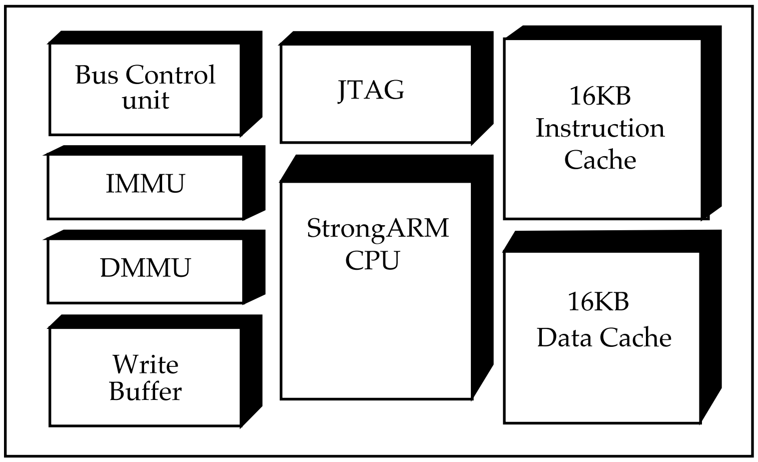
StrongARM RISC Processor

StrongARM is a general purpose 32-bit microprocessor with 16K byte instruction and a 16K byte data caches, write buffer and Memory Management Unit (MMU) combined in a single chip. The StrongARM processor offers high level RISC performance yet its design ensures minimal power consumption - making it ideal for portable, low cost systems.

The innovative 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. This results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective chip.



- **High performance RISC 161MHz@1.65V and 200MHz@2.0V**
- **Memory Management Unit (MMU) support for virtual memory systems**
- **16KB of instruction and 16KB writeback data cache**
- **Write Buffer - enhancing performance**
- **Static operation - low power consumption ideal for power sensitive applications**
- **Fast sub microsecond interrupt response for real-time applications**
- **Excellent high-level language support**
- **Big and Little Endian operating modes**
- **IEEE 1149.1 Boundary Scan**
- **144 Thin Quad Flat Pack (TQFP) package**

Applications:

- Personal computer devices e.g.PDAs
- High performance real time control systems
- Portable telecommunications
- Data communications equipment
- Consumer products
- Settops

StrongARM Data Sheet V2.0

Change Log:

Issue	Date	By	Change
A	22-Dec 94	RTW	Pass 0.0
B	2-Jan-95	RTW	Pass 0.1
C	3-Mar-95	RTW	Pass 0.2
D	15-May-95	RTW	Pass 1.0
E	19-Jan-95	RTW	Pass 2.0

Document No: ?

Issued: 19-Jan-1996

Table of Contents

1.0 Introduction

StrongARM is a general purpose 32-bit microprocessor with a 16KByte instruction cache, a 16KByte writeback data cache, an eight entry write buffer with 16 bytes per entry and a Memory Management Unit (MMU) combined in a single chip. The StrongARM is software compatible with the ARM V4 architecture processor family and can be used with ARM support chips, e.g. IO, memory and video.

The on-chip caches 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 external memory to support additional processors or Direct Memory Access (DMA) channels with minimal performance loss.

The instruction set comprises eight 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 16 logical (31 physical registers), each 32 bits wide;
- Three classes of instructions 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 and
- One class is used to access the privileged state of the machine.

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.

The memory interface has been designed to allow the performance potential to be realized 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 exploitation of page mode access offered by industry standard DRAMs.

StrongARM is a static part and has been designed to run at a reduced voltage to minimize its power requirements. This makes it ideal for portable applications where both these features are essential.

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.

1.1 ARM Architecture

StrongARM implements the ARM V4 architecture as defined in "The ARM Architecture Reference" 28-July-1995 , with the following options.

1.1.1 26 Bit mode

StrongARM supports 26 Bit mode but all exceptions are initiated in 32 bit mode. The P and D bits do not affect the operation of StrongARM; they always read as a one and writes to them are ignored.

1.1.2 Coprocessors

StrongARM only supports MCR and MRC access to coprocessor number 15. These instructions are used to access the memory management, configuration, and cache control registers. All other coprocessor instructions cause an undefined instruction exception. No support for external coprocessors is provided.

1.1.3 Memory Management

Memory management exceptions preserve the base address registers so no *fixup* code is required. Separate translation look aside buffers (TLBs) are implemented for the instruction and data streams. The TLBs each have 32 entries that can each map a segment, a large page, or a small page. The TLB replacement algorithm is round robin. The Data TBs support both the flush-all and flush-single-entry operation, while the instruction TLBs only support the flush all operation.

1.1.4 Instruction Cache

StrongARM has a 16Kbyte instruction cache (Icache) with 32 byte blocks and 32-way associativity. The cache supports the flush-all-entry function. Replacement is round robin within a set. The instruction cache can be enabled while memory management is disabled. When memory management is disabled all of memory is considered cacheable by the Icache.

1.1.5 Data cache

StrongARM has a 16Kbyte data cache (Dcache) with 32 byte blocks and 32-way associativity. The cache supports the flush-all, flush-entry, and copyback-entry functions. The copyback all function is not supported in hardware. This function can be provided by software. The cache is read allocate with round-robin replacement.

1.1.6 Write Buffer

StrongARM has a 8 entry write buffer with each entry able to contain 1 to 16 bytes. The drain-writebuffer operation is supported.

1.2 Block Diagram

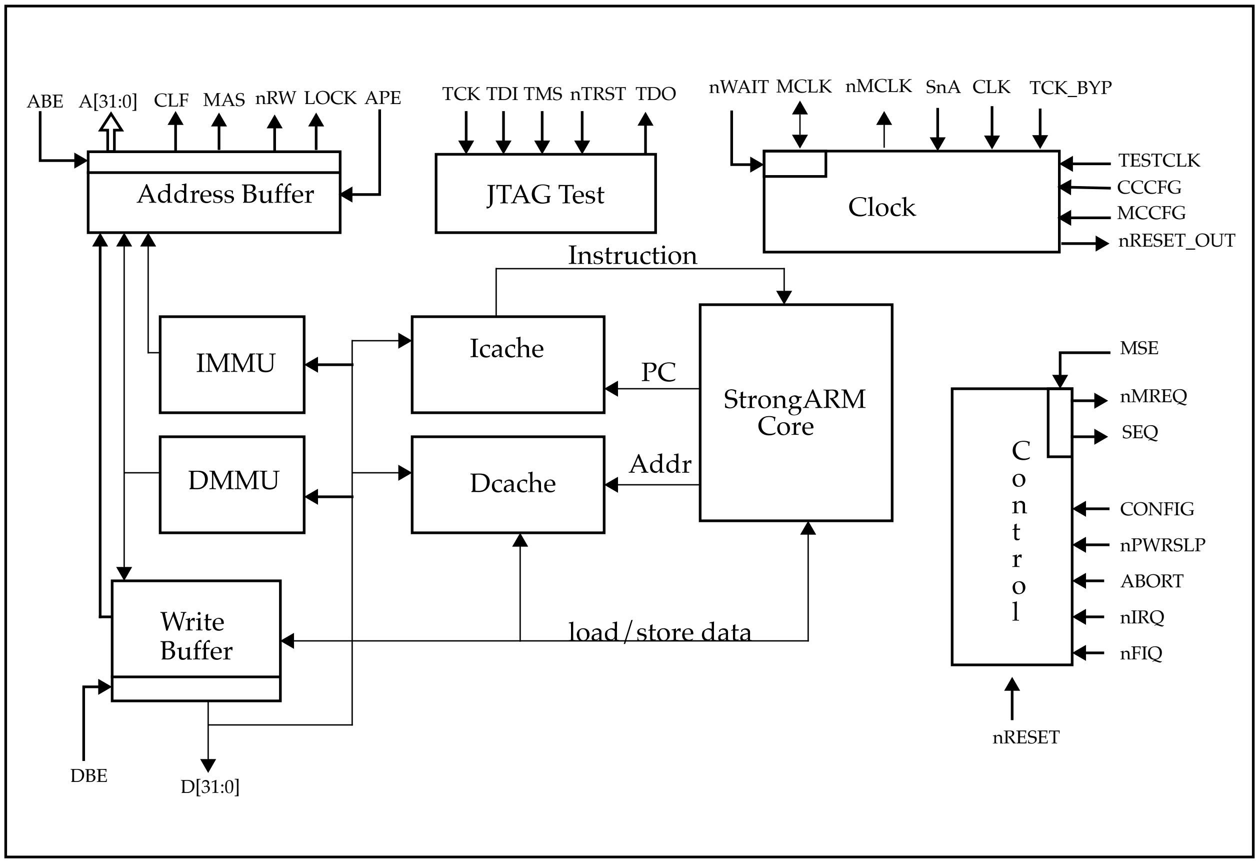


Figure 1: StrongARM Block Diagram

1.3 Functional Diagram

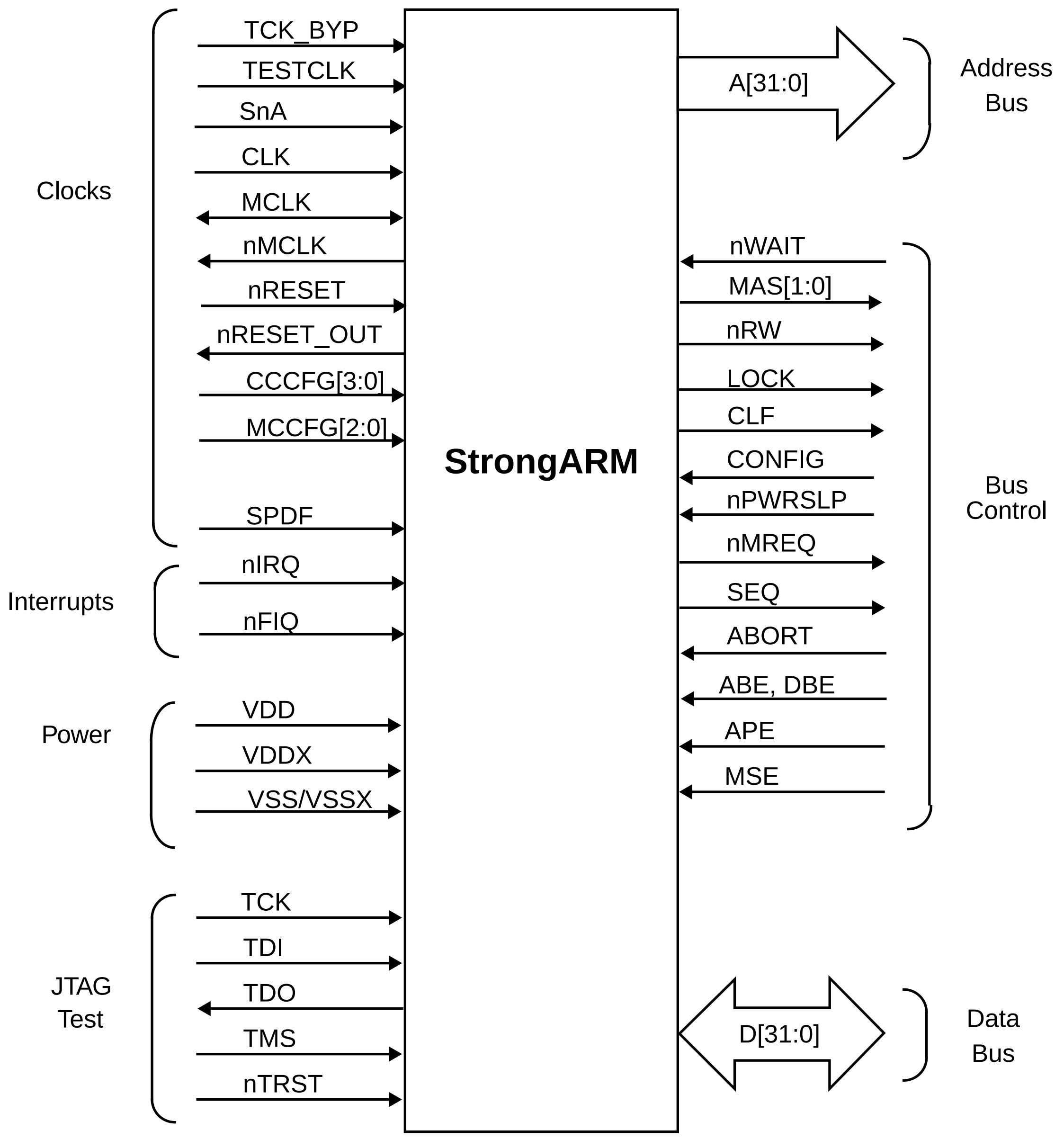


Figure 2: Functional Diagram

2.0 Signal Description

Name	Type	Description
A[31:0]	OCZ	Address Bus. This bus signals the address requested for memory accesses. If APE is HIGH the address changes during MCLK HIGH, if APE is LOW the address changes during the following MCLK LOW period. In enhanced bus mode A[1:0] contain byte mask bits two and three. The bytemask bits are asserted LOW to signal which bytes of the 32 bit data bus contain data to be accessed. In standard mode A[1:0] contain the low order two bits of the address. In both modes A[31:2] contain the upper 30 bits of the byte address.
ABE	IC	Address bus enable. When this input is LOW, the address bus A[31:0], nRW, MAS[1:0], CLF, and LOCK are put into a high impedance state (Note 1).
ABORT	IC	External abort. Allows the memory system to tell the processor that a requested access has failed. Only monitored when StrongARM is accessing external memory.
APE	IC	Address pipeline enable. This input is used to control the timing of the latches on the address bus A[31:0], MAS[1:0], nRW, CLF, and LOCK. Normally these signals change during MCLK HIGH, but they may be held to the next MCLK LOW by driving APE LOW. APE is a static configuration pin and must remain stable at all times.
CCCFG[3:0]	IC	Core Clock Configuration pins. These four pins configure the core clock speed. They must remain stable at all times.
CLF	OCZ	Cache Line Fill. An output signal used by the processor to indicate a cache line fill burst read or write. Cache line fills are 8 words long. If CONFIG is HIGH then cache line fetches are wrapped in the four word subblock then switch to the other subblock. The CLF signal follows address bus timing. Note it is NOT asserted for writes in pass one parts.
CLK	IC	The 3.68 MHZ input clock for the core PLL. This is the base clock in from which the high speed core clock and the MCLK in synchronous mode are generated.
CONFIG	IC	The CONFIG input sets the bus mode to standard (LOW) or enhanced (HIGH). In standard mode all cache line fetches start at word 0 of the cache block and stores are not allowed to merge. If CONFIG is HIGH then cache line fetches are wrapped in the four word subblock then switch to the other subblock starting at the critical word. The write buffer is allowed to merge stores to the same cache block and will use the A[1:0] and MAS[1:0] pins to present a byte mask of the bytes being read or written.The CONFIG signal must remain stable at all times.
D[31:0]	ICOCZ	Data bus. These are bi-directional pins used for data transfers between the processor and external memory. For read operations (when nRW is LOW), the input data must be valid before the falling edge of MCLK. For write operations (when nRW is HIGH), the output data will become valid while MCLK is LOW.
DBE	IC	Data bus enable. When this input is LOW, the data bus, D[31:0] is put into a high impedance state (Note 1). The drivers will always be high impedance except during write operations, and DBE must be driven HIGH in systems which do not require the data bus for DMA or similar activities.

