater video DMA bandwidth than this is required, then a higher **MEMCLK** frequency will need to be used. Clearly, in a real system, the average bandwidth will not achieve this theoretical maximum.

17.4 Video DMA latency

DMA latency is defined to be the time from the generation of the internal request for more data from the video FIFO in the video macrocell, to the time at which the first word of DMA data is clocked into the video macrocell.

There are several possible limiting factors which may determine the worst case DMA latency, depending on the type of memory system with which ARM7500 is configured to be used. There are three possible limiting cases:

1. Internal register programming cycles
2. Burst mode ROM accesses, or very long non sequential ROM accesses
3. DRAM accesses in 16-bit mode

The following assumes that the internal **MEMRFCK** frequency is equal to the **MEMCLK** frequency, ie the prescalers are set to divide by one. The above cases determine the maximum period before arbitration for DMA occurs in different systems. In addition to the latency resulting from these sequences, the worst case latency has a possible 5.5 **MEMCLK** cycles factor for synchronisation, such that the synchronised request arrives just too late to be arbitrated for, and ARM7500 commits to a memory cycle. This 5.5 **MEMCLK** cycles also includes the ARM processor idle cycle on which the arbitration (which was just missed) takes place.

From the clock edge at which arbitration finally takes place, to the time at which the first word of DMA data is clocked into the video macrocell, is 3.5 **MEMCLK** cycles.

Internal register programming bursts can occur in blocks of up to four before rearbitration takes place, and this will take 16 **MEMCLK** cycles. Burst mode ROM cycles are rearbitrated after every four, as are sequential DRAM accesses. Successive non sequential accesses will always allow DMA onto the bus, so it is unlikely that these will be the cause of the worst case latency. However, it would be possible to use the ROM interface in half speed mode, with the slowest ROM timing and a 16 bit wide ROM, in which case an access would take 28 **MEMRFCK** cycles. Under these circumstances the ROM interface could be the limiting factor.

To determine the limiting factor in a system, calculate the number of cycles required for a worst case ROM access. The number of cycles for each programmed value in the ROMCR register is shown below:

For a non sequential access, programming bits 0-2:

000 - 7 cycles	
001 - 6 cycles	For all:
010 - 5 cycles	Multiply by 2 if 16-bit mode set
011 - 4 cycles	Multiply by 2 if half speed bit set
100 - 3 cycles	
101 - 2 cycles	

If the burst bits (3-4) are programmed to a value other than 00, then the total worst case number of cycles will be one times the non sequential number above, plus three times the burst number from below:

01 - 4 cycles	For all:
10 - 3 cycles	Multiply by 2 if 16-bit mode set
11 - 2 cycles	Multiply by 2 if half speed bit set

Then calculate the number of cycles required for a worst case DRAM access. This can only be the limiting factor when 16-bit wide DRAM is used, and in this case the delay will be 5 + (2x7) = 19 cycles

As described above, the worst case delay for four sequential internal register programming cycles is 16 cycles. So the worst case delay is caused by internal register access cycles, ROM or DRAM according to which of the above calculated figures is worst.

DMA can continue over the top of I/O accesses, so these do not feature in the options for worst case delay. So for a system which is limited by internal register access cycles, the worst case latency will be 3.5 + 2 + 16 + 3.5 = 25 **MEMCLK** cycles. So if **MEMCLK** is running at 32MHz, the total worst case DMA latency will be 0.78us.

As another example, suppose that the ROM interface non sequential access time is programmed at 7 cycles, and the sequential programmed to 4, using 16-bit wide ROM. Then the total latency would be 3.5 + 2 + 14 + 8 + 8 + 8 +3.5 = 47 **MEMCLK** cycles. At 32MHz this corresponds to 1.4us.

18

Memory Map

This chapter gives details of the ARM7500 memory map.

18.1 ARM7500 memory map

All addresses featured in the ARM7500 memory map table are physical addresses. Only 29 bits of the address bus are available, which limits the total memory space to 512Mb.

Memory (Mbytes)	Address (Hex)	To (Hex)	Device
0	00000000	00FFFFFF	ROM bank 0
16	01000000	01FFFFFF	ROM bank 1
32	02000000	02FFFFFF	Reserved
48	03000000	0300FFFF	Module I/O space
	03010000	0302BFFF	16MHz PC style I/O
	0302C000	0302FFFF	Reserved
	03030000	0303FFFF	Further module I/O space
	03040000	031FFFFF	Reserved
	03200000	0320FFFF	ARM7500 registers
	03210000	033FFFFF	Simple I/O space
	03400000	034FFFFF	Video registers
	03500000	03FFFFFF	Reserved
64	04000000	07FFFFFF	Reserved
128	08000000	0FFFFFFF	Extended I/O space
256	10000000	DRAM bank 0	
320	14000000	DRAM bank 1	
384	18000000	DRAM bank 2	
448	1C000000	DRAM bank 3	
512	20000000	ROM bank 0 (repeated)	

Table 18-1: ARM7500 memory map table

19 DC and AC Parameters

This chapter gives the ARM7500 DC and AC parameters.

19.1 Absolute maximum ratings

Symbol	Parameters	5V Min	5V Max	Units	Notes
VDD	Supply voltage	VSS-0.3	VSS+7.0	V	1
Vip	Voltage applied to any pin	VSS-0.3	VDD+0.3	V	1
Ts	Storage temperature	-40	125	deg C	1

Table 19-1: ARM7500 DC maximum ratings

Note: *These are stress ratings only. Exceeding the absolute maximum ratings may permanently damage the device. Operating the device at absolute maximum ratings for extended periods may affect device reliability.*

19.2 DC operating conditions

Symbol	Parameters	Min	Typ	Max	Units	Notes
VDD	Supply voltage	4.75	5.0	5.25	V	
Vihc	IC input HIGH voltage	0.8xVDD		VDD	V	1, 2
Vilc	IC input LOW voltage	0.0		0.2xVDD	V	1, 2
Viht	IT input HIGH voltage	2.3V		VDD	V	1, 3
Vilt	IT input LOW voltage	0.0		0.6V	V	1, 3
Vihs	IS input HIGH voltage					1, 5
Vils	IS input LOW voltage					1, 5
Vohc	OCZ output HIGH voltage	0.9xVDD		VDD	V	1, 4
Volc	OCZ output LOW voltage	0.0		0.1xVDD	V	1, 4
Ta	Ambient operating temperature	0		70	C	

