ARM7TDMI (Rev 3) Core Processor

Product Overview

ARM®

ARM7TDMI (Rev 3) Core Processor
Product Overview

Copyright © 2000, 2001. All rights reserved.

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

1 Applications and benefits

Applications

- personal digital assistants

- cell phones

- pagers

- automotive

- modems

- personal audio products.

Benefits

- designed specifically for ASIC and ASSP integration

- supports the Thumb® instruction set to enable 32-bit performance at 16-bit, or even 8-bit cost and increased code density

- high performance allows system designers to integrate more functionality into both price and power sensitive applications

- very low power consumption

- wide range of development tools from ARM and third party suppliers.

Performance

- 0.9MIPS/MHz

- Typical power consumption:
    - at 0.25μm; <0.80mW/MHz
    - at 0.18μm; <0.25mW/MHz

- Typical size:
    - at 0.25μm; 1.00mm^2
    - at 0.18μm; 0.53mm^2

2 The ARM7 family

The ARM7 family includes the ARM7TDMI, ARM7TDMI-S, ARM720T, and ARM7EJ-S processors.

The ARM7TDMI core is the industry's most widely used 32-bit embedded risc microprocessor solution. optimized for cost and power-sensitive applications, the ARM7TDMI solution provides the low power consumption, small size, and high performance needed in portable, embedded applications.

The ARM7TDMI-s core is the synthesizable version of the ARM7TDMI core, available in both verilog and vhdl, ready for compilation into processes supported by in-house or commercially available synthesis libraries. Optimized for flexibility and featuring an identical feature set to the hard macrocell, it improves time-to-market by reducing development time while allowing for increased design flexibility, and enabling >>98% fault coverage.

The ARM720T hard macrocell contains the ARM7TDMI core, 8kb unified cache, and a *Memory Management Unit* (MMU) that allows the use of protected execution spaces and virtual memory. This macrocell is compatible with leading operating systems including windows ce, linux, palm os, and symbian os.

The ARM7EJ-S processor is a synthesizable core that provides all the benefits of the ARM7TDMI – low power consumption, small size, and the thumb instruction set – while also incorporating ARM's latest DSP extensions and Jazelle technology, enabling acceleration of java-based applications.

2.1 Compatible with the ARM9™, ARM9E™, and ARM10™ families, and StrongARM® architecture

software written for the ARM7TDMI processor is 100% binary-compatible with other members of the ARM7 family and forwards-compatible with the ARM9, ARM9E, and ARM10 families, as well as products in intel's StrongARM and xscale architectures. This gives designers a choice of software-compatible processors with strong price-performance points. Support for the ARM architecture today includes:

- operating systems such as windows ce, linux, palm os, and the symbian os

- more than 40 real-time operating systems, including qnx, wind river's vxworks, and mentor graphics' vrtx