Name	Type	Description
LOCK	OCZ	Locked operation. LOCK is driven HIGH, to signal a “locked” memory access sequence, and the memory manager should wait until LOCK goes LOW before allowing another device to access the memory. LOCK changes while MCLK is HIGH and remains HIGH during the locked memory sequence. The LOCK signal follows address bus timing.
MAS[1:0]	OCZ	Memory access size. An output signal used by the processor in standard bus mode to indicate to the external memory system the number of bytes being transferred. MAS[1:0] is 2 for word transfers, 1 for halfword transfers, and 0 for byte transfers, and is valid for both read and write operations. In enhanced mode the MAS pins contain bits zero and one of the bytemask. The bytemask bits are asserted LOW to signal which bytes of the 32 bit data bus contain data to be accessed. The MAS signals follow address bus timing.
MCCFG[2:0]	IC	Memory Clock Configuration pins. These pins configure the system clock speed when SnA is asserted.They must remain stable at all times and may not be dynamically changed. When SnA is asserted MCLK is generated by dividing the core clock by the value of MCCFG plus two (2..9) for MCCFG in the range 0..7.
MCLK	ICOCZ	Memory clock input or output. When SnA is deasserted MCLK is an input clock, when SnA is asserted MCLK is an output. This clock times all StrongARM memory accesses. The LOW or HIGH period of MCLK may be stretched for slow peripherals; alternatively, the nWAIT input may be used with a free-running MCLK to achieve similar effects.
MSE	IC	Memory request/sequential enable. When this input is LOW, the nMREQ and SEQ outputs are put into a high impedance state (Note 1).
nFIQ	IC	Not fast interrupt request. If FIQs are enabled, the processor will respond to a LOW level on this input by taking the FIQ interrupt exception. This is an asynchronous, level-sensitive input, and must be held LOW until a suitable response is received from the processor.
nIRQ	IC	Not interrupt request. As nFIQ, but with lower priority. This is an asynchronous, level-sensitive input, and must be held LOW until a suitable response is received from the processor.
nMCLK	OCZ	Not Memory clock output. When SnA is HIGH, nMCLK is the inverse of MCLK. If SnA is LOW then nMCLK is held LOW. This output can also be disabled by an MCR instruction to save power if both nMCLK is not used.
nMREQ	OCZ	Not memory request. A pipelined signal that changes while MCLK is LOW to indicate whether or not, in the following cycle, the processor will be accessing external memory. When nMREQ is LOW, the processor will be accessing external memory.
nPWRSLP	IC	Power Sleep. When LOW this puts the StrongARM I/O pins into Sleep mode. In Sleep mode all outputs are driven LOW except nMREQ which is driven HIGH.
nRESET	IC	Not reset. This is a level sensitive input which is used to start the processor from a known address. A LOW level will cause the current instruction to terminate abnormally, and the on-chip caches, MMU, and write buffer to be disabled. When nRESET is driven HIGH, the processor will re-start from address 0. nRESET must remain LOW for at least 2 full MCLK cycles. While nRESET is LOW the processor will perform idle cycles.

Name	Type	Description
nRESET_OUT	OCZ	Not reset out. This signal is asserted when nRESET is asserted and deasserts when the processor has completed reseting.
nRW	OCZ	Not read/write. When HIGH this signal indicates a processor write operation; when LOW, a read. The nRW signal follows address bus timing.
nTRST	IC	Test interface reset. Note this pin does NOT have an internal pullup resistor. This pin must be pulsed or driven LOW to achieve normal device operation, in addition to the normal device reset (nRESET).
nWAIT	IC	Not wait. When LOW this allows extra MCLK cycles to be inserted in memory accesses. It must change during the LOW phase of the MCLK cycle to be extended. This is the only way to stall external memory cycles in synchronous mode.
SEQ	OCZ	Sequential address. This signal is the inverse of nMREQ, and is provided for compatibility with existing ARM memory systems. The signal changes while MCLK is HIGH.
SnA	IC	Synchronous/Not Asynchronous. When SnA is LOW, MCLK is an input and nMCLK is driven low. When SnA is HIGH, nMCLK and MCLK are output clocks with the frequency selected by the MCCFG and CCCFG pins. SnA must remain stable at all times and may not be dynamically changed.
SPDF	IC	SPDF. The SPDF pins should be tied low. It is for DIGITAL use only.
TCK	IC	Test interface reference Clock. This times all the transfers on the JTAG test interface.
TCK_BYP	IC	Test clock PLL bypass. When TCK_BYP is HIGH, the TESTCLK is used as the core clock in place of the PLL clock, when LOW the internal PLL output is used.
TDI	IC	Test interface data input. Note this pin does NOT have an internal pullup resistor.
TDO	OCZ	Test interface data output. Note this pin does NOT have an internal pullup resistor.
TESTCLK	IC	Test Clock. TESTCLK is used to provide the core clock when TCK_BYP is HIGH. It should be tied LOW if TCK_BYP is LOW.
TMS	IC	Test interface mode select. Note this pin does NOT have an internal pullup resistor.
VDD		Positive supply for the core. Eight pins are allocated to VDD.
VDDX		Positive supply for the I/O pins. Nine pins are allocated to VDDX.
VSS **VSSX**		Ground supply. Eight pins are allocated to VSS and nine pins are allocated to VSSX. The ground plane on the board should be the same for these pins.

Notes:

1. When output pads 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, as this can dissipate power, especially in the pads.
2. It must be noted that unless all inputs are driven to the VSS or VDDX, the input circuits will consume power.

Key to Signal Types:

IC - Input, CMOS threshold
ICOCZ - Input, CMOS threshold, output CMOS levels, tri-stateable
OCZ - Output, CMOS levels, tri-stateable

3.0 ARM Implementation Options

The following sections detail options in the ARM architecture as implemented by StrongARM.

3.1 Big vs Little Endian

The bigend bit, in the Control Register, sets whether the StrongARM 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.

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.

In the Big Endian scheme 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]**).

The state of bigend only changes the location of the bytes within a 32 bit word. The accessed bytes are changed for the load byte, store byte, load halfword, and store halfword instructions only. Instruction fetches and word load and stores are not changed by the state of the bigend bit.

3.2 Exceptions

Exceptions arise whenever there is a need for the normal flow of program execution to be broken, so that (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.

StrongARM 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; 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 later in this chapter. Most exceptions are fully defined in the ARM Architectural Reference. The following sections specify the exceptions where the StrongARM implementation differs from the ARM Architectural Reference.

StrongARM initiates all exceptions in 32 bit mode. When an exception occurs while running in 26 bit mode StrongARM saves only the PC in R14 and the CPSR in the SPSR of the exception mode. The 32bit handler will have to merge the condition codes, the interrupt enables, and the mode from the SPSR into R14 if a handler wants to run in 26 bit mode.

3.2.1 Reset

When the **nRESET** signal goes LOW, StrongARM stops executing instructions, asserts the nRESET_OUT pin, and then performs idle cycles on the bus.

When **nRESET** goes HIGH again, StrongARM does the following:

(1) Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The values of the saved PC and CPSR are not defined.

(2) Forces M[4:0]=10011 (32 bit Supervisor mode) and sets the I and F bits in the CPSR.

(3) Forces the PC to fetch the next instruction from address 0x0

At the end of the reset sequence, the MMU, Icache, Data Cache, and Write Buffer are disabled. Alignment faults are also disabled, and little-endian mode is enabled.

3.2.2 Abort

An ABORT can be signalled by either the internal Memory Management Unit or from the external **ABORT** input. ABORT 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 disk, and considerable processor activity may be required to recover the data before the access can be performed successfully. StrongARM checks for ABORT during memory access cycles. When aborted StrongARM will respond in one of two ways:

(1) 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.

(2) If the abort occurred during a data access (a *Data Abort*), the action depends on the instruction type.

(a) Single data transfer instructions (LDR, STR) will abort with no registers modified.

(b) The swap instruction (SWP) is aborted as though it had not executed, though externally the read access may take place.

(c) Block data transfer instructions (LDM, STM) abort on the first access that cannot complete. If write-back is set, the base is **NOT** updated. If the instruction would normally have overwritten the base with data (i.e. LDM with the base in the transfer list), the original value in the base register is restored.

When either a prefetch or data abort occurs, StrongARM 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).

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.

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.

3.2.3 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	not used	
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Table 1: Vector Summary

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

3.2.4 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 that 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), StrongARM 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.

3.2.5 Interrupt Latencies

Calculating the worst case interrupt latency for the StrongARM is quite complex due to the cache, MMU and write buffer and is dependent on the configuration of the whole system. Please see *Application Note - Calculating the StrongARM Interrupt Latency*.

3.3 Coprocessors

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

StrongARM uses the internal coprocessor designated #15 for control of the on chip MMU, Caches, and clocks. If a coprocessor other than #15 is accessed, then the CPU will take the undefined instruction trap. The coprocessor load, store, and data operation instructions also take the undefined instruction trap.

4.0 Instruction Set

4.1 Instruction Set

StrongARM implements the ARM V4 architecture as defined in "The ARM Architecture Reference Manual" version 0.20 dated 22-Dec-1994, with previously noted options.

4.2 Instruction Timings

The following table lists the instruction timing for StrongARM. The result delay is the number of cycles the next sequential instruction would stall if it used the result as an input. The Issue Cycles is the number of cycles this instruction takes to issue. For most instructions the result delay is zero and the Issue Cycles is one. For load and stores the timing is for cache hits.

Instruction group	Result delay	Issue Cycles
Data processing	0	1
Mul or Mul/Add giving 32 bit result	1..3	1
Mul or Mul/Add giving 64 bit result	1..3	2
Load Single - writeback of base	0	1
Load Single - load data zero extended	1	1
Load Single -load data sign extended	2	1
Store Single - writeback of base	0	1
Load Multiple (delay for last register)	1	MAX(2,Num of Regs loaded)
Store Multiple - writeback of base	0	MAX(2,Num of Regs loaded)
Branch or Branch and Link	0	1
MCR	2	1
MRC	1	1
MSR to control	0	3
MRS	0	1
Swap	2	2

Table 2: Instruction Timings

5.0 Configuration

The operation and configuration of StrongARM is controlled with coprocessor instructions, configuration pins, and with the Memory Management Page tables. The coprocessor instructions manipulate on-chip registers which control the configuration of the Cache, write buffer, MMU and a number of other configuration options.

Note: The gray areas in the register and translation diagrams are reserved and should be programmed 0 for future compatibility.

5.1 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.

31 28	27			24	23 21	20	19 16	15 12	11			8	7 5	4	3 0
Cond	**1**	**1**	**1**	**0**		**n**	**CRn**	**Rd**	**1**	**1**	**1**	**1**	**OPC_2**	**1**	**CRm**

Cond	-	ARM condition codes
CRn	-	StrongARM Register
CRm	-	Function bits for some MRC/MCR instructions
OPC_2	-	Function bits for some MRC/MCR instructions
Rd	-	ARM Register
n	-	1 MRC register read 0 MCR register write

Format of Internal Coprocessor Instructions MRC and MCR

5.2 Registers

StrongARM contains registers which control the cache and MMU operation. These registers are accessed using CPRT instructions to Coprocessor #15 with the processor in any privileged mode. Only some of registers 0-15 are valid: the result of an access to an invalid register is unpredictable.

Register	Register Reads	Register Writes
0	ID Register	Reserved
1	Control	Control
2	Translation Table Base	Translation Table Base
3	Domain Access Control	Domain Access Control
4	Reserved	Reserved
5	Fault Status	Fault Status
6	Fault Address	Fault Adress
7	Reserved	Cache Operations
8	Reserved	TLB Operations
9..14	Reserved	Reserved
15	Reserved	Test, clock, and Idle

Table 3: Cache & MMU control registers

5.2.1 Register 0 ID

Register 0 is a read-only register that returns the code for this chip: 0x4401A10x. The low order four bits of the register are the chip revision number.

31 24	23 16	15 4	3 0
44	**01**	**A10**	**Revision**

5.2.2 Register 1 Control

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

M Bit 0	**Enable/disable**
	0 - on-chip Memory Management Unit disabled

31	30																		12			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	I	0	0	R	S	B	1	1	1	W	C	A	M

1 - on-chip Memory Management Unit enabled

A Bit 1 **Address Fault Enable/Disable**
0 - alignment fault disabled
1 - alignment fault enabled

C Bit 2 **Data Cache Enable/Disable**
0 - Data cache disabled
1 - Data cache enabled

W Bit 3 **Write buffer Enable/Disable**
0 - Write buffer disabled
1 - Write buffer enabled

B Bit 7 **Big/Little Endian**
0 - Little-endian operation
1 - Big-endian operation

S Bit 8 **System**
This selects the access checks performed by the memory management unit.
See the ARM Architecture Reference for more information.

R Bit 9 **ROM**
This selects the access checks performed by the memory management unit.
See the ARM Architecture Reference for more information.

I Bit 12 **Instruction Cache Enable/Disable**
0 - Instruction cache disabled
1 - Instruction cache enabled

Bit 13..31 Unused
Undefined on Read. Writes ignored.

5.2.3 Register 2 Translation Table Base

Register 2 is a read/write register which holds the base of the currently active Level One page table. Bits [13:0] are undefined on read, ignored on write.

31 14	13 0
Translation Table Base	

5.2.4 Register 3 Domain Access Control

Register 3 is a read/write register which holds the current access control for domains 0 to 15.

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**

5.2.5 Register 4 Reserved

Register 4 is Reserved. Accessing this register has no effect.

5.2.6 Register 5

Read/Write: Fault Status
Reading register 5 returns the current contents of the Fault Status Register (FSR). The FSR is written when a data memory fault occurs or can be written by a MCR to the FSR. It is not updated for a prefetch fault. See *Chapter 7.0 Memory Management Unit (MMU)* for more details. Bits [31:9] are undefined on read, ignored on write. Bit 8 is ignored on write and is always returned as zero.

31 9	8	7 4	3 0
	0	**Domain**	**Status**

5.2.7 Register 6

Read/Write: Fault Address
Reading register 6 returns the current contents of the Fault Address Resgister (FAR). The FAR is written when a data memory fault occurs with the address of the data fault, or can be written by a MCR to the FAR.

31 0
Fault Virtual Address

5.2.8 Register 7 Cache Control Operations

Register 7 is a write-only register. The CRm and OPC_2 fields are used to encode the cache control operations. All other values for OPC_2 and CRm are unpredictable.