Figure 19-1: ARM7500 DC operating conditions

1 Note:Voltages measured with respect to VSS.

2 IC - CMOS inputs

3 IT - TTL inputs (includes BTZ, TOD, and IT pin types)

4 OCZ - Output, CMOS levels, tri-stateable (includes OCZ, BTZ, TOD, and CSOD pin types)

5 IS - CMOS Schmitt inputs (includes ICS and CSOD pin types)

19.3 DC characteristics

Symbol	Parameter	Nom	Units	Note
IDD	Static Supply current	100	µA	1
Isc	Output short circuit current	100	mA	
Ilu	DC latch-up current	>500	mA	
Iin	IC input leakage current	1	uA	
Ioh	Output HIGH current (Vout = VDD-0.4V)		mA	
Iol	Output LOW current (Vout = VSS+0.4V)		mA	
Vihst	IS input rising voltage threshold			2
Vilst	IS input falling voltage threshold			2
Cin	Input capacitance		pF	
ESD	HMB model ESD	4	KV	3

Figure 19-2: ARM7500 DC characteristics

1. When the video subsystem is correctly powered down and ARM7500 is in STOP mode
2. IS - Schmitt trigger input
3. This does not apply to video and sound analog pins: **VIREF**, **ROUT**, **GOUT**, **BOUT**, **SIREF**, **LM**, **LP**, **RM**, **RP**

19.4 AC parameters

The ARM7500 timing diagrams have been included in the appropriate sections of the datasheet.

19.5 Test conditions

The AC timing diagrams assume that the outputs of ARM7500 have been loaded with the capacitive loads shown in the *Test Load* column of the table below; these loads have been chosen as typical of the system in which ARM7500 might be employed. The output pads of ARM7500 are CMOS drivers which exhibit a propagation delay that increases linearly with the increase in load capacitance. An *Output derating* figure is given for each output pad, showing the approximate rate of increase of output time with increasing load capacitance.

Output signal	Test load (pF)	Output derating (ns/pF)
LA[28:0]	tba	tba
D[31:0]		
nCAS[3:0]		
nRAS [3:0]		
RA[11:0]		
ECLK		
ED[7:0]		
HSYNC		
PCOMP		
SDCLK		
SDO_MUTE		
VSYNC		
WS_LNR		
nROMCS		
BD[15:0]		
MSCLK		
MSDATA		
KBCLK		
KBDATA		
nIORQ		
nIOW		
nIOR		
nBLO		
nRBE		
nWBE		
nSIOCS1		
nSIOCS2		
nMSCS		

Table 19-2: ARM7500 AC test conditions

Output signal	Test load (pF)	Output derating (ns/pF)
nEACS	tba	tba
nCCS		
nCDACK		
TC		
nPCCS1		
nPCCS2		
LNBW		
IORNW		
CLK2		
CLK8		
REF8M		
CLK16		

Table 19-2: ARM7500 AC test conditions

20 Packaging

This chapter describes the physical details of the ARM7500.

20.1 Pin diagrams for the ARM7500

The following two diagrams illustrate the top and side views of the ARM7500. All dimensions are given in millimetres.