- cosimulation tools from leading eda vendors

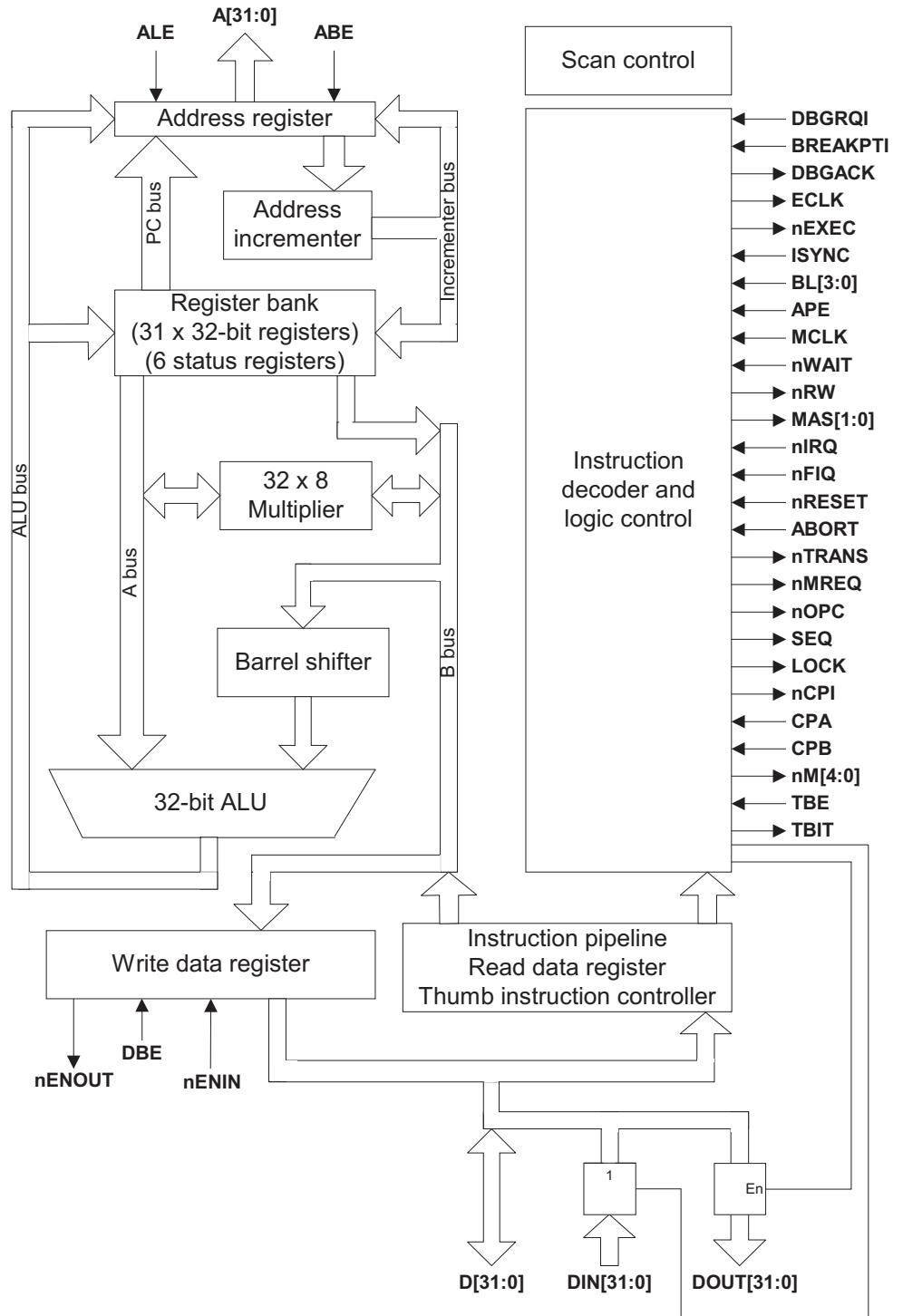- a variety of software development tools.

Figure 1 ARM7TDMI core diagram

3 ARM7TDMI

The ARM7TDMI core is based on the von neumann architecture with a 32-bit data bus that carries both instructions and data. Load, store, and swap instructions can access data from memory. Data can be 8-bit, 16-bit, and 32-bit.

3.1 Instruction pipeline

The ARM7TDMI core uses a three-stage pipeline to increase the flow of instructions to the processor. This allows multiple simultaneous operations to take place and continuous operation of the processing and memory systems.

The instructions are executed in three stages:

- fetch
- decode
- execute.

3.2 Memory interface

The ARM7TDMI memory interface is designed to allow optimum performance potential and minimize memory usage. Speed critical control signals are pipelined to allow system control functions to exploit the fast-burst access modes supported by many memory technologies.

the ARM7TDMI has four basic types of memory cycle:

- internal
- nonsequential
- sequential
- coprocessor register transfer.

There is also the option to use either a single bidirectional data bus or two separate unidirectional data input and output buses.

3.3 Memory formats

The ARM7TDMI can be configured to treat stored words in either big-endian or little-endian format.

3.4 Performance, code density and operating states

The ARM7TDMI core supports two operating states and instruction sets:

- ARM state for 32-bit, word-aligned instructions
- thumb state for 16-bit, halfword-aligned instructions.

The ARM instruction set allows a program to achieve maximum performance with the minimum number of instructions. The simpler thumb instruction set offers much increased code density reducing memory requirement. Code can switch between the ARM and thumb instruction sets on any procedure call.

The majority of ARM7TDMI instructions are executed in a single cycle. These are shown in Table 1 on page 8.