Function	OPC_2	CRm	Data
Flush I+D	0b000	0b0111	Ignored
Flush I	0b000	0b0101	Ignored
Flush D	0b000	0b0110	Ignored
Flush D single entry	0b001	0b0110	Virtual Address
Clean D cache entry	0b001	0b1010	Virtual Address
Clean and flush D cache entry	0b001	0b1110	Virtual Address
Drain Write Buffer	0b100	0b1010	Ignored

Table 4: Cache Control Operations

5.2.9 Register 8 TLB Operations

Register 8 is a write-only register. The CRm and OPC_2 fields are used to encode the following TLB flush operations. All other values for OPC_2 and CRm are unpredictable.

Function	OPC_2	CRm	Data
Flush I+D	0b000	0b0111	Ignored
Flush I	0b000	0b0101	Ignored
Flush D	0b000	0b0110	Ignored
Flush D single entry	0b001	0b0110	Virtual Address

Table 5: TLB control operations

5.2.10 Registers 9 -14 Reserved

Accessing any of these registers unpredictable.

5.2.11 Registers 15   Test, Clock and Idle control

Register 15 is a write-only register. The CRm and OPC_2 fields are used to encode the following control operations. All other values for OPC_2 and CRm are unpredictable.

Function	OPC_2	CRm
Enable odd word loading of Icache LFSR	0b001	0b0001
Enable even word loading of Icache LFSR	0b001	0b0010
Clear Icache LFSR	0b001	0b0100
Move LFSR to R14.Abort	0b001	0b1000
Enable clock switching	0b010	0b0001
Disable clock switching	0b010	0b0010
Disable nMCLK output	0b010	0b0100
Wait for interrupt	0b010	0b1000

Table 6: Test, Clock, and Idle controls

5.2.11.1 IC LFSR controls

The Icache Linear Feedback Shift Register (LFSR) controls OPC_2=1 are documented in the "StrongARM Icache Testing" application note.

5.2.11.2 Clock controls

The four OPC_2=2 functions are used to enable and disable DCLK switching, disabling the nMCLK output and disabling MCLK output and waiting for an interrupt. See chapter 8 -- StrongARM Clocking for information on their use.

6.0 Caches and Write Buffer

To reduce effective memory access time StrongARM has an Instruction cache, a data cache, and a write buffer. In general these are transparent to program execution. The following sections describe each of these and give all necessary programming information.

6.1 Instruction Cache (IC)

StrongARM contains a 16kByte instruction cache (IC). The IC has 512 lines of 32 bytes (8 words), arranged as a 32 way set associative cache, and uses the virtual addresses generated by the processor core. The IC is always reloaded a line at a time (8 words). It may be enabled or disabled via the StrongARM Control Register and is disabled on **nRESET**. The operation of the cache when memory management is enabled is further controlled the *Cacheable* or C bit stored in the Memory Management Page Table (see chapter *7.0 Memory Management Unit (MMU).*). If Memory Management is disabled all addresses are marked as cacheable (C=1). When Memory Management is enabled the C bit in each page table entry can disable caching for an area of virtual memory.

6.1.1 IC Operation

In the StrongARM the instruction cache will be searched regardless of the state of the C bit, only reads that miss the cache will be affected. If, on an IC miss, the C bit is a one or the MMU is disabled, a linefetch of 8 words will be performed and it will be placed in a cache bank with a round-robin replacement algorithm. If, on a miss, the MMU is enabled and the C bit is a zero for the given virtual address, an external memory access for a single word will be performed and the cache will not be written.The IC should be enabled as soon as possible after reset for best performance.

6.1.2 IDC validity

The IC 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 IC validity must be ensured. The IC is not coherent with stores to memory so programs that write cacheable instruction locations must ensure the IC validity. Instruction fetches do not check the writebuffer so data must not only be pushed out of the cache but the write buffer must also be drained.

6.1.2.1 Software IC Flush

The entire IC can be invalidated by writing to the StrongARM cache operations register (Register 7). The cache will be flushed immediately when the register is written, but note that the following four instruction fetches may come from the cache before the register is written.

6.1.3 IC Enable/Disable and Reset

The IC is automatically disabled and flushed on **nRESET**. Once enabled, cacheable read accesses will cause lines to be placed in the cache. If the IC is subsequently disabled, no new lines will be placed in the cache, but the cache will still be searched and if the data is found it will be used by the processor. If the data in the cache must not be used, then the cache must be flushed.

6.1.3.1 To enable the IC

To enable the IC set bit 12 in Control Register. The MMU and IC may be enabled simultaneously with a single control register write.

6.1.3.2 To disable the IC

To disable the IC clear bit 12 in Control Register.

6.2 Data Cache (DC)

StrongARM contains a 16KByte writeback data cache. The DC has 512 lines of 32 bytes (8 words), arranged as a 32 way set associative cache, and uses the virtual addresses generated by the processor. A line also contains the physical address the block was fetched from and two dirty bits. There is a dirty bit associated with both the first and second half of the block. When a store hits in the cache the dirty bit associated with it is set. When a block is evicted from the cache the dirty bits are used to decide if all, half, or none of the block will be written back to memory using the physical address stored with the block. The DC is always reloaded a line at a time (8 words). It may be enabled or disabled via the StrongARM Control Register and is disabled on **nRESET**. The operation of the cache is further controlled by the *Cacheable* or C bit and the *Bufferable* or B bit stored in the Memory Management Page Table (see chapter *7.0 Memory Management Unit (MMU)*.). For this reason, in order to use the DC, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register.

6.2.1 Cacheable Bit - C

The **Cacheable** bit determines whether data being read may be placed in the DC 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 StrongARM'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].

6.2.2 Bufferable Bit - B

The **bufferable** bit does not affect writes that hit in the DC. If a store hits in the DC the store is assumed to be bufferable. Writebacks of dirty lines are treated as bufferable writes. See section 6.3 Write Buffer for more information on the B bit.

6.2.3 DC Operation

In the StrongARM the cache will be searched regardless of the state of the C bit, only reads that miss the cache will check the C bit. The C bit controls loading the cache on a miss not checking the cache on an access.

6.2.3.1 Cacheable Reads C = 1

A linefetch of 8 words will be performed and it will be placed in a cache bank with a round-robin replacement algorithm.

6.2.3.2 Uncacheable Reads C = 0

An external memory access will be performed and the cache will not be written.

Note: Load Multiples to C=0 space do NOT perform a burst read. This may added to the chip in a later pass.

6.2.4 DC validity

The DC 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 DC validity must be ensured.

6.2.4.1 Software DC Flush

StrongARM supports the Flush, clean, and clean-and-flush operations on single entries of the DC by writes to the Cache Operations regsiters. The flush whole cache is also supported. Note that since this is a writeback cache, in order to not lose data, a flush whole must be preceeded by a sequence of loads to cause the cache to write back any dirty entries. The following code will cause the DC to flush all dirty entries.

```
;+
;Call:
;     R0  points to the start of a 16384 byte region of readable data used
;         only for this cache flushing routine. If this area is used for
;         by other code then 32K must be loaded and the flush MCR is not
;         not needed.
;     bl    writeBackDC
;Return:
;     R0, R1, R2 trashed
;     Data cache is clean
;-
writeBackDC
      add         r1, r0, #16384
l1
      ldr         r2, [r0], #32
      teq         r1, r0
      bne         l1
      mcr         p15, 0, r0, c7, c6, 0
      mov         pc, r14
```

6.2.4.2 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, doubly-mapped virtual addresses should be marked as uncacheable.

6.2.5 DC Enable/Disable and Reset

The DC is automatically disabled and flushed on **nRESET**. Once enabled, cacheable read accesses will cause lines to be placed in the cache. If subsequently disabled, no new lines will be placed in the cache, but it will still be searched and if the data is found it will be used by the processor. Write operations will continue to update the cache, thus maintaining consistency with the external memory. If the data in the cache must not be used, then the cache must be flushed.

6.2.5.1 To enable the DC

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

6.2.5.2 To disable the DC

To disable the DC clear bit 2 in Control Register.

6.3 Write Buffer (WB)

The StrongARM write buffer is provided to improve system performance. It can buffer up to eight blocks of data of 1 to 16 bytes, at independent addresses. It may be enabled or disabled via the W bit (bit 3) in the StrongARM Control Register and the buffer is disabled and and all entries are marked empty on reset. The operation of the write buffer is further controlled by the *Cacheable* or C bit and the *Bufferable* or B bit, which are 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. It is not possible to abort buffered writes externally; the abort pin will be ignored.

6.3.1 Bufferable bit

This bit controls whether a write operation may use the write buffer. Typically main memory will be bufferable and I/O space unbufferable.

6.3.2 Write Buffer Operation

When the CPU performs a store, the DC is first checked. If the Dcache hits on the store and the protection for the location and mode of the store allows the write then the write completes in the cache and the writebuffer is not used. If the location misses in the DC then the translation entry for that address is inspected and the state of the B and C bits determines which of the three following actions are performed. If the write buffer is disabled via the StrongARM Control Register, writes are treated as if the B bit is a zero.

6.3.2.1 Writes to a Bufferable and Cacheable location (B=1,C=1)

If the write buffer is enabled and the CONFIG pin is asserted and the processor performs a write to a bufferable and cacheable location, and the data is in the DC, then the data is written to the DC, and the DC line is marked dirty. If a write to a bufferable area misses in the data cache, the data is placed in the write buffer and the CPU continues execution. When placing the write data into the write buffer if the data is being written to the same 16 byte aligned area of the previous write then the current write is merged into the same entry in the write buffer. The write buffer performs the external write some time later. If a write is done and the write buffer is full then the processor is stalled until there is sufficient space in the buffer. If the CONFIG pin is not asserted then no merging is allowed.

6.3.2.2 Writes to Bufferable and nonCacheable location (B=1,C=0)

If the write buffer is enabled and the processor performs a write to a bufferable but not cacheable location, and misses in the data cache, the data is placed in the write buffer and the CPU continues execution. When placing the write data into the write buffer no merging is allowed and each write takes it own write buffer entry. The write buffer performs the external write some time later.

Note: the case of a data cache hit for a bufferable but not cacheable location is a system programming bug. In this case the data is written to the DC, and the DC line is marked dirty.

6.3.2.3 Unbufferable Writes (B=0)

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. This will require several external clock cycles.

6.3.3 To enable the Write Buffer

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

6.3.4 To disable the Write Buffer

To disable the write buffer, clear bit 3 in Control Register. Note that any writes already in the write buffer will complete normally, but a drain write buffer needs to be done to force all writes out to memory.
Note the write buffer will be used for copybacks from the DC even when it is disabled.

7.0 Memory Management Unit (MMU)

7.1 Overview

StrongARM implements the standard ARM memory management functions using two, 32 entry fully associative TBs. One is used for instruction accesses and the other for data accesses. On a TB miss the translation table hardware is invoked to retrieve the translation and access permission information. Once retrieved, if the entry maps to a valid Page or Section then the information is placed into the TB. The replacement algorithm in the TB is round robin. For an invalid page or section an abort is generated and the entry is not placed in the TB.

7.1.1 MMU registers

See section 5.2 for a description of the MMU CP15 registers supported by StrongARM.

7.2 MMU Faults and CPU Aborts

The MMU generates four faults:

Alignment Fault
Translation Fault
Domain Fault
Permission Fault

Alignment faults are generated by word loads or stores with the low order two address bits not zero, and by load or store halfwords with the low order address bit a one. Translation faults are generated by access to pageds maked invalid by the memory management page tables. Domain faults and permission faults are generated by accesses to memory disallowed by the current mode, domain, and page protection. See the ARM architecture reference for more information.

In addition, an external abort may be raised on external data accesses.

7.3 External Aborts

In addition to the MMU-generated aborts, StrongARM has an external abort pin which may be used to flag an error on an external memory access. However, not all accesses can be aborted in this way, so this pin must be used with great care. Writes to memory areas marked as bufferable ignore the external abort pin.

The following accesses may be aborted and restarted safely. If any of the following are aborted the external access will cease on the next cycle. In the case of a read-lock-write sequence in which the read aborts, the write will not happen.

Reads
Unbuffered writes
Level One descriptor fetch
Level Two descriptor fetch
Read-lock-write sequence

7.3.1 Cacheable reads (Linefetches)

A linefetch may be safely aborted on any word in the transfer. If an abort occurs during the linefetch then the cache will be purged, so it will not contain invalid data. It the abort happens before the word that was requested by the access is returned, the load will be aborted. It the abort happens after the word that was requested by the access is returned, the load will complete, and the fill will be aborted (but no exception will be generated).

7.3.2 Buffered writes

Buffered writes cannot be externally aborted. Therefore, the system should be configured such that it does not do buffered writes to areas of memory which are capable of flagging an external abort.

7.4 Interaction of the MMU, IC, DC and Write Buffer

The MMU, IC, DC, and WB may be enabled / disabled independently. The IC can be enabled with the MMU enabled or disabled. However, the DC and WB can only be enabled when the MMU is enabled. Because the write buffer is used to hold dirty copyback cached lines from the DC, it must be enabled along with the DC. Therefore only four of the eight combinations of the MMU, DC, and WB enables are valid. There are no hardware interlocks on these restrictions, so invalid combinations will cause undefined results.