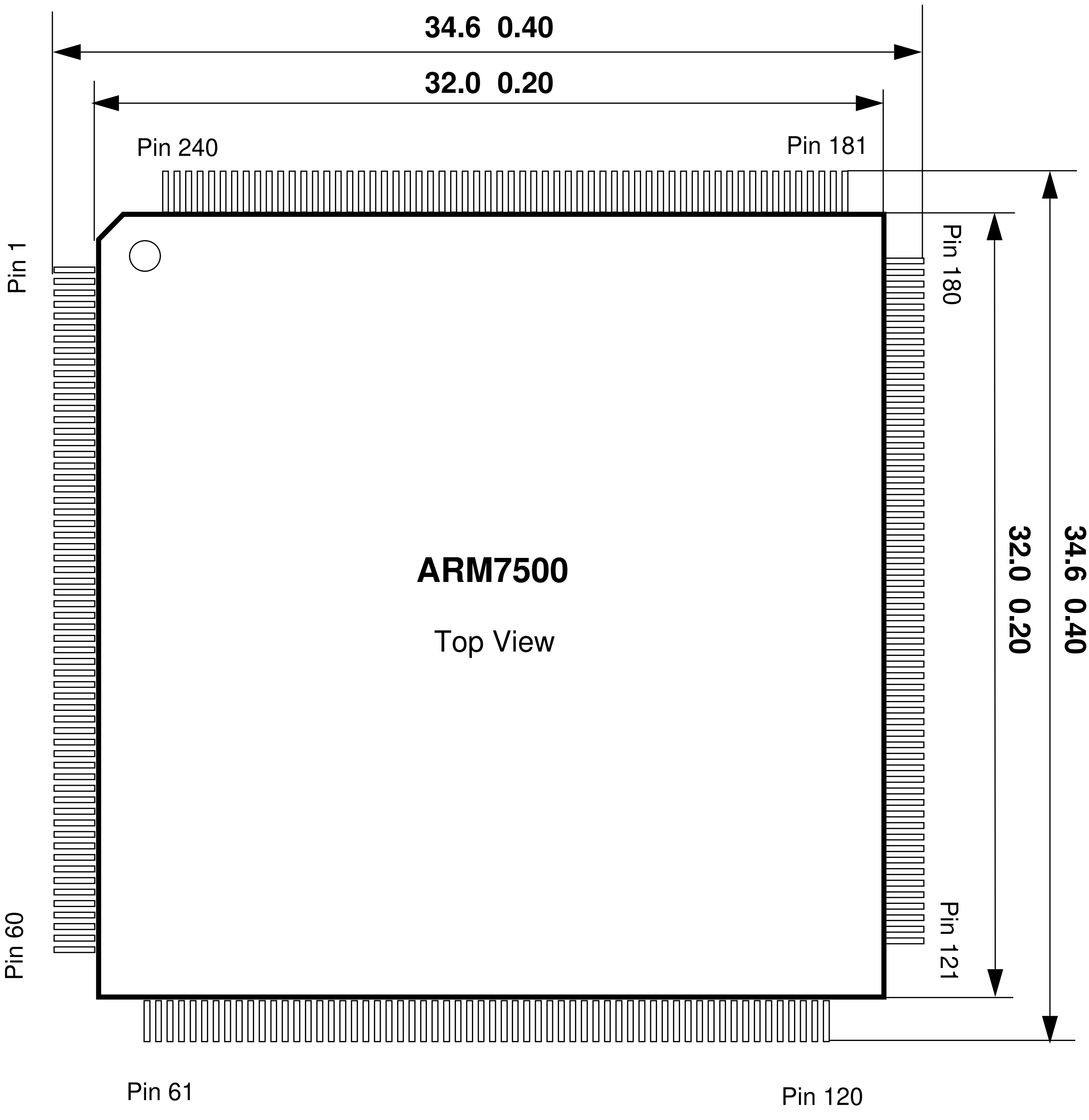


Figure 20-1: Pin diagram for the ARM7500

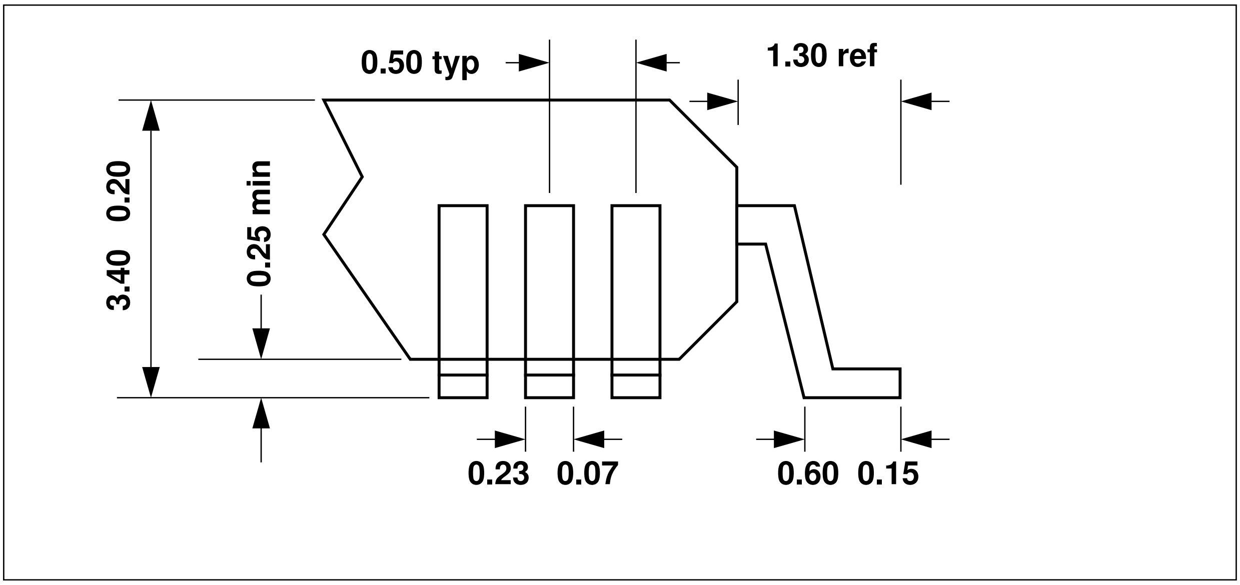


Figure 20-2: Side view of ARM7500 chip

21 Pinout

This chapter describes the ARM7500 pinout.

21.1 Pin details

The following table gives the signal name for each of the 240 pins of the ARM7500.

Pin number	Signal name	Pin number	Signal name
1	LA[15]	27	D[23]
2	LA[16]	28	D[22]
3	LA[17]	29	D[21]
4	LA[18]	30	VSS_CORE
5	LA[19]	31	D[20]
6	LA[20]	32	VDD_CORE
7	LA[21]	33	D[19]
8	VDD	34	D[18]
9	LA[22]	35	VSS
10	VSS	36	D[17]
11	LA[23]	37	D[16]
12	LA[24]	38	D[15]
13	LA[25]	39	D[14]
14	LA[26]	40	D[13]
15	LA[27]	41	VDD
16	LA[28]	42	D[12]
17	D[31]	43	D[11]
18	D[30]	44	D[10]
19	D[29]	45	D[9]
20	D[28]	46	D[8]
21	VSS	47	VSS
22	D[27]	48	D[7]
23	D[26]	49	D[6]
24	VDD	50	D[5]
25	D[25]	51	D[4]
26	D[24]	52	D[3]

Table 21-1: Pin numbers and signal names

Pin number	Signal name	Pin number	Signal name
53	D[2]	82	ED[2]
54	D[1]	83	ED[1]
55	D[0]	84	ED[0]
56	VDD	85	VSS
57	PCOMP	86	VSYNC
58	VSS	87	VSS_CORE
59	VCLKI	88	HSYNC
60	VCLKO	89	VDD_CORE
61	VDD_SOUND	90	VIREF
62	LP	91	VDD_ANALOG
63	RP	92	ROUT
64	RM	93	BOUT
65	LM	94	GOUT
66	VSS_SOUND	95	VSS_ANALOG
67	SIREF	96	nTEST
68	SDO_MUTE	97	nINT8
69	SCLK	98	nINT3
70	SDCLK	99	nINT6
71	WS_LNR	100	INT7
72	SINK	101	RA[11]
73	ECLK	102	RA[10]
74	VSS	103	RA[9]
75	HCLK	104	VSS
76	ED[7]	105	RA[9]
77	ED[6]	106	VDD
78	ED[5]	107	RA[7]
79	VDD	108	RA[6]
80	ED[4]	109	RA[5]
81	ED[3]	110	RA[4]

Table 21-1: Pin numbers and signal names (Continued)

Pin number	Signal name	Pin number	Signal name
111	RA[3]	140	nROMCS
112	RA[2]	141	BD[15]
113	RA[1]	142	BD[14]
114	RA[0]	143	I_OCLK
115	VSS	144	VSS
116	nRAS[3]	145	nEVENT2
117	VDD	146	BD[13]
118	nRAS[2]	147	BD[12]
119	nRAS[1]	148	BD[11]
120	nRAS[0]	149	VDD
121	VDD_ATOD	150	BD[10]
122	ATODREF	151	VSS_CORE
123	ATOD[3]	152	MEMCLK
124	ATOD[2]	153	VDD_CORE
125	ATOD[1]	154	BD[9]
126	ATOD[0]	155	BD[8]
127	VSS_ATOD	156	BD[7]
128	nCAS[3]	157	BD[6]
129	nCAS[2]	158	BD[5]
130	VSS	159	VSS
131	nCAS[1]	160	BD[4]
132	VDD	161	BD[3]
133	nCAS[0]	162	BD[2]
134	nWE	163	BD[1]
135	OSCPOWER	164	BD[0]
136	OSCDELAY	165	MSCLK
137	SnA	166	VDD
138	RESET	167	MSDATA
139	nRESET	168	KBCLK