 ARM DVI 0027B

3.5 Operating modes

The ARM7TDMI core has seven modes of operation:

- User mode is the usual program execution state

- Fast Interrupt (FIQ) mode supports data transfer or channel processes to allow very fast interrupt processing and to preserve values across interrupt calls

- Interrupt (IRQ) mode is used for general purpose interrupt handling

- Supervisor mode is a protected mode for the operating system

- Abort mode is entered after a data or instruction prefetch abort

- System mode is a privileged user mode for the operating system

- Undefined mode is entered when an undefined instruction is executed.

3.6 Coprocessors

Up to 16 coprocessors can be connected to an ARM7TDMI system.

3.7 Debug features

The ARM7TDMI processor core incorporates hardware extensions for advanced debugging features to simplify the development of application software, operating systems, and hardware. The debug extensions allow the core to be forced into debug state.

The internal state of the ARM7TDMI core can be examined using a jtag interface to allow the insertion of instructions into the core pipeline and avoid using the external data bus.

A typical debug system comprises:

- a debug host (a computer running a toolkit from ARM or third party)

- a protocol converter to serve as the communications point between the high-level commands issued by the debug host and the low-level commands of the jtag interface

- the target core, ARM7TDMI.

The ARM7TDMI core includes an internal functional unit known as the embeddedice logic. The embeddedice logic is configured to monitor ARM7TDMI core activity for specific instruction fetches and data accesses. Execution halts when the values pre-programmed match the current values causing a breakpoint or watchpoint, respectively. Configuration is done through a dedicated scan chain via the JTAG interface.

The ARM7TDMI can also be connected to an *Embedded Trace Macrocell* (ETM). The etm provides comprehensive debug and trace facilities by allowing information on the processor's state to be captured before and after a specific event, whilst the core runs at full speed. A dedicated, configurable trace port and fifo allow the compressed trace data to be read out by an external trace port analyser without affecting the processor.

3.8 Instruction speed summary

Due to the pipelined architecture of the CPU, instructions overlap considerably. In a typical cycle, one instruction can be using the data path while the next is being decoded and the one after that is being fetched. For this reason Table 1 on page 8 presents the incremental number of

---

cycles required by an instruction, rather than the total number of cycles for which the instruction uses part of the processor. Elapsed time, in cycles, for a routine can be calculated from the figures listed in Table 1.

These figures assume that the instruction is actually executed. Unexecuted instructions take one sequential cycle. In Table 1:

- n is the number of words transferred
- m is 1 if bits [32:8] of the multiplier operand are all zero or all one
- m is 2 if bits [32:16] of the multiplier operand are all zero or all one
- m is 3 if bits [31:24] of the multiplier operand are all zero or all one
- b is the number of cycles spent in the coprocessor busy-wait loop
- n is a nonsequential memory cycles is a sequential memory cycle
- i is an internal memory cycle
- c is a coprocessor register transfer memory cycle.

Table 1 ARM instruction speed summary

Instruction	Cycle count	Additional
Data Processing	1S	+ 1I for SHIFT(Rs)
+ 1S + 1N if R15 written		
MSR, MRS	1S	-
LDR	1S+1N+1I	+ 1S + 1N if R15 loaded
STR	2N	-
LDM	nS+1N+1I	+ 1S + 1N if R15 loaded
STM	(n-1)S+2N	-
SWP	1S+2N+1I	-
B,BL	2S+1N	-
SWI	2S+1N	-
MUL,MLA	1S+mI	-
MUL	1S+mI	-
MLA	1S+(m+1)I	-
MULL	1S+(m+1)I	-
MLAL	1S+(m+2)I	-
CDP	1S+bI	-
LDC,STC	(n-1)S+2N+bI	-
MCR	1N+bI+1C	-
MRC	1S+(b+1)I+1C	-

3.9   signals

Table 2 on page 9 lists and describes all of the signals used for the ARM7TDMI.