MMU	DC	WB
off	off	off
on	off	off
on	off	on
on	on	on

Table 7: Valid MMU, DC & Write Buffer 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 three 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 Flat
Fetch Translated
```

To disable the MMU

(1) Disable the WB by clearing bit 3 in the Control Register.
(2) Disable the DC by clearing bit 2 in the Control Register.
(3) Disable the IC by clearing bit 12 in the Control Register.
(4) 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.

8.0 StrongARM Clocking

8.1 StrongARM operating modes

The StrongARM chip supports three operating modes, Normal, Idle, and Sleep, and can be switched between them to minimize power consumption. In Normal mode the chip executes instructions. In Idle, no instructions are executed but the internal Phase Locked Loop (PLL) continues to run. In Sleep, core power, VDD is switched off; the chip does not execute instructions and the PLL does not run. The I/O power, VDDX, is used to hold the all output pins that are enabled at zero except nMREQ and nMCLK which are held at a one.

8.2 StrongARM Clocking

The StrongARM chip receives a 3.68MHz clock (CLK) from a crystal-based clock generator on the CLK pin and uses an internal PLL to lock to the input clock and multiply the frequency by a variable multiplier to produce a high speed core clock (CCLK). The 3.68MHz oscillator and PLL run constantly in Normal and Idle modes. There are two clocking domains in StrongARM, the core logic domain clocked by DCLK and the bus interface domain clocked by MCLK. The core clock, DCLK, switches between being driven by the high speed core clock, CCLK and the bus clock, MCLK. CCLK is used except when StrongARM is waiting for fills to complete after a cache miss. At RESET clock switching is disabled and the DCLK is driven from MCLK . The Clock switching can also be disabled by writing to the CP15 register 15 with OPC_2=2 and CRm=2. Clock switching is enabled by writing to the CP15 register 15 with OPC_2=2 and CRm=1. Disabling switching only disables switching for DCLK, it does not force the DCLK to MCLK. To force DCLK to be driven by MCLK after disabling switching in is necessary to force an instruction or data cache miss.

StrongARM can be configured to either source or sink MCLK. If SnA is deasserted, MCLK is an input to StrongARM. If SnA is asserted the StrongARM drives the MCLK pin with clock generated by dividing CCLK by 2 to 9 as specified by the MCCFG pins. StrongARM synchronizes signals which cross between the CCLK and MCLK domains. The SnA pin should be programmed by connecting it to VDDX or VSS and should not change. With SnA asserted nMCLK, an inverted copy of MCLK is also provided, it is enabled at reset and can be disabled by writing to the CP15 register 15 with OPC_2=2 and CRm=4 to save power if it is not used. If SnA is deasserted the nMCLK pin is held LOW.

8.2.1 Switching to Idle

With SnA deasserted, Idle is entered by disabling clock switching and then doing a load from a noncacheable location (C=0). The external driver of MCLK should then stop MCLK to stop StrongARM from executing instructions. To resume normal operation MCLK is restarted and clock switching is enabled.

With SnA asserted, Idle is entered by disabling clock switching, then doing a load from a noncacheable location (C=0), and then doing a wait for interrupt instruction. StrongARM stops driving MCLK in the next low phase and holds MCLK low. Asserting either nFIQ or nIRQ will cause MCLK to start clocking again. To resume normal operation, reenable clock switching.

8.2.2 Switching to Sleep

To enter Sleep mode assert nRESET and nPWRSLP, then switch off the VDD power. To exit Sleep restore the VDD supply and deassert nPWRSLP then nRESET . Note there is no difference between initial reset and leaving Sleep mode.

8.2.3 Core Clock Configuration - CCCFG

The high speed core clock frequency is configured at reset by the four Core Clock Configuration pins (CCCFG[3:0)]). These pins should be programmed by connecting them to VDDX or VSS and should not change . The following table gives the CCLK frequency as a function of the CCCFG pins. The PLL generates core clock at +-5% of the nominal frequency.

CCCFG	CCLK Frequency MHZ
0	88.3
1	95.6
2	99.4
3	106.7
4	143.5
5	150.9
6	161.9
7	169.3
8	191.3
9	202.4
10	213.4
11	228.1
12	242.8
13	257.6
14	276.0
15	287.0

Table 8: CCLK configurations

8.2.4 Memory Clock Configuration - MCCFG

If SnA is asserted then the MCLK frequency is selected by the three Memory Clock Configuration pins (MCCFG[2:0]). These pins should be programmed by connecting them to VDDX or VSS and should not change. The following table gives the MCLK divisors as a function of the MCCFG pins. The MCLK is a 50% duty cycle clock. Note: Some combinations of MCCFG and CCCFG settings will produce an MCLK output that exceeds the maximum supported MCLK frequency.

Note: Pass one parts do not generate a 50% duty cycle MCLK for odd CCLK divisors. Pass one parts generate an integer number of CCLK times of of high and low. For example CCLK divisor of 3 generates an MCLK with with 1/3 time high and 2/3 time low.

MCCFG	CCLK divisor
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9

Table 9: MCLK configurations

8.2.5 Tester and Debug Clocks

If TCK_BYP is high then the PLL ouput is not used and the high speed core clock is supplied externally on the TESTCLK pin. This mode is for testing use only and not supported for standard operation.

9.0 Bus Interface

The following chapter describes the external bus interface for StrongARM.

9.1 Bus modes

The StrongARM bus has two modes, standard and enhanced. When in Enhanced mode data cache line fills are wrapped to provide the critical word first, and the write buffer allows more generalized merging. The CONFIG signal is asserted (=1) for enhanced mode and deasserted (=0) for standard mode. The **CONFIG** signal can only change while **nRESET** is asserted.

9.1.1 Standard and Enhanced mode

In enhanced mode line fills are done critical word first and the burst read is wrapped at the end of the four word cache subblock, then the other subblock is accessed in the same order. In standard mode, line fills are started on cache block word 0 so no wrap is needed.

Note: This matches current SSRAMS , burst mode EDO DRAMs., and SDRAMs.

Miss Address bits [4:2]	Fill Order
0	0,1,2,3,4,5,6,7
1	1,2,3,0,5,6,7,4
2	2,3,0,1,6,7,4,5
3	3,0,1,2,7,4,5,6
4	4,5,6,7,0,1,2,3
5	5,6,7,4,1,2,3,0
6	6,7,4,5,2,3,0,1
7	7,4,5,6,3,0,1,2

Table 10: Enhanced Mode Wrapping order

In enhanced mode, the write buffer is allowed to merge random byte, halfword, and word stores. On each bus cycle the word address is specified on the **A[31:2]** pins and the bytes to be read/written are specified by a byte mask. The byte mask bits [3:0], if one, specify that bytes [3:0] of the data bus are being transferred. The byte mask is presented on the **A[1:0]** and **MAS[1:0]** pins and is asserted LOW.

Byte Mask Pins	D bus Bytes
A[1]	D[31:24]
A[0]	D[23:16
MAS[1]	D[15:8]
MAS[0]	D[7:0]

Table 11: Byte Mask

In standard mode the write buffer will only merge stores from store multiple instructions and castouts. In standard mode a **MAS[1:0]** of 2 is for word, 1 is for halfword, 0 is for byte. The full byte address of the transfer is given by **A[31:0]**

Read data must always be returned on the correct byte of the data bus **D[31:0].** The chip provides write data on the data bus bytes specified by the byte mask in Enhanced mode or by the low order two address bits and the memory access size in Standard more. *Note: unlike previous ARM chips StrongARM does not replicate byte stores to all four bytes of the data bus.*

9.2 StrongARM Bus Stalls

The bus interface is controlled by **MCLK**, and all timing parameters are referenced with respect to this clock. The way the bus is stalled depends on the mode of the bus. If the system is providing MCLK, SnA=0, the bus is stalled in one of two ways.

1) The LOW or HIGH phases of the clock may be stretched