Table 21-1: Pin numbers and signal names (Continued)

Pin number	Signal name	Pin number	Signal name
169	KBDATA	198	VDD
170	VSS	199	nSIOCS1
171	nPOR	200	nEASCS
172	IOP[7]	201	nMSCS
173	IOP[6]	202	nBLO
174	IOP[5]	203	nRBE
175	IOP[4]	204	nWBE
176	IOP[3]	205	CLK2
177	IOP[2]	206	REF8M
178	IOP[1]	207	CLK8
179	IOP[0]	208	CLK16
180	ID	209	nIORQ
181	OD[1]	210	VSS
182	OD[0]	211	nIOR
183	SETCS	212	VSS_CORE
184	INT9	213	CPUCLK
185	nINT4	214	VDD_CORE
186	INT5	215	nIOW
187	READY	216	VDD
188	nIOGT	217	nCCS
189	nBLI	218	nCDACK
190	nXIPMUX16	219	IORNW
191	nINT1	220	nPCCS2
192	INT2	221	nPCCS1
193	VSS	222	LNBW
194	nEVENT1	223	LA[0]
195	nXIPLATCH	224	LA[1]
196	TC	225	LA[2]
197	nSIOCS2	226	VSS

Table 21-1: Pin numbers and signal names (Continued)

Pin number	Signal name	Pin number	Signal name
227	LA[3]	234	LA[9]
228	LA[4]	235	LA[10]
229	LA[5]	236	LA[11]
230	LA[6]	237	LA[12]
231	LA[7]	238	VSS
232	LA[8]	239	LA[13]
233	VDD	240	LA[14]

Table 21-1: Pin numbers and signal names (Continued)

A Initialisation and Boot Sequence

This appendix describes the ARM7500 initialisation and boot sequence.

A.1 Introduction

ARM7500 is designed to operate with 16 or 32 bit wide memory systems. In order to avoid a hardware selection mechanism, the ARM7500 is designed to always power up with bit 6 of the ROMCR0 register set to 1, such that the chip expects to receive the first instructions from a 16 bit wide ROM bank. For a system which is actually using 16 bit wide ROM, no special action is required. For a system which uses 32 bit wide ROM, a software solution is needed to enable the chip to boot successfully.

An example method of programming the first locations of ROM in order to boot the device successfully is described in the following section. The example assumes that the reset vector is to be located at physical memory address zero.

A.1.1 Example boot sequence

The processor will start executing code from physical address 0. As ARM7500 is initially configured to operate with a 16 bit wide ROM, it will fetch the lower half-word of the first instruction from the lower 16 bits of address 0, and the upper half-word of the instruction from the lower 16 bits of address 4. If these first two locations have been programmed with instructions to load the PC with the reset and undefined instruction vectors, then the combination of the lower half-words from the first and second location always creates an instruction with a never-true condition code, and so execution will drop through to the next instruction. This will be true for all the LDR PC instructions in the exception table. The exception table occupying the first eight locations in ROM is shown below.

This vector table resides at physical address 0.

Address	Instruction
0	LDR PC, RESET_VEC
4	LDR PC, UNDEF_VEC
8	LDR PC, SWI_VEC
C	LDR PC, PREF_VEC
10	LDR PC, DATA_VEC
14	LDR PC, RES_VEC
18	LDR PC, IRQ_VEC
1C	LDR PC, FIQ_VEC

Table A-1: Vector table

Immediately after the table, the ARM7500 should be set into 32 bit mode. The eight locations from address 20 to 3C must be programmed with eight half words in the lower sixteen bits of each location, which will form the four required 32 bit instructions when read in pairs by the ARM7500. The upper 16 bits of each location will be ignored by the ARM7500 while still in 16 bit mode. The four instructions program the ROMCR0 register into 32 bit mode, and cause program execution to jump back to the reset vector at physical address zero, which will now be executed correctly. The MOV PC,#0 instruction which actually causes execution to jump back to zero will have been prefetched in 16 bit mode, even though it occurs after the ARM7500 ROMCR0 register has been reprogrammed. The table below shows the data required at memory locations 0x20 to 0x3C to implement this scheme. The boot code above is a general

Data	Address	Instruction	Notes
0x0000B632	20		
0x0000E3A0	24	MOV R11, 0x03200000	point at register base
0x00000000	28		
0x0000E3A0	2C	MOV R0, #&0	32b,slow,218.75us,no burst
0x00000080	30		
0x0000E5CB	34	STRB R0, [R11,0x80]	Program ROMCR0 & switch mode
0x0000F000	38		
0x0000E3A0	3C	MOV PC, #0	Jump to 0

Table A-2: Instructions for programming the ROM register

example which will set the ROM interface to use the slowest access timing, to ensure it will work with all systems. It is adviseable to program the ROM control registers early on with the fastest parameters useable by the interface, as this will drastically speed up execution. In addition, on power up the default state of the CLKCTL register is for the CPUCLK, MEMCLK and I_OCLK external clock inputs to be divided by 2, and these should be programmed to divide by 1 if appropriate. This will also speed up execution.

A.1.2 Other methods

The above method is an example of how the ARM7500 can be booted from a system using 32 bit wide ROM. There are other methods of doing this which may be more appropriate for the required application. The main advantage of the method described above is that it allows the exception vector table to reside at physical address 0. If this is not a requirement then the the instructions which reprogram the ROMCR0 register could reside from location 0 onwards, and the vector table can be mapped into DRAM by the operating system software.

B Dual Panel Liquid Crystal Displays

This appendix describes dual panel LCD driving within the video and sound macrocell.

B.1 Programming the video subsystem

The external register (address 0xC00xxxxx) bit 13 (lcd) must be programmed to one, as for normal LCD operation.

Bit 13 of the control register (address 0xE00xxxxx) must be programmed to one. This is the 'dup' bit to set duplex mode operational.

Video data will be channelled simultaneously to the top and bottom halves of the screen. The first quad word received from memory will be interpreted for the first part of the first raster in the top half of the screen, and the second quad word will be interpreted for the indentical part of the lower half of the screen. ARM7500 will handle the sequencing of DMA data so that the video buffer can still be programmed as though there was only one panel.