Table 2 Signal Descriptions

Name	Type	Description
A[31:0] Address bus	Output	This is the 32-bit address bus. **ALE**, **ABE** and **APE** are used to control when the address bus is valid.
ABE Address bus enable	Input	The address bus drivers are disabled when this is LOW, putting the address bus into a high impedance state. This also controls the **LOCK**, **MAS[1:0]**, **nRW**, **nOPC**, and **nTRANS** signals in the same way. **ABE** must be tied HIGH if there is no system requirement to disable the address drivers.
ABORT Memory abort	Input	The memory system uses this signal to tell the processor that a requested access is not allowed.
ALE Address latch enable	Input	This signal is provided for backwards compatibility with older ARM processors; for new designs, if address re-timing is required, ARM recommends the use of **APE**, and for **ALE** to be connected HIGH. The address bus, **LOCK**, **MAS[1:0]**, **nRW**, **nOPC**, and **nTRANS** signals are latched when this is held LOW. This allows these address signals to be held valid for the complete duration of a memory access cycle. For example, when interfacing to ROM, the address must be valid until after the data has been read.
APE Address timing pipeline enable	Input	Selects whether the address bus, **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS**, and **nOPC** signals operate in pipelined (**APE** is HIGH) or de-pipelined mode (**APE** is LOW). Pipelined mode is particularly useful for DRAM systems, where it is desirable to provide the address to the memory as early as possible, to allow longer periods for address decoding and the generation of DRAM control signals. In this mode, the address bus does not remain valid to the end of the memory cycle. De-pipelined mode can be useful for SRAM and ROM access. Here the address bus, **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS**, and **nOPC** signals must be kept stable throughout the complete memory cycle. However, this does not provide optimum performance.
BIGEND Big-endian configuration	Input	Selects how the processor treats bytes in memory: HIGH for big-endian format; LOW for little-endian.
BL[3:0] Byte latch control	Input	The values on the data bus is latched on the falling edge of **MCLK** when these signals are HIGH. For most designs these signals should be tied HIGH.
BREAKPT Breakpoint	Input	A conditional request for the processor to enter debug state is made by placing this signal HIGH. If the memory access at that time is an instruction fetch, the processor enters debug state only if the instruction reaches the execution stage of the pipeline. If the memory access is for data, the processor enters debug state after the current instruction completes execution. This allows extension of the internal breakpoints provided by the EmbeddedICE logic.
BUSDIS Bus disable	Output	When INTEST is selected on scan chain 0, 4, or 8 this is HIGH. It can be used to disable external logic driving onto the bidirectional data bus during scan testing. This signal changes after the falling edge of **TCK**.
BUSEN Data bus configuration	Input	A static configuration signal that selects whether the bidirectional data bus (**D[31:0]**) or the unidirectional data busses (**DIN[31:0]** and **DOUT[31:0]**) are used for transfer of data between the processor and memory. When **BUSEN** is LOW, **D[31:0]** is used; **DOUT[31:0]** is driven to a value of zero, and **DIN[31:0]** is ignored, and should be tied LOW. When **BUSEN** is HIGH, **DIN[31:0]** and **DOUT[31:0]** are used; **D[31:0]** is ignored and must be left unconnected.
COMMRX Communications channel receive	Output	When the communications channel receive buffer is full this is HIGH. This signal changes after the rising edge of **MCLK**.

Table 2 Signal Descriptions (continued)