2) **nWAIT** can be used to insert entire **MCLK** cycles into the access. When LOW, this signal maintains the LOW phase of the cycle by gating out **MCLK**. **nWAIT** is asserted before the rising edge of **MCLK** and latch by StrongARM. The **ABORT** input is not sampled durning clock cycles in which **nWAIT** is stalling the bus.

If StrongARM is providing MCLK, SnA=1, only the nWAIT signal can be used to stall the bus with the same timing as with SnA=0.

9.3 Cycle Types

There are two cycle types performed by StrongArm. These are *idle* cycles and *sequential* cycles. Idle and sequential cycles are combined to perform memory accesses. The two cycle types are differentiated by nMREQ. The SEQ signal is the inverse of nMREQ and is provided for compatibility with earlier memory controllers. nMREQ and SEQ are pipelined, and so their value determines what type the following cycle will be. nMREQ and SEQ become valid during the LOW phase of the cycle before the one to which they refer. The following table shows the encoding used to generate the idle and sequential cycles.

nMREQ	CYCLE TYPE
0	SEQUENTIAL
1	IDLE

Table 12: Cycle Type Encodings

The address from StrongArm becomes valid during the HIGH phase of MCLK, but can be delayed using APE. It is also pipelined, and its value refers to the following memory access.

9.4 Memory Access

There are two types of memory access. These are *non-sequential* and *sequential* . The non-sequential cycles occur when a new memory access takes place. Sequential cycles occur when the cycle is of the same type as, and the address is 1 word (4 bytes) greater than, the previous access or during cache line fills. Cache line fill cycles occur in responce to a cache miss and the address sequence is given above. Cache line fills have the CLF HIGH and use the defined wrapping order but otherwise are the same as sequential cycles. CLF has the same timing as the address.

Non-sequential accesses consist of an idle cycle followed by a sequential cycle, sequential accesses consist simply of a sequential cycle. In the case of a non-sequential access, the address is valid throughout the idle cycle, allowing extra time for memory decoding.

9.5 Read/Write

Memory accesses may be read or write, differentiated by the signal **nRW**. This signal has the same timing as the address. In the case of a write, the StrongArm outputs data on the data bus during the memory cycle. They become valid during **MCLK** LOW, and are held until the end of the cycle. In the case of a read, the data is sampled at the end of the memory cycle. **nRW** may not change during a sequential or cache line fill access, so if a read from address A is followed immediately by a write to address (A+4), then the write to address (A+4) would be a non-sequential access.

9.6 Address Pipeline Enable (APE)

APE is used to control the timing of the address signals including **A[31:0], MAS[1:0**], **nRW, CLF**, and **LOCK**. Normally these signals change during MCLK HIGH, but they may be held to the next MCLK LOW if **APE** is LOW. **APE** is a static configuration pin and must remain stable at all times. The rest of this chapter assumes **APE** is HIGH. If **APE** is LOW the address will be held a MCLK phase longer and be driven an MCLK phase later.

9.7 Memory Access Types

The following timing diagrams show a one word read or write memory access, a two word sequential access, two - one word accesses back to back. Notice that an idle cycle followed by a sequential cycle distinguishes a non-sequential access. Note also these diagrams are for when **APE** is HIGH.

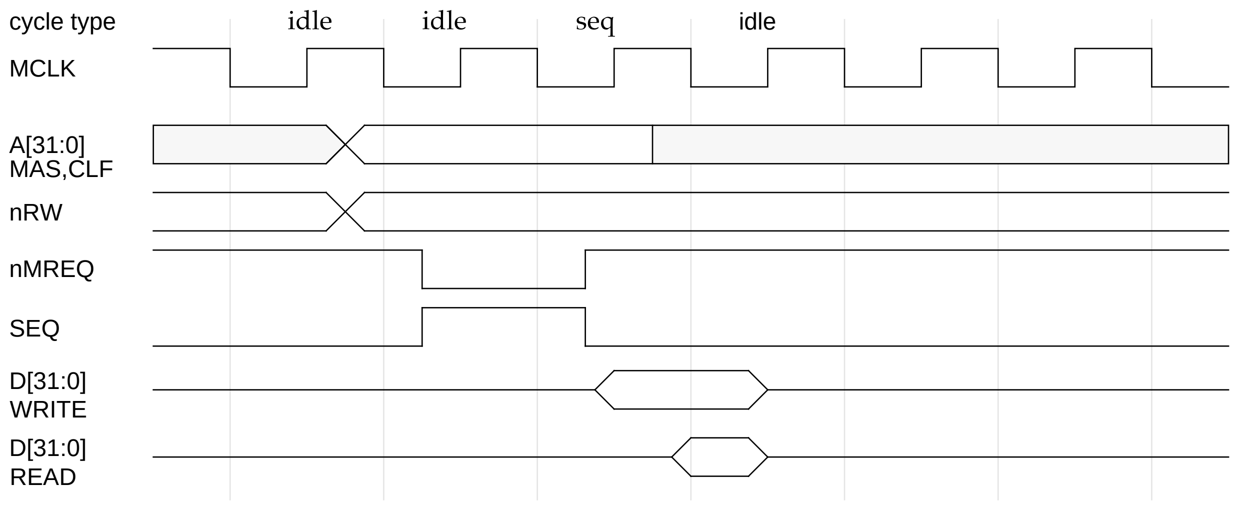


Figure 3: One word Read or Write

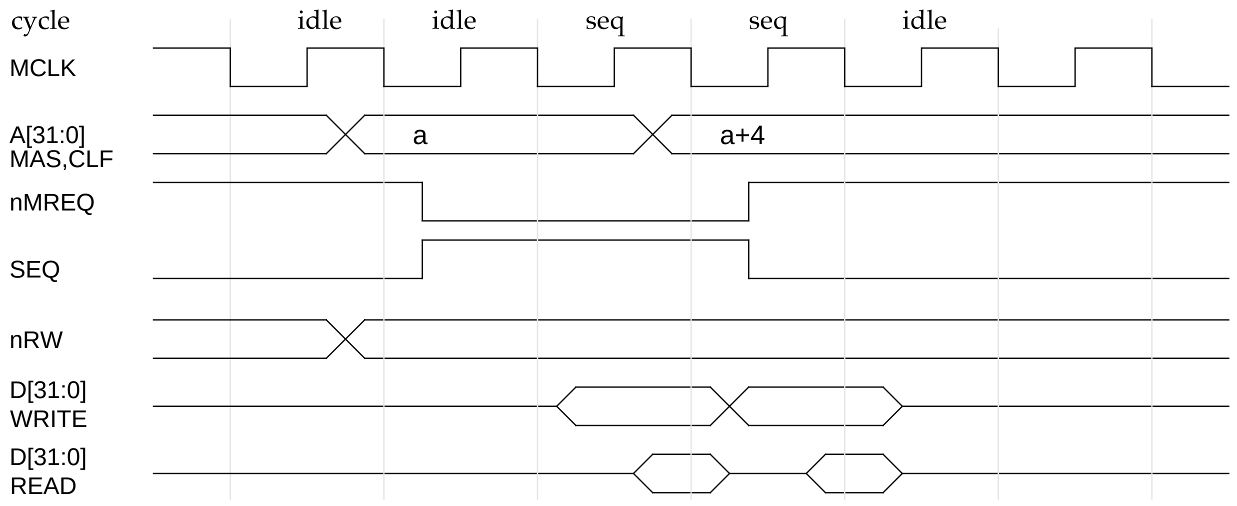


Figure 4: Two Word Sequential Read or Write

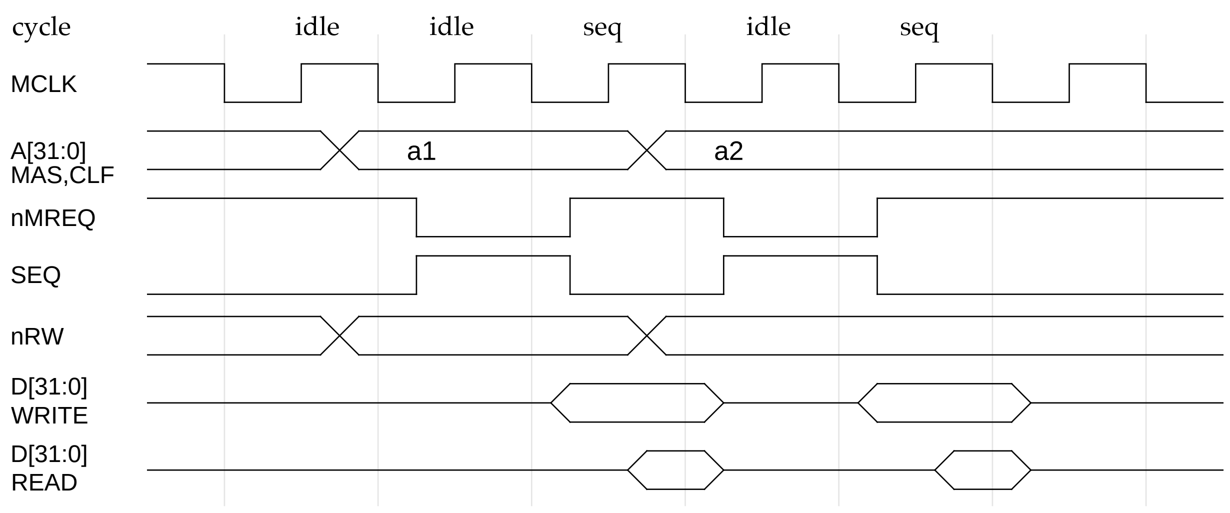


Figure 5: Two - One Word Non-Sequential Read or Writes back to back

9.8 External Accesses

StrongArm performs bus access for many different operations. All are constructed by combinations of non-sequential or sequential accesses. There may be any number of idle cycles between two other memory accesses. If a memory access is followed by an idle period on the bus (as opposed to another non-sequential access), then the address **A[31:0]**, and the signals **nRW** and **DL[1:0]** will remain at their previous value in order to avoid unnecessary bus transitions.

The accesses performed by a StrongARM are:

Unbuffered Write	Level 1 translation fetch
Uncached Read	Level 2 translation fetch
Buffered Write	Cache Line Copyback
Linefetch	Read-Lock-Write sequence

9.8.1 Unbuffered Writes / Uncacheable Reads

These are the most basic access types. Apart from the difference between read and write, they are the same. Each may consist of a single access. A multiple access consists of a non-sequential access followed by a sequential access. These cycles always reflect the type (i.e. read/write) of the instruction requesting the cycle. Level 1 and 2 translation fetches also generate uncacheable reads.

9.9 Buffered Write

The external bus cycle of a buffered write is identical to the bus cycle of an unbuffered write. Note, if in enhanced mode, that if several write accesses are stored concurrently within the write buffer, then each access on the bus will start with a non-sequential access. If several write accesses occur to the same cache subblock, StrongArm will merge these in the write buffer and write to memory using several sequential accesses with the byte masks indicating the valid byte lanes. Cache line copybacks are also buffered writes.

9.10 Linefetch

This access appears on the bus as a non-sequential access followed by sequential accesses. Linefetches are not required to start on an 8-word boundary and will be 8 words long. The first access will fetch the critical word if the bus is in enhanced mode or word 0 if in standard mode. CLF will be HIGH for line fetches. The following diagram is a cache line fill in enhanced mode. If it was in standard mode the fill would have started at location 0x00 rather than location 0x18.

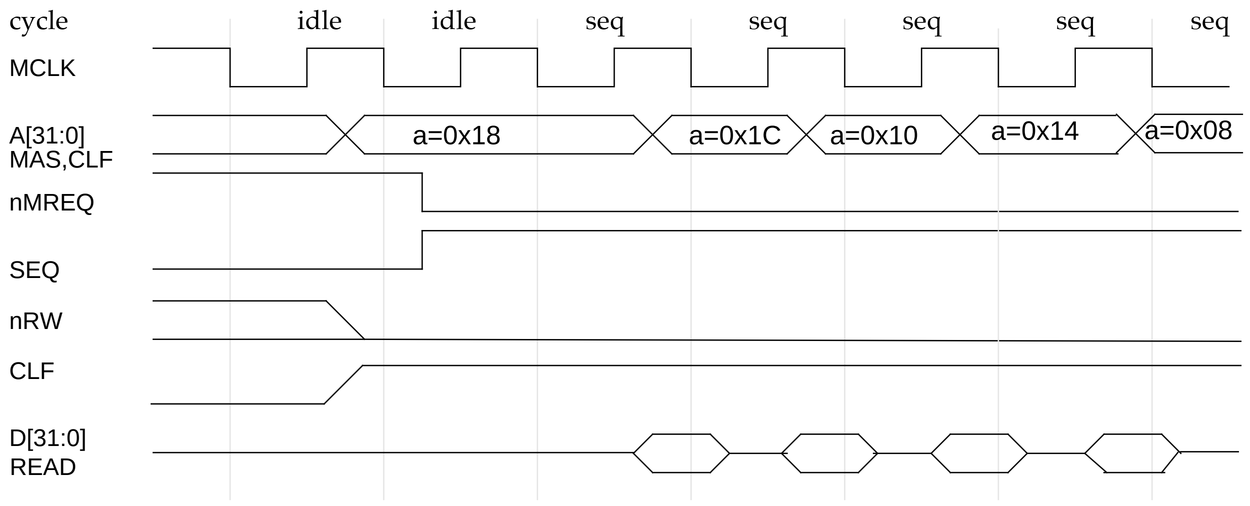


Figure 6: Linefetch

9.11 Read - lock -write

The read-lock-write sequence is generated by an SWP instruction to a non-cacheable/non-bufferable location. On the bus it consists of a single word read access followed by a single word write access to the same address, and both are treated as non-sequential accesses. The cycle is differentiated by the **LOCK** signal. **LOCK** has the timing of address, ie it goes HIGH in the HIGH phase of **MCLK** at the start of the read access. However, it always goes LOW at the end of the write access, even if the following cycle is an idle cycle (unless of course the following access was a read-lock-write sequence). There may be several idle cycles between the read and the write. Note that for a cacheable/bufferable access the two access are done in the data cache, preceeded by a standard line fill if needed.

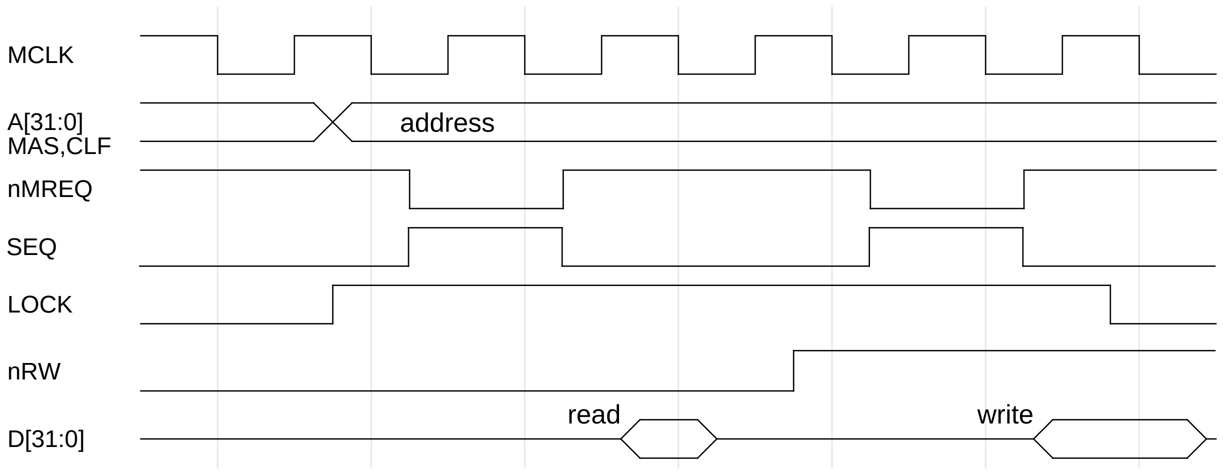


Figure 7: Read - Locked - Write

9.12 Stalling the Bus

StrongARM can be stalled by **nWAIT** as shown for a load in the following diagram. The diagram shows a load request that has been stalled for one cycle waiting for return data on the third word.

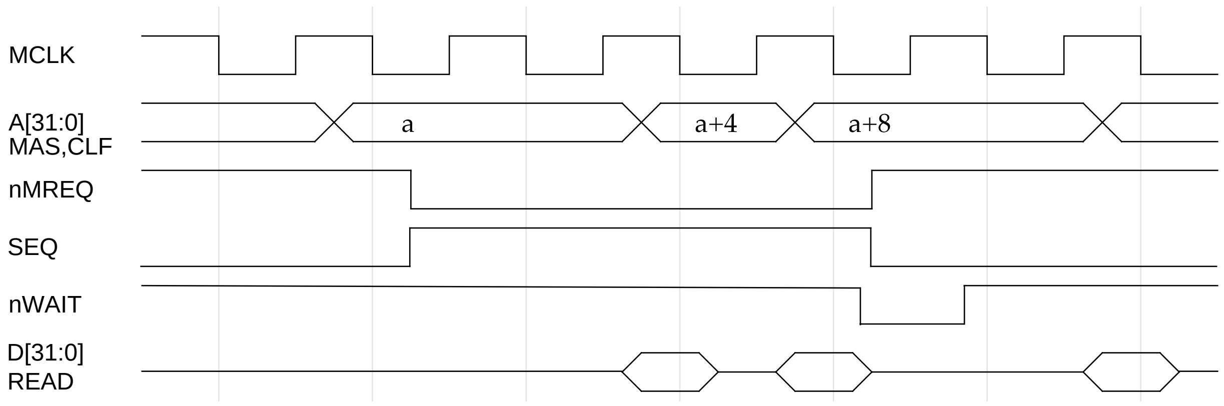


Figure 8: Use of nWAIT pin to stop StrongARM for 1 MCLK cycle

9.13 Summary of transactions

StrongARM will only generate a subset of all possible transactions on the bus. StrongARM will generate only single non-sequential transactions and the following bursts:

9.13.1 Read bursts

The only actions that cause read bursts are cache line fills. All other reads will be single non-sequential accesses. All cache line fills are 8 words long.

9.13.2 Write bursts

In Standard mode write bursts are caused by STM to B=1 locations and castouts of the data cache. Because the data cache keeps a dirty bit for each 16 byte subblock in standard mode write bursts are either 4 or 8 words long. In enhanced mode the write buffer will merge random writes. The write burst sizes in enhanced mode are 2, 3, 4, or 8 words. The **CLF** bit is asserted on stores for all 32 byte write bursts. *Note: Pass one parts never assert **CLF** on writes.*