When the cursor is moved, in addition to the programming of the Vertical Cursor start (VCSR) and end (VCER) registers and the horizontal cursor start (HCSR) register as described below, bits 13 and 14 of VCSR (address 0x9600xxxx) should be programmed to:

14:13

0 0	Dual Panel mode not activated
0 1	Cursor in upper half screen
1 0	Cursor in lower half screen
1 1	Cursor straddles both halves

Normally VCSR defines the number of rasters from Vsync to the start of the cursor, and VCER defines the number of rasters from Vsync to the end of the cursor display. See *Chapter 9: Video and Sound Programmer's Model* for details of exactly how these are programmed.

For split screen operation, the programming of VCER and VCSR will be the same as for a single panel LCD when the cursor is completely in the top or bottom half of the screen, but when the cursor straddles the boundary, the values of these two registers will have a different meaning. The value in the VCSR register will be the number of rasters from the top of the lower panel to the end of the cursor image, and VCER will be programmed with the number of rasters from the top of the display to the start of the cursor image in the upper panel. The cursor is displayed in the lower half screen from the value of VDSR to VCSR, and in the upper half screen from the value of VCER to VDER. So effectively the start register is defining the "end" of the cursor in the bottom half and the end register is defining the "start" of the cursor in the top half. This is the case because the top of the lower half of the screen will be written to before the bottom of the upper half.

B.2 Configuring DMA within ARM7500

The video and sound macrocell must first be programmed to drive dual panel LCDs as above. Once this has been done the macrocell will always make quad word DMA requests in pairs. ARM7500 is then set into dual panel mode by programming bit 7 ("dup") of the Video Control register VIDCR (Address 0x1E0) to 1. The eight bits of the Video Control register are now allocated as follows

VIDCR (address 0x 1E0)

7	6	5	4	3	2	1	0
D	X	E	X	X	X	X	X

X = Undefined

E = Enable

D = Duplex LCD

When duplex mode is enabled, ARM7500 will DMA two quad words from memory, offset by half the size of the video buffer, to enable two parallel data streams to be output by the video and sound macrocell to the two panels of the LCD. All DMA is quad word only, so the auto increment of the DMA address is now always 0x10. The VIDSTART and VIDEND registers will be programmed in the normal way, as for a single panel, with the addresses of the first and last quad words in memory. The VIDINITA register should be programmed with the address of the first quad word to be displayed on the upper panel of the LCD, and the VIDINITB register with the address of the first quad word to be displayed on the lower panel of the LCD. The difference between the two addresses should be half the number of bytes in the video buffer. It is perfectly possible for VIDINITA to be pointing to an address in the lower half of the buffer, in which case VIDINITB should be set to point to an address in the top half of the buffer, offset by half the buffer size again.

If either of the INIT register values are equal to the END register, then bit 30 of the relevant INIT register must be set HIGH for correct operation (the “last” bit). Clearly both “last” bits should never be programmed HIGH at the same time.

B.2.1 Cursor

In order to ensure smooth transition of the cursor across the dual panel boundary, it is necessary to have four images of the cursor stored in memory. This is because the ARM7500 DMA registers must only be programmed with quad word aligned addresses, but as the cursor is always 32 pixels wide at 2 bits per pixel, the address of data corresponding to a particular row of the cursor may be aligned with a two word boundary.

The four images should be arranged as two pairs of contiguous images of the cursor. Only alternate rows of each cursor image will start on quad word boundaries, for reasons stated above, and so the two pairs of images are offset so that the first has all its odd rows starting on quad word boundaries, and the second has all its even rows starting on quad word boundaries. This means that ARM7500 can address any row of the cursor using only quad word aligned DMA pointers.

Normally only the first image will be used. However, when the cursor happens to be straddling the split screen boundary, a different strategy is adopted. The VCSR and VCER registers in the video and sound macrocell are programmed differently as described in the above, and the cursor init register must be set to point to the location corresponding to the position of the row of the cursor which appears at the top of the lower part of the screen. In conjunction with the different meaning of the vertical cursor position registers in the video and sound macrocell, this will enable a smooth transition across the boundary.

B.2.2 Video frame buffer restrictions

In order for the dual panel LCD to be driven correctly, it is necessary for the video frame buffer to contain an even number of quad words, and to be aligned to a quad word boundary. The cursor buffer must also be aligned to a quad word boundary.

C

Using the ASTCR register at high MEMCLK frequencies

This appendix describes the use of the ASTCR register.

Using the ASTCR register at high MEMCLK

C.1 Introduction

Whenever the ARM processor performs a memory cycle it is clocked by **MCLK** which is derived from **MEMCLK**. The I/O controller inside ARM7500 is clocked by derivatives of **I_OCLK**. Thus, when the ARM processor performs a read from or a write to an area of I/O space, some synchronisation must occur. The ARM7500 bus controller decodes the address of the ARM processor access and if it recognises it as an I/O access must send an I/O cycle request signal to the I/O controller. This is synchronised to the internal I/O clock, IOCK32. The I/O controller then performs the necessary cycle asserting one (or more) of the I/O chip select signals, eg. nCCS. When the I/O controller knows the I/O cycle is about to finish it asserts an I/O grant signal which is synchronised back to the internal memory clock, **MEMRFCK**. The Bus controller will then terminate the cycle by creating a falling edge on **MCLK** which clocks the ARM processor.

The address from the ARM processor is latched when **MCLK** is LOW so that it is held stable throughout I/O cycles (as well as ROM). It is therefore important that **MCLK** should not fall too quickly after the end of the I/O chip select, else the address may change too quickly violating the required hold time. ARM7500 has been designed to support **MEMCLK** running at a frequency much higher than **I_OCLK**. In this situation the I/O grant generated by the I/O controller will be synchronised more quickly back to **MEMRFCK** and so the address will change sooner after the end of the I/O chip select. Thus the I/O controller must delay the point at which it generates the I/O grant to ensure the address hold time is maintained.

A patented technique using the ASTCR register bit, 0x0320000CC, has been employed to allow the address hold time to be maintained when **MEMCLK** frequency is greater than **I_OCLK** frequency whilst not imposing greater than necessary wait states when **MEMCLK** has the same or lower frequency than **I_OCLK**.

For a given system the **I_OCLK** frequency should be fixed at 32MHz, while **MEMCLK** frequency will be fixed according to the speed grade of DRAMs being used. The amount of hold time required between the end of the I/O chip select and the latched address changing should be determined and then ASTCR should be set dependent on the following details.