Name	Type	Description
COMMTX Communications channel transmit	Output	When the communications channel transmit buffer is empty this is HIGH. This signal changes after the rising edge of **MCLK**.
CPA Coprocessor absent	Input	Placed LOW by the coprocessor if it is capable of performing the operation requested by the processor.
CPB Coprocessor busy	Input	Placed LOW by the coprocessor when it is ready to start the operation requested by the processor. It is sampled by the processor when **MCLK** goes HIGH in each cycle in which nCPI is LOW.
D[31:0] Data bus	Input/ Output	Used for data transfers between the processor and external memory.During read cycles input data must be valid on the falling edge of **MCLK**.During write cycles output data remains valid until after the falling edge of **MCLK**.This bus is always driven except during read cycles, irrespective of the value of **BUSEN**. Consequently it must be left unconnected if using the unidirectional data busses.
DBE Data bus enable	Input	Must be HIGH for data to appear on either the bi-directional or unidirectional data output bus. When LOW the bidirectional data bus is placed into a high impedance state and data output is prevented on the unidirectional data output bus.It can be used for test purposes or in shared bus systems.
DBGACK Debug acknowledge	Output	When the processor is in a debug state this is HIGH.
DBGEN Debug enable	Input	A static configuration signal that disables the debug features of the processor when held LOW. This signal must be HIGH to allow the EmbeddedICE logic to function.
DBGRQ Debug request	Input	A request for the processor to enter debug state after executing the current instruction is made by placing this signal HIGH.
DBGRQI Internal debug request	Output	This is the logical OR of **DBGRQ** and bit 1 of the debug control register.
DIN[31:0] Data input bus	Input	Unidirectional bus used to transfer instructions and data from the memory to the processor. This bus is only used when **BUSEN** is HIGH; if unused then it should be tied LOW.This bus is sampled during read cycles on the falling edge of **MCLK**.
DOUT[31:0] Data output bus	Output	Unidirectional bus used to transfer data from the processor to the memory system. This bus is only used when **BUSEN** is HIGH; otherwise it is driven to a value of zero.During write cycles the output data becomes valid while **MCLK** is LOW, and remains valid until after the falling edge of **MCLK**.
DRIVEBS Boundary scan cell enable	Output	Controls the multiplexors in the scan cells of an external boundary scan chain. This must be left unconnected, if an external boundary scan chain is not connected.
ECAPCLK EXTEST capture clock	Output	Only used on the ARM7TDMI test chip, and must otherwise be left unconnected.
ECAPCLKBS EXTEST capture clock for boundary scan	Output	Used to capture the device inputs of an external boundary scan chain during EXTEST. When scan chain 3 is selected, the current instruction is EXTEST and the TAP controller state machine is in the CAPTURE- DR state, then this signal is a pulse equal in width to **TCK2**. This must be left unconnected, if an external boundary scan chain is not connected.

Table 2 Signal Descriptions (continued)

Name	Type	Description
ECLK External clock output	Output	In normal operation, this is simply **MCLK**, optionally stretched with nWAIT, exported from the core. When the core is being debugged, this is **DCLK**, which is generated internally from **TCK**.
EXTERN0 External input 0	Input	This is connected to the EmbeddedICE logic and allows breakpoints and watchpoints to be dependent on an external condition.
EXTERN1 External input 1	Input	This is connected to the EmbeddedICE logic and allows breakpoints and watchpoints to be dependent on an external condition.
HIGHZ High impedance	Output	When the **HIGHZ** instruction has been loaded into the TAP controller this signal is HIGH.
ICAPCLKBS INTEST capture clock	Output	This is used to capture the device outputs in an external boundary scan chain during INTEST. This must be left unconnected, if an external boundary scan chain is not connected.
IR[3:0] TAP controller instruction register	Output	Reflects the current instruction loaded into the TAP controller instruction register. These bits change on the falling edge of **TCK** when the state machine is in the UPDATE-IR state.
ISYNC Synchronous interrupts	Input	Set this HIGH if **nIRQ** and **nFIQ** are synchronous to the processor clock; LOW for asynchronous interrupts.
LOCK Locked operation	Output	When the processor is performing a locked memory access this is HIGH. This is used to prevent the memory controller allowing another device to access the memory. It is active only during the data swap (SWP) instruction. This is one of the signals controlled by **APE**, **ALE** and **ABE**.
MAS[1:0] Memory access size	Output	Used to indicate to the memory system the size of data transfer (byte, halfword or word) required for both read and write cycles, become valid before the falling edge of **MCLK** and remain valid until the rising edge of **MCLK** during the memory cycle. The binary values 00, 01, and 10 represent byte, halfword and word respectively (11 is reserved).This is one of the signals controlled by **APE**, **ALE** and **ABE**.
MCLK Memory clock input	Input	This is the main clock for all memory accesses and processor operations. The clock speed can be reduced to allow access to slow peripherals or memory. Alternatively, the **nWAIT** can be used with a free-running **MCLK** to achieve the same effect.
nCPI Not coprocessor instruction	Output	LOW when a coprocessor instruction is processed. The processor then waits for a response from the coprocessor on the **CPA** and **CPB** lines. If **CPA** is HIGH when **MCLK** rises after a request has been initiated by the processor, then the coprocessor handshake is aborted, and the processor enters the undefined instruction trap.If CPA is LOW at this time, then the processor will enters a busy-wait period until **CPB** goes LOW before completing the coprocessor handshake.
nENIN NOT enable input	Input	This must be LOW for the data bus to be driven during write cycles. Can be used in conjunction with **nENOUT** to control the data bus during write cycles.
nENOUT Not enable output	Output	During a write cycle, this signal is driven LOW before the rising edge of **MCLK**, and remains LOW for the entire cycle. This can be used to aid arbitration in shared bus applications.
nENOUTI Enable output internal	Output	During a coprocessor register transfer C-cycle from the EmbeddedICE communications channel coprocessor to the ARM core, this signal goes LOW. This can be used to aid arbitration in shared bus systems.