9.13.3 Transaction Summary

The following table lists all the transactions that StrongARM can generate. No burst will cross an aligned 32 byte boundary.

Bus Operation	**CONFIG**	Burst size	Starting address bits [4:2]	note
Read single	0 or 1	1	any	
Read burst	0	8	0	Generated by cache line fills
Read burst	1	8	any	Generated by cache line fills
Write single	0	1	any	
Write single	1	1	any	1..4 bytes are written as specified by the byte mask
Write burst	0	2	0, 1, 2 4, 5, 6	All 8 bytes are written
Write burst	0	3	0, 1 4, 5	All 12 bytes are written

Table 13: StrongARM Transactions

Write burst	0	4	0 4	All 16 bytes are written
Write burst	1	2	0, 1, 2 4, 5, 6	Each word can have one to four bytes written
Write burst	1	3	0, 1 4, 5	The first and last word can have one to four bytes written. The middle word can have zero to four bytes written
Write burst	1	4	0 4	The first and last word can have one to four bytes written. The middle words can have zero to four bytes written
Write burst	0 or 1	8	0	**CLF** is asserted and all 32 bytes are written

Table 13: StrongARM Transactions

10.0 Boundary Scan Test Interface

The boundary-scan interface conforms to the IEEE Std. 1149.1- 1990, Standard Test Access Port and Boundary-Scan Architecture (please refer to this standard for an explanation of the terms used in this section and for a description of the TAP controller states.) StrongArm will only support JTAG continuity testing.

10.1 Overview

The boundary-scan interface provides a means of driving and sampling all the external pins of the device irrespective of the core state. This function permits testing of both the device's electrical connections to the circuit board, and (in conjunction with other devices on the circuit board having a similar interface) testing the integrity of the circuit board connections between devices. The interface intercepts all external connections within the device, and each such "cell" is then connected together to form a serial register (the boundary scan register). The whole interface is controlled via 5 dedicated pins: **TDI**, **TMS**, **TCK**, **nTRST** and **TDO**. *Figure 9: Test Access Port (TAP) Controller Sate Transitions* shows the state transitions that occur in the TAP controller.

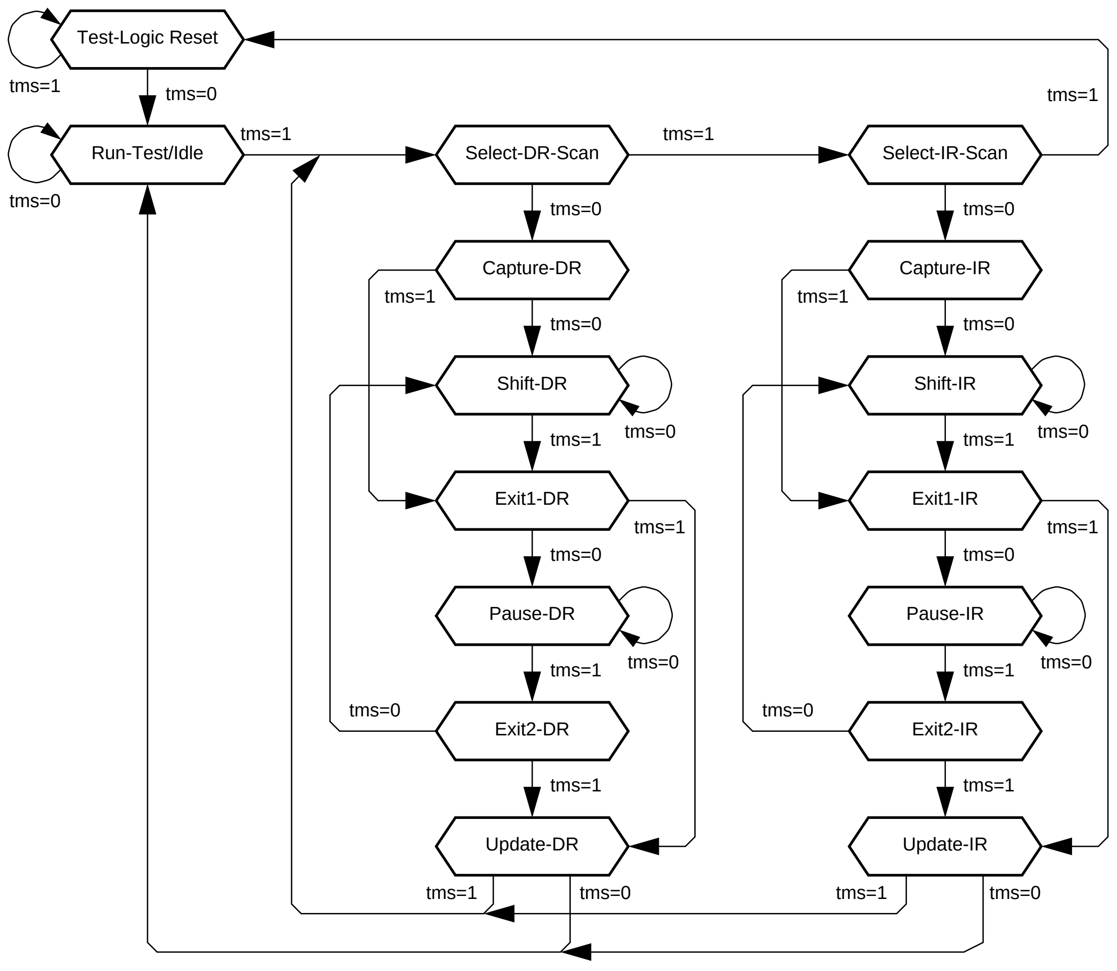


Figure 9: Test Access Port (TAP) Controller Sate Transitions

10.2 Reset

The boundary-scan interface includes a state-machine controller (the TAP controller). In order to force the TAP controller into the correct state after power-up of the device, a reset pulse must be applied to the **nTRST** pin. If the boundary scan interface is to be used, then **nTRST** must be driven LOW, and then HIGH again. If the boundary scan interface is not to be used, then the **nTRST** pin may be tied permanently LOW. Note that a clock on **TCK** is not necessary to reset the device.

The action of reset (either a pulse or a DC level) is as follows:

System mode is selected (i.e. the boundary scan chain does NOT intercept any of the signals passing between the pads and the core).

IDcode mode is selected. If **TCK** is pulsed, the contents of the ID register will be clocked out of **TDO**.

10.3 Pullup Resistors

The IEEE 1149.1 standard effectively requires that **TDI**, **nTRST** and **TMS** should have internal pullup resistors. In order to minimise static current draw, **nTRST** has an internal pulldown resistor. These four pins can be left unconnected for normal operation and overdriven to use the JTAG features.

10.4 nPWRSLP

The **nPWRSLP** input is not sampled by the JTAG interface and must be deasserted to use the JTAG interface.

10.5 Instruction Register

The instruction register is 5 bits in length.

There is no parity bit. The fixed value loaded into the instruction register during the CAPTURE-IR controller state is: 00001.

10.6 Public Instructions

The following public instructions are supported:

Instruction	**Binary Code**
EXTEST	00000
SAMPLE/PRELOAD	00001
CLAMP	00100
HIGHZ	00101
IDCODE	00110
BYPASS	11111
Private	00010, 00011, 00111, 01000-01111, 10000-11110

In the descriptions that follow, **TDI** and **TMS** are sampled on the rising edge of **TCK** and all output transitions on **TDO** occur as a result of the falling edge of **TCK**.

10.6.1 EXTEST (00000)

The BS (boundary-scan) register is placed in test mode by the EXTEST instruction.

The EXTEST instruction connects the BS register between **TDI** and **TDO**.

When the instruction register is loaded with the EXTEST instruction, all the boundary-scan cells are placed in their test mode of operation.

In the CAPTURE-DR state, inputs from the system pins and outputs from the boundary-scan output cells to the system pins are captured by the boundary-scan cells. In the SHIFT-DR state, the previously captured test data is shifted out of the BS register via the **TDO** pin, while new test data is shifted in via the **TDI** pin to the BS register parallel input latch. In the UPDATE-DR state, the new test data is transferred into the BS register parallel output latch. Note that this data is applied immediately to the system logic and system pins.

10.6.2 SAMPLE/PRELOAD (00001)

The BS (boundary-scan) register is placed in normal (system) mode by the SAMPLE/PRELOAD instruction.

The SAMPLE/PRELOAD instruction connects the BS register between **TDI** and **TDO**.

When the instruction register is loaded with the SAMPLE/PRELOAD instruction, all the boundary-scan cells are placed in their normal system mode of operation.

In the CAPTURE-DR state, a snapshot of the signals at the boundary-scan cells is taken on the rising edge of **TCK**. Normal system operation is unaffected. In the SHIFT-DR state, the sampled test data is shifted out of the BS register via the **TDO** pin, while new data is shifted in via the **TDI** pin to preload the BS register parallel input latch. In the UPDATE-DR state, the preloaded data is transferred into the BS register parallel output latch. Note that this data is not applied to the system logic or system pins while the SAMPLE/PRELOAD instruction is active. This instruction should be used to preload the boundary-scan register with known data prior to selecting EXTEST instructions (see the table below for appropriate guard values to be used for each boundary-scan cell).

10.6.3 CLAMP (00100)

The CLAMP instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO**.

When the CLAMP instruction is loaded into the instruction register, the state of all output signals is defined by the values previously loaded into the boundary-scan register. A guarding pattern (specified for this device at the end of this section) should be pre-loaded into the boundary-scan register using the SAMPLE/PRELOAD instruction prior to selecting the CLAMP instruction.

In the CAPTURE-DR state, a logic 0 is captured by the bypass register. In the SHIFT-DR state, test data is shifted into the bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero. The bypass register is not affected in the UPDATE-DR state.

10.6.4 HIGHZ (00101)

The HIGHZ instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO**.

When the HIGHZ instruction is loaded into the instruction register, all outputs are placed in an inactive drive state.

In the CAPTURE-DR state, a logic 0 is captured by the bypass register. In the SHIFT-DR state, test data is shifted into the bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero. The bypass register is not affected in the UPDATE-DR state.

10.6.5 IDCODE (00110)

The IDCODE instruction connects the device identification register (or ID register) between **TDI** and **TDO**. The ID register is a 32-bit register that allows the manufacturer, part number and version of a component to be determined through the TAP.

When the instruction register is loaded with the IDCODE instruction, all the boundary-scan cells are placed in their normal (system) mode of operation.