When ASTCR is LOW (reset value):

I/O cycle type	*Minimum Hold time*
Simple I/O	2 **MEMCLK** periods minus 1 **I_OCLK** period
Module I/O	2 **MEMCLK** periods minus 1.5 **I_OCLK** periods
PC style I/O	2 **MEMCLK** periods minus 1.5 **I_OCLK** periods

When ASTCR is HIGH:

I/O cycle type	*Minimum Hold time*
Simple I/O	2 **MEMCLK** periods minus 0.5 **I_OCLK** periods
Module I/O	2 **MEMCLK** periods minus 0.5 **I_OCLK** periods
PC style I/O	2 **MEMCLK** periods minus 0.5 **I_OCLK** periods

NOTE: This assumes divide by 1 mode for the clocks, **MEMCLK** and **I_OCLK**.

For example, in a system with **I_OCLK**=32MHz and **MEMCLK**=40MHz, the minimum hold time for a PC-style access will be 3.125ns if ASTCR=0 and 34.375ns if ASTCR=1. In addition there will be a small amount of extra hold time due to the delay from the internal memory clock **RCLK** to the latch enable signal for the address. It should be further noted that these times refer to the signals changes at the pad on the inside of ARM7500. The relative capacitive loading of the latched address and I/O chip select will determine the overall timing.

D Expanding PC-style I/O to 32 bit

This appendix describes the extension of PC-style I/O to 32 bit.

Pre-release Draft - Unrestricted

Expanding PC-style I/O to 32 bit

D.1 Introduction

ARM7500 provides 16-bit I/O accesses as standard using the BD[15:0] port for all I/O types. The PC style I/O accesses, however, can be extended to allow full 32-bit accesses without any loss in access speed by the addition of external 16-bit transceivers. ARM7500 provides all the control signals necessary to support these external devices.

During PC style I/O write cycles the I/O controller routes the lower 16-bit halfword from the ARM processor's data bus onto BD[15:0] and drives the upper 16-bit halfword onto D[31:16]. During read cycles, the ARM processor's data bus is driven from two sources; the lower halfword from the data latched from BD[15:0]; and the upper halfword from D[31:16]. If the external devices to provide the upper halfword of data are not present or the I/O peripheral does not support more than 16-bits then the software must ignore the upper halfword read back into the ARM processor registers.

➲*Figure D-1: 32-bit I/O interface* shows an example of the system connections required to provide a full 32-bit I/O interface. The write and read path should each contain a 16-bit latch with tri-state output enable control. The write latch should latch data from D[31:16] when **nBLO** is HIGH and drive the latched data onto the expanded I/O bus, BDHI[15:0], when **nWBE** is active LOW. The read latch should latch data from BDHI[15:0] when **nBLO** is HIGH and drive the latched data onto D[31:16], when **nRBE** is active LOW.

Note that, like the BD[15:0] bus, the write enable **nWBE** remains active LOW by default. It is de-asserted only during the read cycles, thus the I/O device must not attempt to drive BD[15:0] or BDHI[15:0] except when a read cycle is taking place.

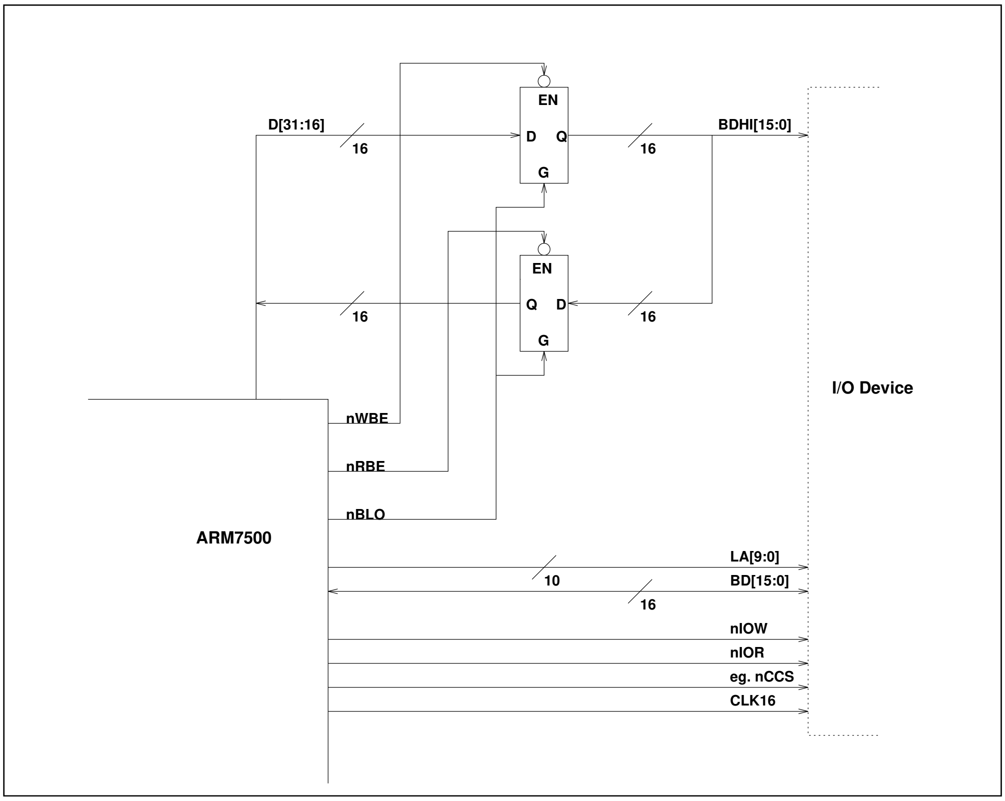


Figure D-1: 32-bit I/O interface

E ARM7500 Video Clock sources

This appendix describes the ARM7500 video clock sources.

E.1 Introduction

In order to facilitate the high resolution screen modes that ARM7500 is capable of producing, a suitable high frequency clock must be applied. As screen mode is changed, so the pixel rate must also change. This can be done via the various clock inputs, by the on-chip *pre-scaler*, or by using an external *voltage controlled oscillator* in conjunction with the on-chip phase comparator, to form a phase-locked-loop (PLL).

It is intended that most systems be built with a phase-locked-loop system. The required circuitry is simple, and allows a high degree of flexibility. The advantages are that all the necessary clock frequencies can be derived from the one circuit, and so the requirement for multiple on board crystals and clock switching circuitry is eliminated.

E.2 Clock sources

ARM7500 has 3 primary inputs for its pixel clock. These are **HCLK**, **VCLKI**, and the internal i_oclk32 signal, derived from **I**_OCLK. The intention is that **VCLKI** and the internal i_oclk32 signal, derived from **I**_OCLK be used to drive the phase comparator, and that **HCLK** would only be used to provide the highest frequency clock if this frequency is above the maximum VCO frequency.