Table 2 Signal Descriptions (continued)

Name	Type	Description
nEXEC Not executed	Output	When the instruction in the execution unit is not being executed because, for example, it has failed its condition code check, this is HIGH.
nFIQ Not fast interrupt request	Input	Taking this LOW causes the processor to be interrupted if the appropriate enable in the processor is active. The signal is level-sensitive and must be held LOW until a suitable response is received from the processor. **nFIQ** can be synchronous or asynchronous to **MCLK**, depending on the state of ISYNC.
nHIGHZ Not HIGHZ	Output	When the current instruction is **HIGHZ** this signal is LOW. This is used to place the scan cells of that scan chain in the high impedance state. This must be left unconnected, if an external boundary scan chain is not connected.
nIRQ Not interrupt request	Input	As **nFIQ**, but with lower priority. Can be taken LOW to interrupt the processor when the appropriate enable is active. **nIRQ** can be synchronous or asynchronous to **MCLK**, depending on the state of **ISYNC**.
nM[4:0] Not processor mode	Output	These are the inverse of the internal status bits indicating the current processor mode.
nMREQ Not memory request	Output	When the processor requires memory access during the following cycle this is LOW.
nOPC Not op-code fetch	Output	When the processor is fetching an instruction from memory this is LOW. This is one of the signals controlled by **APE**, **ALE** and **ABE**.
nRESET Not reset	Input	Used to start the processor from a known address. A LOW level causes the instruction being executed to terminate abnormally.This signal must be held LOW for at least two clock cycles, with **nWAIT** held HIGH. When LOW the processor performs internal cycles with the address incrementing from the point where reset was activated. The address overflows to zero if **nRESET** is held beyond the maximum address limit. When HIGH for at least one clock cycle, the processor restarts from address 0.
nRW Not read or Write	Output	When the processor is performing a read cycle, this is LOW. This is one of the signals controlled by APE, ALE and ABE.
nTDOEN Not TDO enable	Output	When serial data is being driven out on **TDO** this is LOW. Normally used as an output enable for a **TDO** pin in a packaged part.
nTRANSNot memory translate	Output	When the processor is in User mode, this is LOW. It can be used either to tell the memory management system when address translation is turned on, or as an indicator of non-User mode activity.This is one of the signals controlled by **APE**, **ALE** and **ABE**.
nTRST Not test reset	Input	Reset signal for the boundary scan logic. This pin must be pulsed or driven LOW to achieve normal device operation, in addition to the normal device reset, **nRESET**.
nWAIT Not wait	Input	When LOW the processor extends an access over a number of cycles of **MCLK**, which is useful for accessing slow memory or peripherals. Internally, **nWAIT** is logically ANDed with MCLK and must only change when **MCLK** is LOW.If **nWAIT** is not used it must be tied HIGH.
PCLKBS Boundary scan update clock	Output	This is used by an external boundary scan chain as the update clock. This must be left unconnected, if an external boundary scan chain is not connected.

Table 2 Signal Descriptions (continued)