In the CAPTURE-DR state, the device identification code (specified at the end of this section) is captured by the ID register. In the SHIFT-DR state, the previously captured device identification code is shifted out of the ID register via the **TDO** pin, while data is shifted in via the **TDI** pin into the ID register. In the UPDATE-DR state, the ID register is unaffected.

10.6.6 BYPASS (11111)

The BYPASS instruction connects a 1 bit shift register (the BYPASS register) between **TDI** and **TDO**.

When the BYPASS instruction is loaded into the instruction register, all the boundary-scan cells are placed in their normal (system) mode of operation. This instruction has no effect on the system pins.

In the CAPTURE-DR state, a logic 0 is captured by the bypass register. In the SHIFT-DR state, test data is shifted into the bypass register via **TDI** and out via **TDO** after a delay of one **TCK** cycle. Note that the first bit shifted out will be a zero. The bypass register is not affected in the UPDATE-DR state.

10.7 Test Data Registers

Figure 10: Boundary Scan Block Diagram illustrates the structure of the boundary scan logic.

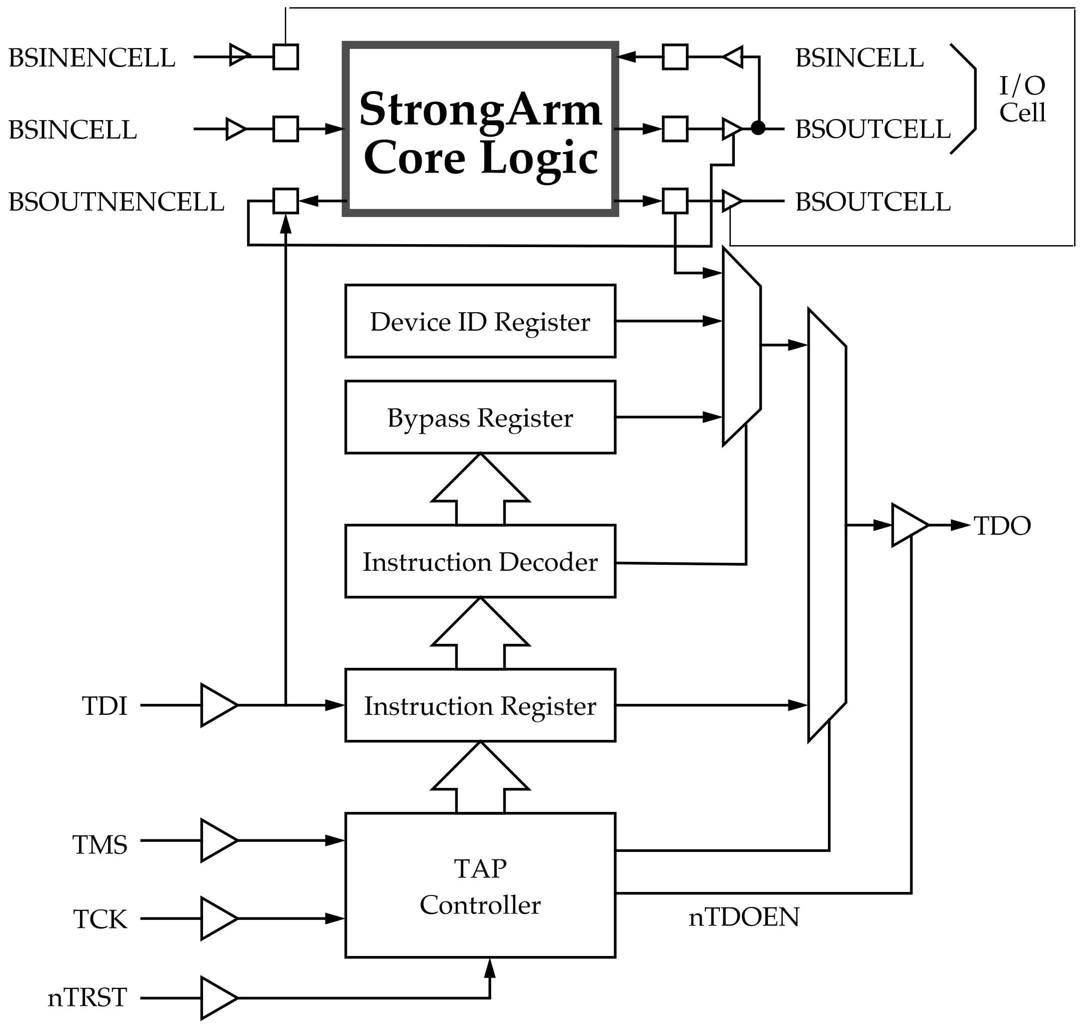


Figure 10: Boundary Scan Block Diagram

10.7.1 Bypass Register

Purpose: This is a single bit register which can be selected as the path between **TDI** and **TDO** to allow the device to be bypassed during boundary-scan testing.

Length: 1 bit

Operating Mode: When the BYPASS instruction is the current instruction in the instruction register, serial data is transferred from **TDI** to **TDO** in the SHIFT-DR state with a delay of one **TCK** cycle.

There is no parallel output from the bypass register.

A logic 0 is loaded from the parallel input of the bypass register in the CAPTURE-DR state.

10.7.2 StrongArm Device Identification (ID) Code Register

Purpose: This register is used to read the 32-bit device identification code. No programmable supplementary identification code is provided.

Length: 32 bits

The format of the ID register is as follows:

31 28	27 12	11 0
version	**Part Number**	**JEDEC code**

The high order four bits of the ID register are the version and bits 27..0 are 0x102C06B.

Operating Mode: When the IDCODE instruction is current, the ID register is selected as the serial path between **TDI** and **TDO**.

There is no parallel output from the ID register.

The 32-bit device identification code is loaded into the ID register from its parallel inputs during the CAPTURE-DR state.

10.7.3 StrongArm Boundary Scan (BS) Register

Purpose: The BS register consists of a serially connected set of cells around the periphery of the device, at the interface between the core logic and the system input/output pads. This register can be used to isolate the pins from the core logic and then drive or monitor the system pins.

Operating modes: The BS register is selected as the register to be connected between **TDI** and **TDO** only during the SAMPLE/PRELOAD, and EXTEST instructions. Values in the BS register are used, but are not changed, during the CLAMP instruction.

In the normal (system) mode of operation, straight-through connections between the core logic and pins are maintained and normal system operation is unaffected.

In TEST mode (i.e. when EXTEST is the currently selected instruction), values can be applied to the output pins independently of the actual values on the input pins and core logic outputs. On the StrongArm all of the boundary scan cells include an update register and thus all of the pins can be controlled in the above manner. An additional boundary-scan cell is interposed in the scan chain in order to control the enabling of the data bus.

The correspondence between boundary-scan cells and system pins, system direction controls and system output enables is as shown in the Boundary Scan Signals & Pins table . The cells are listed in the order in which they are connected in the boundary-scan register, starting with the cell closest to **TDI**.

The EXTEST guard values specified in the Boundary Scan Signals & Pins table should be clocked into the boundary-scan register (using the SAMPLE/PRELOAD instruction) before the EXTEST instruction is selected, to ensure that known data is applied to the core logic during the test. This guard value should also be used when new EXTEST vectors are clocked into the boundary-scan register.

The values stored in the BS register after power-up are not defined. Similarly, the values previously clocked into the BS register are not guaranteed to be maintained across a Boundary Scan reset (from forcing nTRST LOW or entering the Test Logic Reset state).

10.8 Boundary Scan Interface Signals

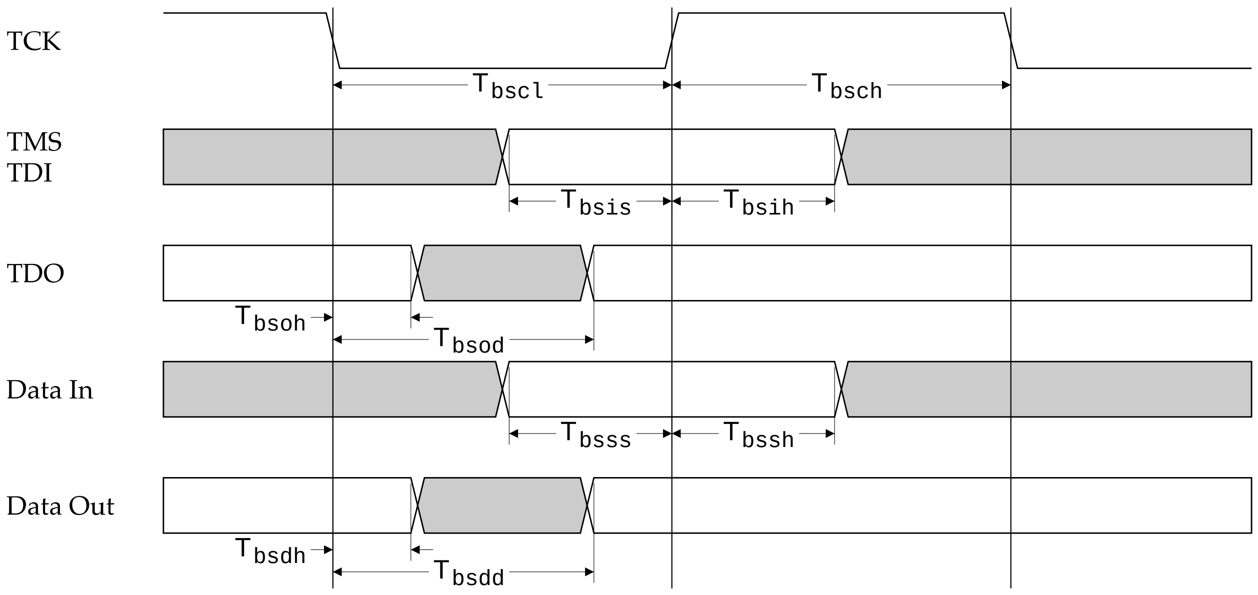


Figure 11: Boundary Scan General Timing

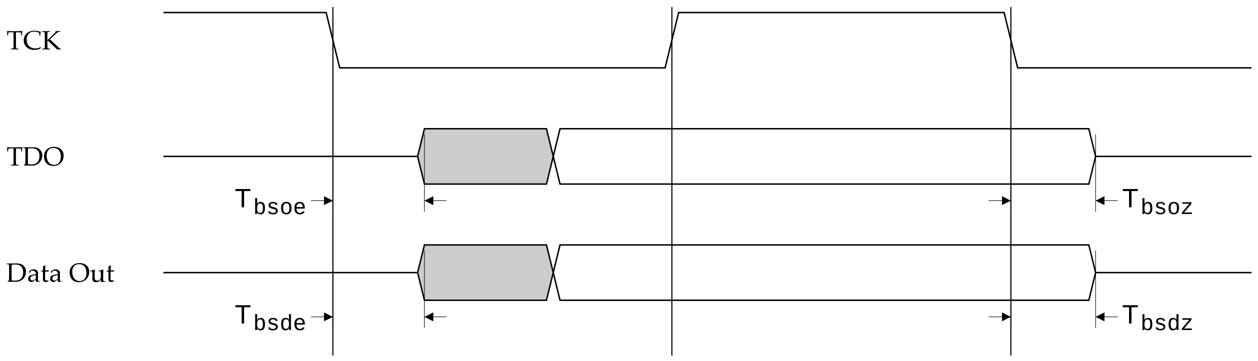


Figure 12: Boundary Scan Tri-state Timing

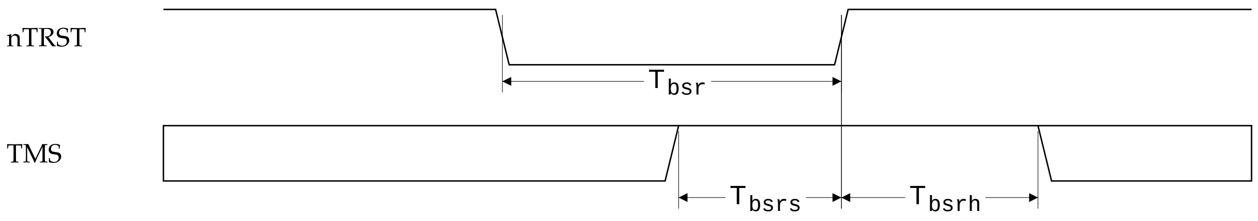


Figure 13: Boundary Scan Reset Timing

Symbol	Parameter	Min	Typ	Max	Units	Notes
Tbscl	**TCK** low period	50			ns	9
Tbsch	**TCK** high period	50			ns	9
Tbsis	**TDI**,**TMS** setup to [TCr]	10			ns	
Tbsih	**TDI**,**TMS** hold from [TCr]	10			ns	
Tbsoh	**TDO** hold time	5			ns	1
Tbsod	TCf to **TDO** valid			40	ns	1
Tbsss	I/O signal setup to [TCr]	5			ns	4
Tbssh	I/O signal hold from [TCr]	20			ns	4
Tbsdh	data output hold time	5			ns	5
Tbsdd	TCf to data output valid			40	ns	
Tbsoe	**TDO** enable time	5			ns	1,2
Tbsoz	**TDO** disable time			40	ns	1,3
Tbsde	data output enable time	5			ns	5,6
Tbsdz	data output disable time			40	ns	5,7
Tbsr	Reset period	30			ns	
Tbsrs	tms setup to [TRr]	10			ns	9
Tbsrh	tms hold from [TRr]	10			ns	9

Table 14: StrongArm Boundary Scan Interface Timing

Notes:

1. Assumes a 25pF load on **TDO**. Output timing derates at 0.072ns/pF of extra load applied.
2. **TDO** enable time applies when the TAP controller enters the Shift-DR or Shift-IR states.
3. **TDO** disable time applies when the TAP controller leaves the Shift-DR or Shift-IR states.
4. For correct data latching, the I/O signals (from the core and the pads) must be setup and held with respect to the rising edge of **TCK** in the CAPTURE-DR state of the SAMPLE/PRELOAD and EXTEST instructions.
5. Assumes that the data outputs are loaded with the AC test loads (see AC parameter specification).
6. Data output enable time applies when the boundary scan logic is used to enable the output drivers.
7. Data output disable time applies when the boundary scan is used to disable the output drivers.
8. **TMS** must be held high as **nTRST** is taken high at the end of the boundary-scan reset sequence.
9. **TCK** may be stopped indefinitely in either the low or high phase.

No.	Cell Name	Pin	Type	Output enable BS Cell	Guard Value EX	
from tdi						
1	mse	MSE	IN	-		
2	config	CONFIG	IN	-		
3	nmreq	nMREQ	OUT	mse		
4	seq	SEQ	OUT	mse		
5	clf	CLF	OUT	abe		
6	lock	LOCK	OUT	abe		
7	nrw	nRW	OUT	abe		
8	spdf	SPDF	OUT	-		
9	mccfg0	MCCFG[0]	IN	-		
10	mccfg1	MCCFG[1]	IN	-		
11	mccfg2	MCCFG[2]	IN	-		
12	cccfg0	CCCFG[2]	IN	-		
13	cccfg[1]	CCCFG[3]	IN	-		
14	nresetout	nRESET_OUT	OUT	-		
15	abe	ABE	IN	abe		
16	mas0	MAS[0]	OUT	abe		
17	mas1	MAS[1]	OUT	abe		
18	a0	A[0]	OUT	abe		
19	a1	A[1]	OUT	abe		
20	a2	A[2]	OUT	abe		
21	a3	A[3]	OUT	abe		
22	a4	A[4]	OUT	abe		
23	a5	A[5]	OUT	abe		
24	a6	A[6]	OUT	abe		
25	a7	A[7]	OUT	abe		
26	a8	A[8]	OUT	abe		
27	a9	A[9]	OUT	abe		
28	a10	A[10]	OUT	abe		
29	a11	A[11]	OUT	abe		
30	a12	A[12]	OUT	abe		
31	a13	A[13]	OUT	abe		
32	a14	A[14]	OUT	abe		
33	a15	A[15]	OUT	abe		
34	a16	A[16]	OUT	abe		
34	a17	A[17]	OUT	abe		
35	a18	A[18]	OUT	abe		
36	a19	A[19]	OUT	abe		
37	a20	A[20]	OUT	abe		
38	a21	A[21]	OUT	abe		
39	a22	A[22]	OUT	abe		
40	a23	A[23]	OUT	abe		
41	a24	A[24]	OUT	abe		
42	a25	A[25]	OUT	abe		
43	a26	A[26]	OUT	abe		
44	a27	A[27]	OUT	abe		
45	a28	A[28]	OUT	abe		
46	a29	A[29]	OUT	abe		
47	a30	A[30]	OUT	abe		
48	a31	A[31]	OUT	abe		
49	ape	APE	IN	-		
50	abort	ABORT	IN	-		
51	mclkin	MCLK	IN	-		
52	mclkout	MCLK	OUT	sna		
53	nmclk	NMCLK	OUT	sna		