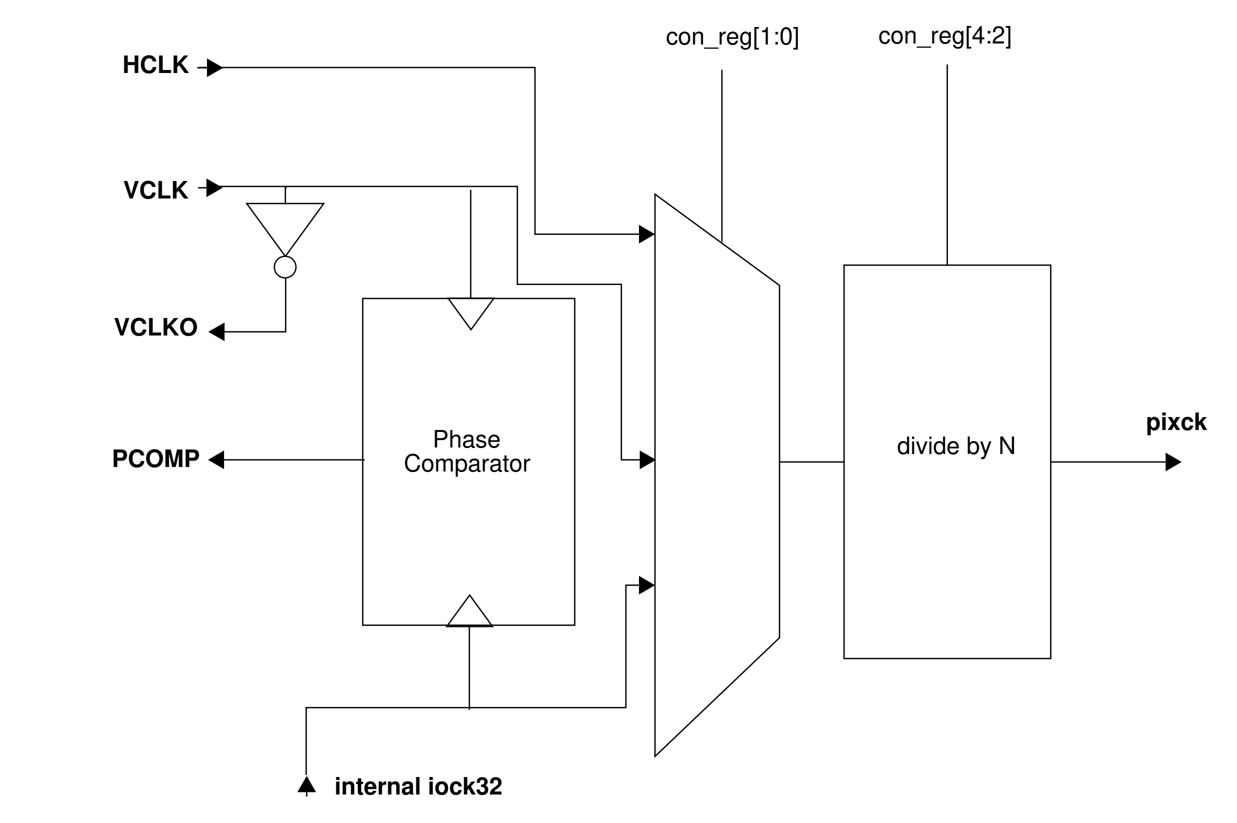


Figure E-1: Video and sound macrocell internal clock system

In addition to the pixel clock inputs, there is one other clock input, **SCLK**.

The sound system can be clocked from the internal 32MHz ioclk32 or a 16MHz **SCLK** (there is a divide-by-2 in the sound system). The digital sound system may run at a different frequency, (low MHz range), and this must be applied directly on **SCLK**. The pixel clock source is selected by programming bits 0 and 1 of the control register. The pixel clock selected can then be passed through a pre-scaler which can divide the clock by between 1 and 8. This is done by programming bits 2 to 4 of the control register. See *9.27 Control register (conreg): Address 0xE* on page 9-15.

Note that any unused clock pin should be tied low.

E.3 Using the phase comparator

The Video and sound macrocell contains a phase comparator which, in conjunction with an external voltage controlled oscillator (VCO), can be used to build a phase-locked-loop. The phase comparator comprises two counters and a phase detector. The counters are pre-loadable down counters, one clocked from the internal i_oclk32 signal, derived from I_OCLK, and the other clocked from **VCLKI**. The moduli of the counters is programmed in the Frequency Synthesiser Register.

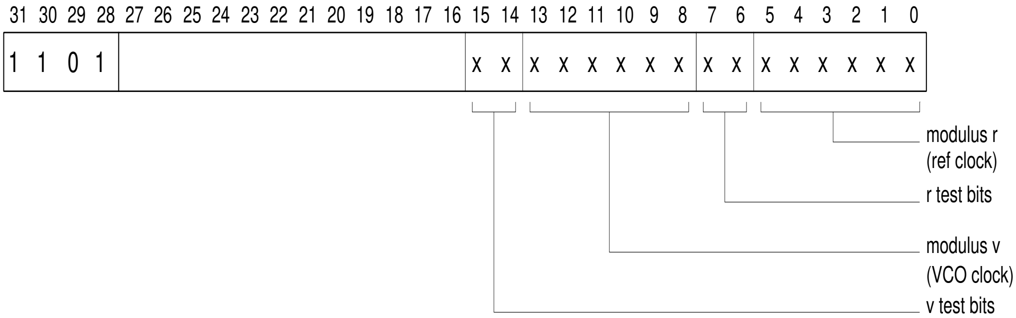


Figure E-2: Frequency Synthesizer Register

In this register, the test bits have the following meaning:

bit [6]	force **PCOMP** high and driven
bit [7]	clear r-modulus counter
bit [14]	force **PCOMP** low and driven
bit [15]	clear v-modulus counter

These bits are only programmed during test and at reset (see section *E.4 Phase Comparator Reset*). The internal i_oclk32 signal, derived from I_OCLK input provides a reference clock which is recommended to be 24MHz. The **VCLKI** input is driven from the output of the VCO, and it is this which is selected as the pixel clock.

The VCO is driven by ARM7500's **PCOMP** output, which for most of the time is at the tri-state value. When the VCO's frequency needs to be increased, **PCOMP** goes high, and vice-versa for when the frequency needs to be decreased. The **PCOMP** output needs to be filtered before applying to the VCO.