Name	Type	Description
RANGEOUT0 EmbeddedICE **RANGEOUT0**	Output	When the EmbeddedICE watchpoint unit 0 has matched the conditions currently present on the address, data, and control busses, then this is HIGH. This signal is independent of the state of the watchpoint enable control bit.**RANGEOUT0** changes when ECLK is LOW.
RANGEOUT1 EmbeddedICE **RANGEOUT1**	Output	As **RANGEOUT0** but corresponds to the EmbeddedICE watchpoint unit 1.
RSTCLKBS Boundary scan reset clock	Output	When either the TAP controller state machine is in the RESET state or when **nTRST** is LOW, then this is HIGH. This can be used to reset external boundary scan cells.
SCREG[3:0] Scan chain register	Output	These reflect the ID number of the scan chain currently selected by the TAP controller. These change on the falling edge of **TCK** when the TAP state machine is in the UPDATE-DR state.
SDINBS Boundary scan serial input data	Output	This provides the serial data for an external boundary scan chain input. It changes from the rising edge of **TCK** and is valid at the falling edge of **TCK**.
SDOUTBS Boundary scan serial output data	Input	Accepts serial data from an external boundary scan chain output, synchronized to the rising edge of **TCK**. This must be tied LOW, if an external boundary scan chain is not connected.
SEQ Sequential address	Output	When the address of the next memory cycle is closely related to that of the last memory access, this is HIGH. In ARM state the new address can be for the same word or the next; in THUMB state, the same halfword or the next.It can be used, in combination with the low-order address lines, to indicate that the next cycle can use a fast memory mode (for example DRAM page mode) or to bypass the address translation system.
SHCLKBS Boundary scan shift clock, phase one	Output	Used to clock the master half of the external scan cells and follows **TCK1** when in the SHIFT-DR state of the state machine and scan chain 3 is selected. When not in the SHIFT-DR state or when scan chain 3 is not selected, this clock is LOW.
SHCLK2BS Boundary scan shift clock, phase two	Output	As **SHCLKBS** but follows **TCK2** instead of **TCK1**. This must be left unconnected, if an external boundary scan chain is not connected.
TAPSM[3:0] TAP controller state machine	Output	These reflect the current state of the TAP controller state machine. These bits change on the rising edge of **TCK**.
TBE Test bus enable	Input	When LOW, **D[31:0]**, **A[31:0]**, **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS**, and **nOPC** are set to high impedance.
TBIT Thumb bit	Output	When the processor is executing the THUMB instruction set, this is HIGH; LOW when executing the ARM instruction set.
TCK Test clockInput		Clock signal for all test circuitry. When in debug state, this is used to generate **DCLK**, **TCK1** and **TCK2**.
TCK1 Test clock, phase one	Output	HIGH when **TCK** is HIGH (slight phase lag due to the internal clock non-overlap).
TCK2 Test clock, phase two	Output	HIGH when **TCK** is LOW (slight phase lag due to the internal clock non-overlap). It is the non-overlapping complement of **TCK1**.

Table 2 Signal Descriptions (continued)

Name	Type	Description
TDI Test data input	Input	Serial data for the scan chains.
TDO Test data output	Output	Serial data from the scan chains.
TMS Test mode select	Input	Mode select for scan chains.
VDD Power supply	Power	Provide power to the device.
VSS Ground	Power	These connections are the ground reference for all signals.

4 ARM Architecture v4T

4.1 ARM7TDMI processor core

The ARM7TDMI processor core implements the ARMv4T *Instruction Set Architecture* (ISA). This is a superset of the ARMv4 ISA which adds support for the 16-bit Thumb instruction set. Software using the Thumb instruction set is compatible with all members of the ARM Thumb family, including ARM9, ARM9E, and ARM10 families.

4.2 Registers

The ARM7TDMI core consists of a 32-bit datapath and associated control logic. This datapath contains 31 general-purpose 32-bit registers, 7 dedicated 32-bit registers coupled to a barrel-shifter, Arithmetic Logic Unit, and multiplier.

4.3 Modes and exceptions

The ARM7TDMI supports seven modes of operation:
- User mode
- Fast Interrupt (FIQ)
- Interrupt (IRQ)
- Supervisor mode
- Abort mode
- Undefined mode
- System mode.

All modes other than User are privileged modes. These are used to service hardware interrupts, exceptions, and software interrupts. Each privileged mode has an associated *Saved Program Status Register* (SPSR). This register is use to save the state of the *Current Program Status Register* (CPSR) of the task immediately before the exception occurs.

In these privileged modes, mode-specific banked registers are available. These are automatically restored to their original values on return to the previous mode and the saved CPSR restored from the SPSR.

System mode does not have any banked registers. It uses the User mode registers. System mode runs tasks that require a privileged processor mode and allows them to invoke all classes of exception.
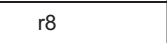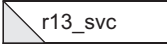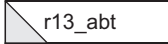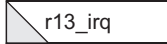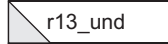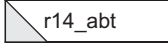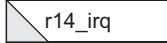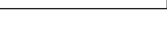
4.4 Processor states

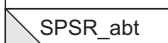The ARM7TDMI processor can be in one of two states:

ARM state

In ARM state, 16 general registers and one or two status registers are accessible at any one time. The registers available to the programmer in each mode, in ARM state, are illustrated in Figure 2 on page 16.