54	nwait	nWAIT	IN	-		
55	clk	CLK	IN	-		
56	test_byp	TEST_BYP	IN	-		
57	testclk	TESTCLK	IN	-		
58	cccfg1	CCCFG[1]	IN	-		
59	cccfg0	CCCFG[0]	IN			
60	nreset	nRESET	IN			
61	sna	SnA	IN	-		
62	fiq	FIQ	IN	-		
63	irq	IRQ	IN	-		
64	din0	D[0]	IN	-		
65	dout0	D[0]	OUT	endout		
67	din1	D[1]	IN	-		
68	dout1	D[1]	OUT	endout		
69	din2	D[2]	IN	-		
70	dout2	D[2]	OUT	endout		
71	din3	D[3]	IN	-		
72	dout3	D[3]	OUT	endout		
73	din4	D[4]	IN	-		
74	dout4	D[4]	OUT	endout		
75	din5	D[5]	IN	-		
76	dout5	D[5]	OUT	endout		
77	din6	D[6]	IN	-		
78	dout6	D[6]	OUT	endout		
79	din7	D[7]	IN	-		
80	dout7	D[7]	OUT	endout		
81	din8	D[8]	IN	-		
82	dout8	D[8]	OUT	endout		
83	din9	D[9]	IN	-		
84	dout9	D[9]	OUT	endout		
85	din10	D[10]	IN	-		
86	dout10	D[10]	OUT	endout		
87	din11	D[11]	IN	-		
88	dout11	D[11]	OUT	endout		
89	din12	D[12]	IN	-		
90	dout12	D[12]	OUT	endout		
91	din13	D[13]	IN	-		
92	dout13	D[13]	OUT	endout		
93	din14	D[14]	IN	-		

Table 15: Boundary Scan Signals & Pins

No.	Cell Name	Pin	Type	Output enable BS Cell	Guard Value EX	
94	dout14	D[14]	OUT	endout		
95	din15	D[15]	IN	-		
96	dout15	D[15]	OUT	endout		
97	din16	D[16]	IN	-		
98	dout16	D[16]	OUT	endout		
99	din17	D[17]	IN	-		
100	dout17	D[17]	OUT	endout		
101	din18	D[18]	IN	-		
102	dout18	D[18]	OUT	endout		
103	din19	D[19]	IN	-		
104	dout19	D[19]	OUT	endout		
105	din20	D[20]	IN	-		
106	dout20	D[20]	OUT	endout		
107	din21	D[21]	IN	-		
108	dout21	D[21]	OUT	endout		
109	din22	D[22]	IN	-		
110	dout22	D[22]	OUT	endout		
111	din23	D[23]	IN	-		
112	dout23	D[23]	OUT	endout		
113	din24	D[24]	IN	-		
114	dout24	D[24]	OUT	endout		
115	din25	D[25]	IN	-		
116	dout25	D[25]	OUT	endout		
117	din26	D[26]	IN	-		
118	dout26	D[26]	OUT	endout		
119	din27	D[27]	IN	-		
120	dout27	D[27]	OUT	endout		
121	din28	D[28]	IN	-		
122	dout28	D[28]	OUT	endout		
123	din29	D[29]	IN	-		
124	dout29	D[29]	OUT	endout		
125	din30	D[30]	IN	-		
126	dout30	D[30]	OUT	endout		
127	din31	D[31]	IN	-		
128	dout31	D[31]	OUT	endout		
129	endout		OUTEN1			

to TDO

Table 15: Boundary Scan Signals & Pins

Key:		
	IN	Input pad
	OUT	Output pad
	INEN1	Input enable active high
	OUTEN1	Output enable active high

11.0 DC Parameters

11.1 Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Units	Note
VDD	Core Supply voltage	VSS-0.5	VSS+2.2	V	1
VDDX	I/O voltage	MIN(VSS-.05, VDD-.3)	VSS+3.6	V	1
Vip	Voltage applied to any pin	VSS-0.5	VSS+3.6	V	1
Ts	Storage temperature	-40	125	deg C	1

Table 16: StrongArm DC Maximum Ratings

Note:

These are stress StrongArm 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.

11.2 DC Operating Conditions

Symbol	Parameter	Min	Typ	Max	Units	Notes
VDD	Core Supply voltage	1.5	1.65-2.0	2.2	V	
VDDX	Switched I/O Supply	3.0	3.3	3.6		
Vihc	IC input HIGH voltage	.9*VDDX		VDDX	V	1,2
Vilc	IC input LOW voltage	0.0		.1*VDDX	V	1,2
Vohc	OCZ output HIGH voltage	.8*VDDX		VDDX	V	1,2
Volc	OCZ output LOW voltage	0.0		0.2*VDDX	V	1,2
Ta	Ambient operating temperature	0		70	°C	

Table 17: StrongArm **DC Operating Conditions**

Notes:

(1) Voltages measured with respect to VSS.

(2) IC - CMOS-level inputs (includes IC and ICOCZ pin types)

(3) OCZ - Output, CMOS levels, tri-stateable

11.3 DC Characteristics

Symbol	Parameter	Nom	Units	Note
Iin	IC input leakage current		uA	
Ioh	Output HIGH current (Vout = VDD-0.4V)		mA	
Iol	Output LOW current (Vout = VSS+0.4V)		mA	
Cin	Input capacitance	5	pF	
ESD	HMB model ESD	2	KV	2

Table 18: StrongARM DC Characteristics

Notes:

(1) Nominal values shown are derived from transient analysis simulations.

(2) ESD - 2 KV minimum CDM

Normal operation mode power is 300mW at 100MHz and 1.65V Vdd, 450mW at 160MHz and 1.65V Vdd, 900mW at 200MHz and 2V Vdd. Idle current is less than 20 mW. Sleep mode current is less than 50µA with VDD off and nPWRSLP asserted.

12.0 AC Parameters

NOTE: The AC parameters are still subject to change.

12.1 Test Conditions

The AC timing diagrams presented in this section assume that the outputs of StrongArm have been loaded with a 50pf capacitive load. The output pads of StrongArm are CMOS drivers which exhibit a propagation delay that increases 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 or decreasing load capacitance.

Output Signal	Output Derating (ns/pF) 50pf load
`A[31:0], nR/W,LOCK MAS,CLF`	tbs
`D[31:0]`	tbs
`nMREQ,SEQ`	tbs

Table 19: StrongArm AC Test Conditions

12.2 Module Considerations

The edge rates for the StrongARM processor are such that the lumped load model presented above can only be used for etch lengths up to 1 inch. Over one inch of etch the signal is a transmission line and needs to be modeled as such.

12.3 Main Bus Signals

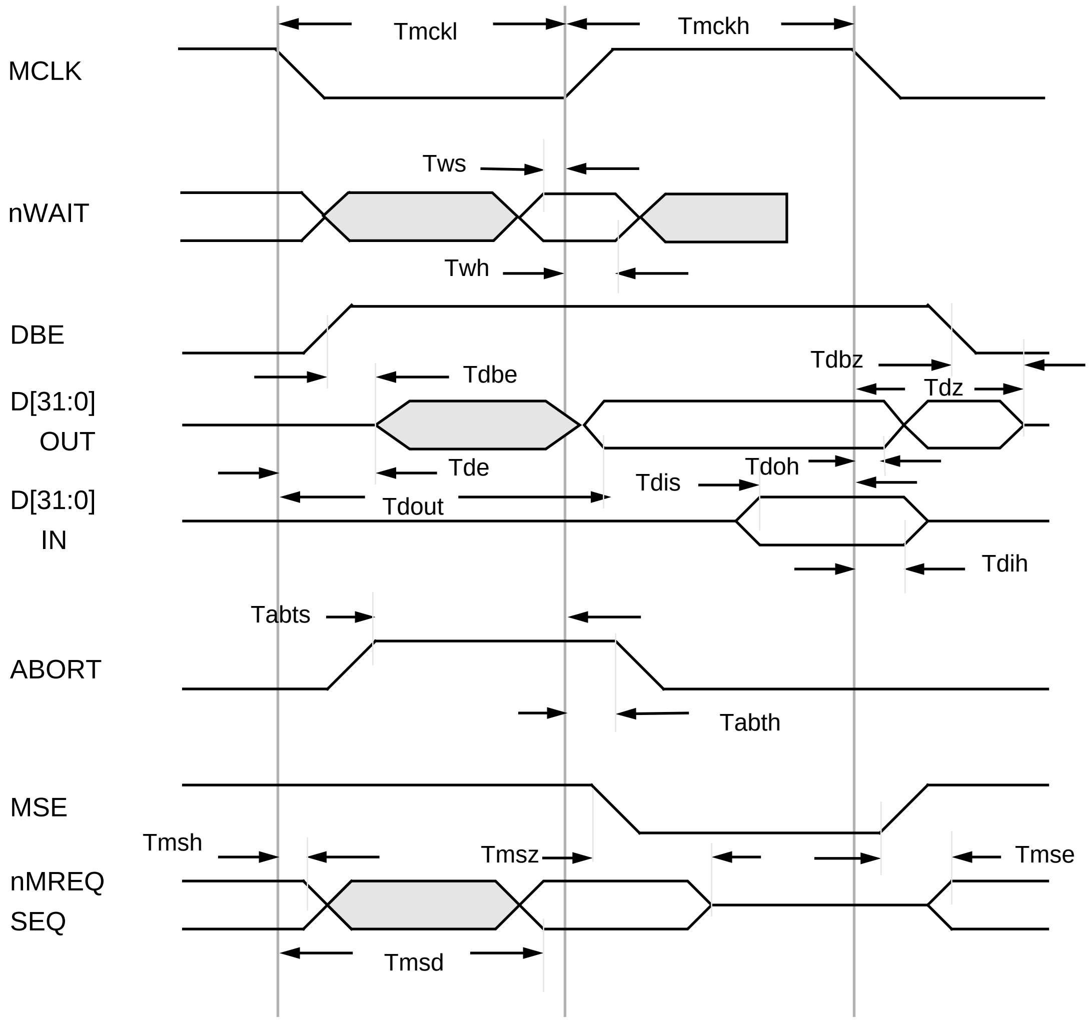


Figure 14: StrongArm Main Bus Timing

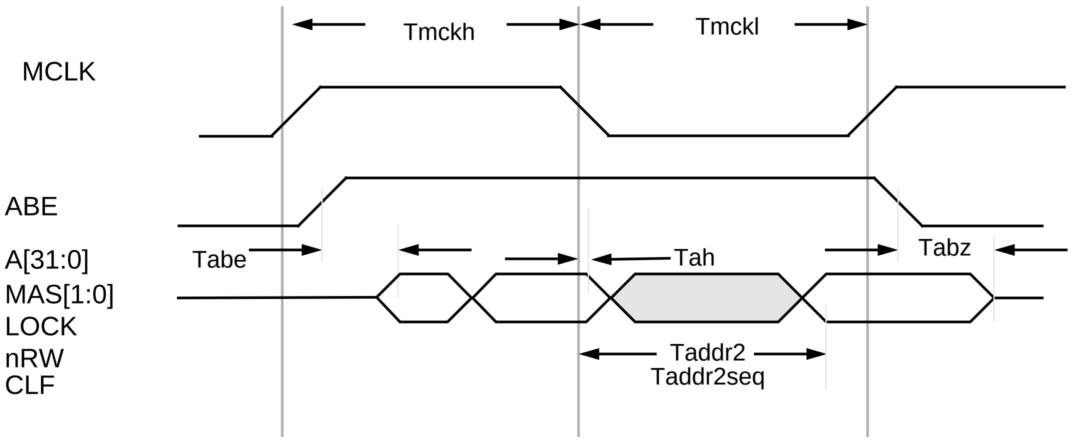


Figure 15: StrongArm Address timing with APE=LOW

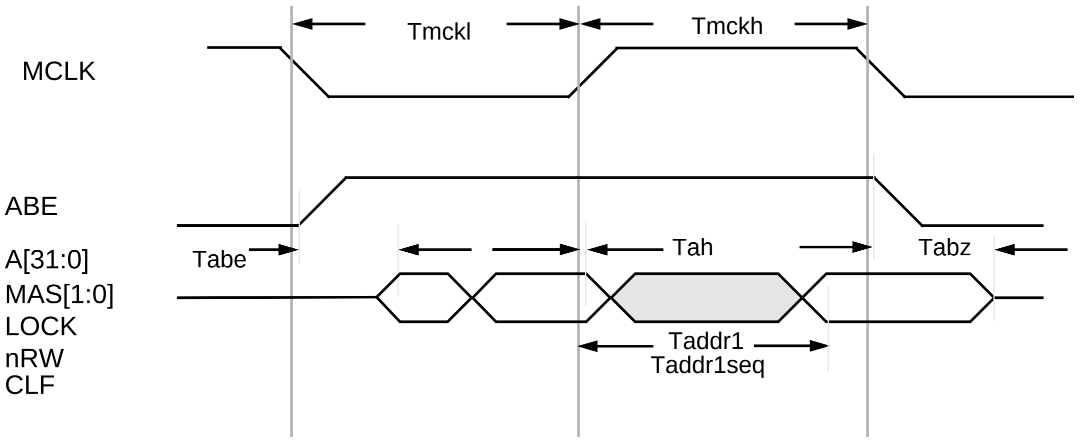


Figure 16: StrongArm Address timing with APE=HIGH

Symbol	Parameter	1.5V Min	1.5V Max	2.0V Min	2.0V Max	Unit	Note
Tmckl	MCLK LOW time	10		7.5		ns	1
Tmckh	MCLK HIGH time	10		7.5		ns	1
Tws	nWAIT setup to MCLK	2		2		ns	2
Twh	nWAIT hold from MCLK	3		3		ns	2
Tabe	address bus enable		7.5		7.5	ns	2
Tabz	address bus disable		3		3	ns	2
Taddr1	MCLK to address delay APE high		17		15	ns	2,3
Taddr2	MCLK to address delay APE low		11		9	ns	2,3
Taddr1seq	MCLK to address delay APE high		11		9	ns	2,3
Taddr2seq	MCLK to address delay APE low		11		9	ns	2,3
Tah	address hold time	2		1		ns	2
Tdbe	DBE to data enable		7.5		7.5	ns	2
Tde	MCLK to data enable	1		1		ns	2
Tdbz	DBE to data disable		3		3	ns	2
Tdz	MCLK to data disable		4.5		3	ns	2
Tdout	data out delay		8		6	ns	2
Tdoh	data out hold	2		1		ns	2
Tdis	data in setup	1		1		ns	2
Tdih	data in hold	2		2		ns	2
Tabts	ABORT setup time	1		1		ns	2
Tabth	ABORT hold time	2		2		ns	2
Tmse	nMREQ & SEQ enable		8		8	ns	2
Tmsz	nMREQ & SEQ disable		3		3	ns	2
Tmsd	nMREQ & SEQ delay		8		6	ns	2
Tmsh	nMREQ & SEQ hold	2		1		ns	2

Table 20: StrongArm Bus timing

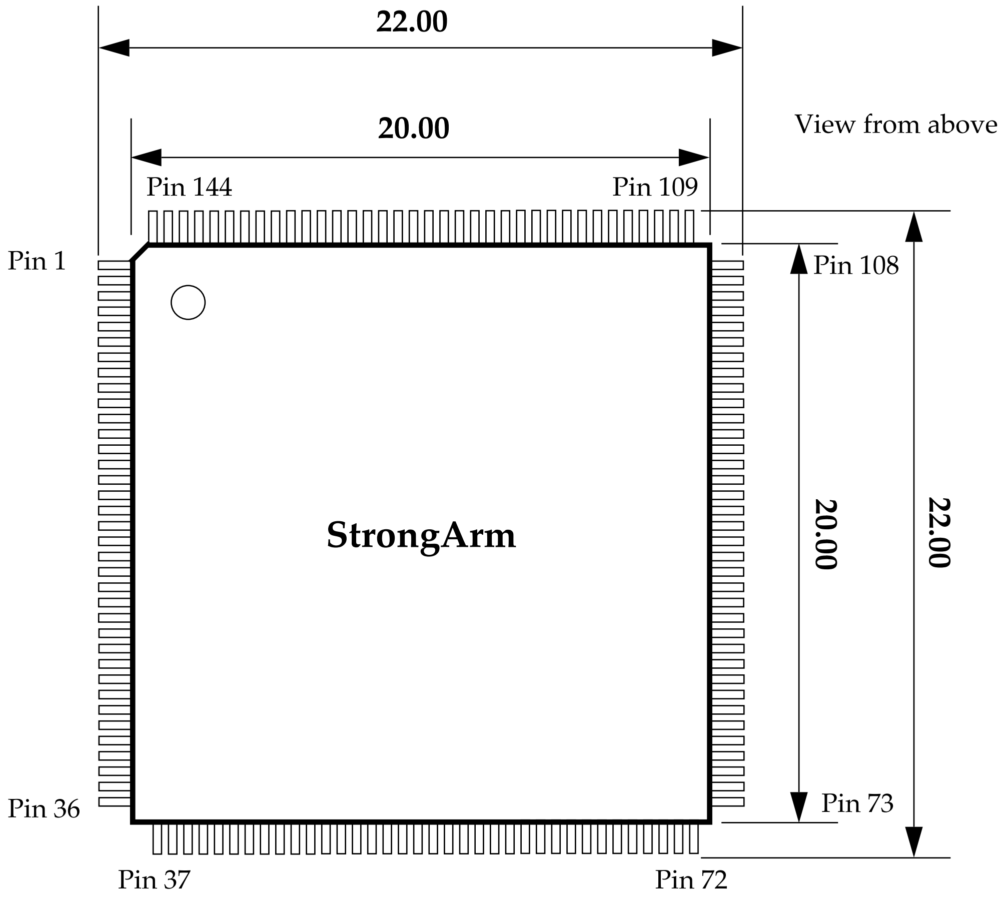
Notes:

(1) **MCLK** timings measured between clock edges at 90%/10% of Vdd.

(2) The timings of these buses are measured to 80%/20% for outputs and 90%/10% for inputs.

(3) The Taddrx times are for the first word of a burst. The Taddrxseq times are for the subsequent words of a burst.

13.0 Physical Details



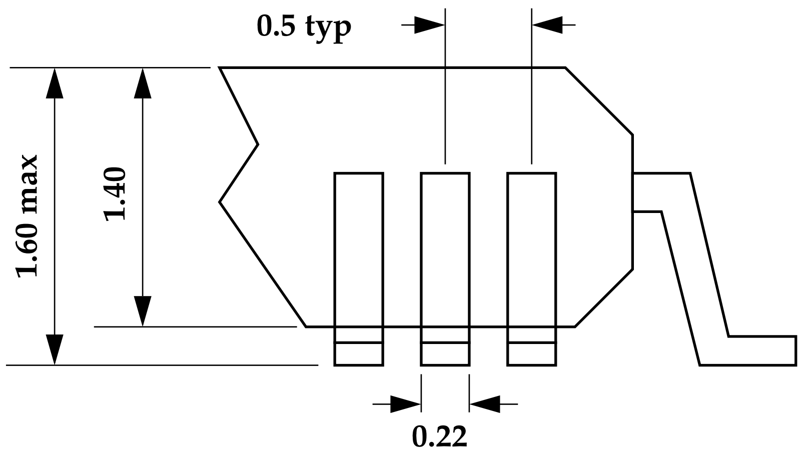


Figure 17: StrongARM 144 Pin TQFP Mechanical Dimensions in mm

13.1 Pinout

Pin	Signal	Type
1	D[0]	i/o
2	D[1]	i/o
3	D[2]	i/o
4	D[3]	i/o
5	VDD	-
6	VSS	-
7	VSSX	-
8	VDDX	-
9	D[4]	i/o
10	D[5]	i/o
11	D[6]	i/o
12	D[7]	i/o
13	D[8]	i/o
14	D[9]	i/o
15	D[10]	i/o
16	D[11]	i/o
17	VDDX	-
18	VSSX	-
19	VSS	-
20	VDD	-
21	D[12]	i/o
22	D[13]	i/o
23	D[14]	i/o
24	D[15]	i/o
25	D[16]	i/o
26	D[17]	i/o
27	D[18]	i/o
28	D[19]	i/o
29	VDDX	-
30	VSSX	-
31	VSS	-
32	VDD	-
33	D[20]	i/o
34	D[21]	i/o
35	D[22]	i/o
36	DBE	i
37	D[23]	i/o
38	D[24]	i/o
39	D[25]	i/o
40	D[26]	i/o
41	D[27]	i/o
42	VSSX	-
43	VDDX	-
44	D[28]	i/o
45	D[29]	i/o
46	D[30]	i/o
47	D[31]	i/o
48	TDO	o
49	TDI	i
50	nTRST	i
51	TMS	i
52	TCK	i
53	n/c	
54	VSS	-
55	VDD	-
56	MSE	i
57	CONFIG	i
58	nMREQ	o
59	SEQ	o
60	CLF	o
61	LOCK	o
62	nRW	o
63	VSSX	-
64	VDDX	-
65	SPDF	i
66	MCCFG[0]	i
67	MCCFG[1]	i
68	MCCFG[2]	i
69	CCCFG[2]	i
70	CCCFG[3]	i
71	nRESET_OUT	o
72	ABE	i
73	MAS[0]	o
74	MAS[1]	o
75	A[0]	o
76	A[1]	o
77	A[2]	o
78	VDDX	-
79	VSSX	-
80	VSS	-
81	VDD	-
82	A[3]	o
83	A[4]	o
84	A[5]	o
85	A[6]	o
86	A[7]	o
87	A[8]	o
88	A[9]	o
89	A[10]	o
90	VDD	-
91	VSS	-
92	VSSX	-
93	VDDX	-
94	A[11]	o
95	A[12]	o
96	A[13]	o
97	A[14]	o
98	A[15]	o
99	A[16]	o
100	A[17]	o
101	A[18]	o
102	VDDX	-
103	VSSX	-
104	VSS	-
105	VDD	-
106	A[19]	o
107	A[20]	o
108	A[21]	o
109	A[22]	o
110	A[23]	o
111	A[24]	o
112	A[25]	o
113	VSSX	-
114	VDDX	-
115	A[26]	o
116	A[27]	o
117	A[28]	o
118	A[29]	o
119	A[30]	o
120	A[31]	o
121	APE	i
122	ABORT	i
123	MCLK	i/o
124	nMCLK	o
125	VSSX	-
126	VDDX	-
127	nWAIT	i
128	CLK	i
129	VDD	-
130	TCK_BYP	i
131	TESTCLK	i
132	VSS	-
133	VDD	-
134	n/c	
135	n/c	
136	n/c	
137	n/c	
138	CCCFG[1]	i
139	CCCFG[0]	i
140	nPWRSLP	i
141	nRESET	i
142	SnA	i
143	nFIQ	i
144	nIRQ	i

Table 21: Pinout - StrongArm in 144 pin Thin Quad Flat Pack