The choice of filter and VCO are left to the user. A very simple and effective system can be built using an 74AC04 inverter pack, and a very simple LC filter. The filtered VCO output controls the operating voltage of the 74AC04 device. This system is shown in *Figure E-3: Suggested VCO/PLL circuit*, and gives an enormous range of frequencies (LF to hundreds of MHz). Since the output of this VCO is AC coupled, **VCLKI** needs to be biased at the mid voltage point. This is done by connecting a large resistor between **VCLKI** and **VCLKO** (**VCLKO** is the inversion of **VCLKI**).

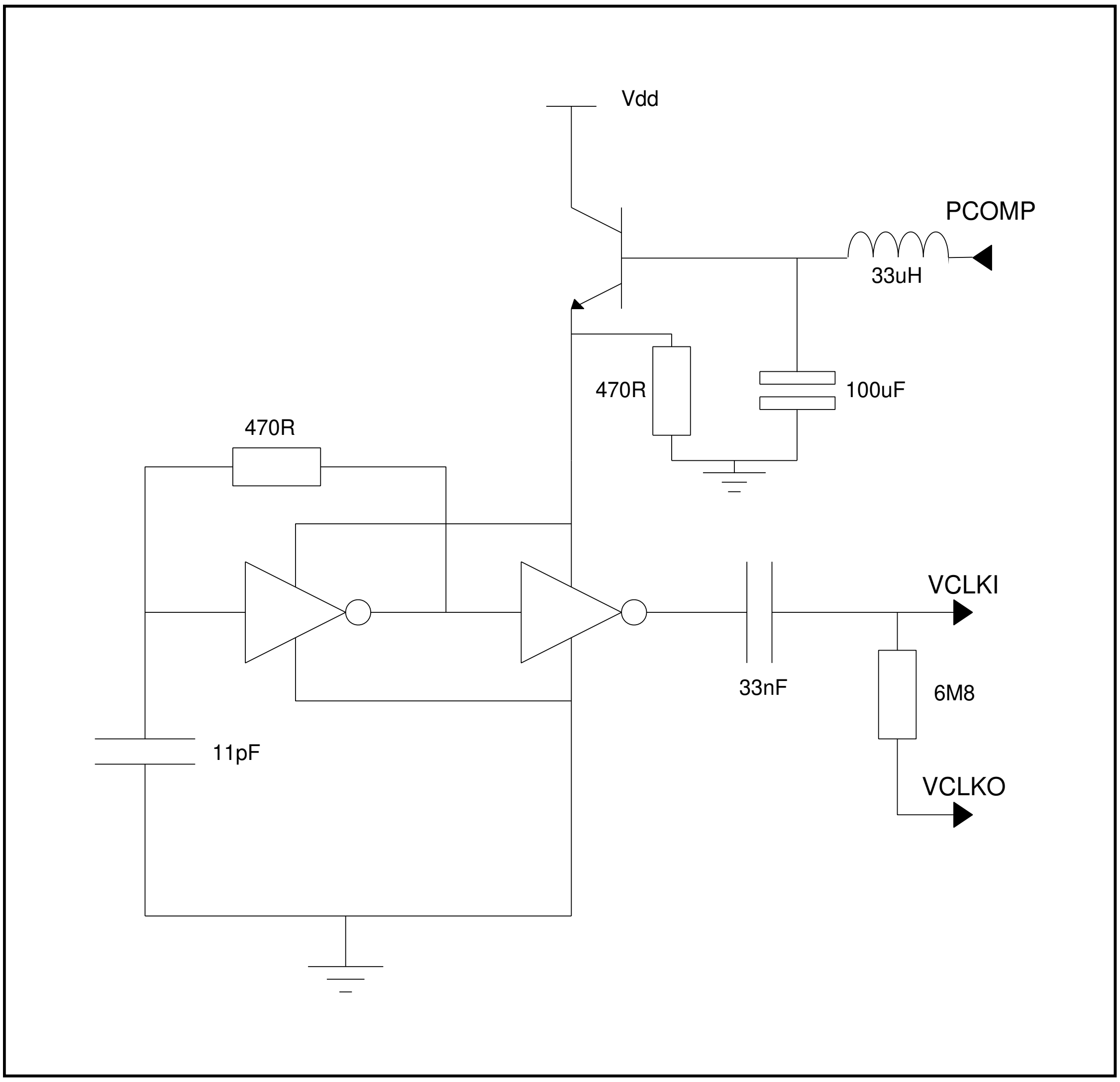


Figure E-3: Suggested VCO/PLL circuit

The actual frequency of the VCO is determined by the ratio of the v-modulus to the r-modulus as follows.

$$F_{VCO} = F_{REF} \times \frac{Vmodulus}{Rmodulus}$$

(Note that for a modulus of r, r-1 is programmed, and likewise for the v modulus.)

Table E-1: Synthesised VCO frequency settings gives a list of useful frequencies with corresponding values of r and v moduli, assuming a reference frequency of 32MHz. Obviously there are many values of r and v which give the same ratio. The lower the values, the more frequently the output of the VCO will be updated and so the r and v values should be chosen to suit the response of the filter.

r-modulus	v-modulus	VCO frequency/ MHz
8	2	8.0
16	6	12.0
4	2	16.0
8	6	24.0
2	2	32.0
8	9	36.0
16	35	70.0
4	15	120.0

Table E-1: Synthesised VCO frequency settings

E.4 Phase Comparator Reset

The phase comparator and VCO form a closed loop feedback system which has potential to become unstable. If the system powers up in the state where the **PCOMP** output is trying to drive the VCO's output higher and higher, then very quickly it will reach a frequency which the phase comparator cannot resolve and thus recovery is impossible.

To avoid this, the following reset procedure must be applied carefully. The test bits in the Frequency Synthesiser Register can be used to force the phase comparator's output either high or low. Thus, soon after power up, this register must be programmed with bits 15, 14 and 7 high, and bit 6 low. The r and v moduli can have anything programmed into them, but r must be greater than v. This operation forces the VCO's frequency to decrease.

When the real pixel rate is to be programmed, it should be done in two steps:

1 The values of the r and v moduli should be programmed, but the test bits left

in the initialisation state.

2 All the test bits should be cleared.

The VCO will then ramp up to its operating frequency. Subsequently, a change of frequency can be achieved simply by reprogramming the r and v moduli.

Index

K

L

M

O

P

R

S

T

U

V

W