ARM-state general registers and program counter

General registers	System and User	FIQ	Supervisor	Abort	IRQ	Undefined
	r0	r0	r0	r0	r0	r0
	r1	r1	r1	r1	r1	r1
	r2	r2	r2	r2	r2	r2
	r3	r3	r3	r3	r3	r3
	r4	r4	r4	r4	r4	r4
	r5	r5	r5	r5	r5	r5
	r6	r6	r6	r6	r6	r6
	r7	r7	r7	r7	r7	r7
	r8	r8_fiq	r8	r8	r8	r8
	r9	r9_fiq	r9	r9	r9	r9
	r10	r10_fiq	r10	r10	r10	r10
	r11	r11_fiq	r11	r11	r11	r11
	r12	r12_fiq	r12	r12	r12	r12
	r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
	r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
Program counter	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

Program status registers						
	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

ARM-state program status registers

◺ *= banked register*

Figure 2 Register organization in ARM state

Thumb state

In Thumb state, eight general registers, the *Program Counter* (PC), *Stack Pointer* (SP), *Link Register* (LR), and *Current Program Status Register* (CPSR) are accessible. The registers available to the programmer in each mode, in Thumb state, are illustrated in Figure 3 on page 17.

ARM DVI 0027B

Thumb state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

Thumb state program status registers

◣ = *banked register*

Figure 3 Register organization in Thumb state

4.5 Exceptions

The ARM7TDMI supports seven types of exception:

- FIQ – fast interrupt
- IRQ – normal interrupt
- Data abort
- Prefetch abort
- Software interrupt
- Undefined instruction
- Reset.

All exceptions have banked registers for R14 and R13. After an exception, R14 holds the return address for exception processing. This address is used both to return after the exception is processed and to address the instruction that caused the exception.

R13 is banked across exception modes to provide each exception handler with a private stack pointer. The fast interrupt mode also banks registers 8 to 12 so that interrupt processing can begin without the need to save or restore these registers.

4.6 Status registers

All other processor states are held in status registers. The current operating processor status is in the CPSR. The CPSR holds:

- four ALU flags (Negative, Zero, Carry, and Overflow)
- an interrupt disable bit for each of the FIQ and IRQ interrupts
- a bit to indicate ARM or Thumb execution state
- five bits to encode the current processor mode.

The program status register format is shown in Figure 4 on page 19.

4.7 Conditional execution

All ARM instructions are conditionally executed and can optionally update the four condition code flags (Negative, Zero, Carry, and Overflow) according to their result. Fifteen conditions are implemented.

4.8 Classes of instructions

The ARM and Thumb instruction sets can be divided into four broad classes of instruction:
- data processing instructions
- load and store instructions
- branch instructions
- coprocessor instructions.

4.9 Data processing instructions

The data processing instructions operate on data held in general-purpose registers. Of the two source operands, one is always a register. The other has two basic forms:

- an immediate value

- a register value, optionally shifted.

If the operand is a shifted register the shift amount can have an immediate value or the value of another register. Four types of shift can be specified. Most data processing instructions can perform a shift followed by a logical or arithmetic operation.

Multiply instructions come in two classes:
- normal, 32-bit result
- long, 64-bit result variants.

Both types of multiply instruction can optionally perform an accumulate operation.

4.10 Load and store instructions

Single or multiple registers can be loaded and stored at one time.

Load and store single register instructions can transfer a 32-bit word, a 16-bit halfword, or an 8-bit byte between memory and a register. Byte and halfword loads can be automatically zero extended or sign extended as they are loaded.

Load and store instructions have three primary addressing modes:
- offset
- pre-indexed
- post-indexed.

The address is formed by adding or subtracting an immediate or register- based offset to or from a base register. Register-based offsets can also be scaled with shift operations. Pre-indexed and post-indexed addressing modes update the base register with the result of the offset calculation.

As the PC is a general-purpose register, a 32-bit value can be loaded directly into the PC to perform a jump to any address in the 4GB memory space.

   ARM DVI 0027B

Load and store multiple instructions perform a block transfer of any number of the general purpose registers to or from memory. Four addressing modes are provided:

- pre-increment addressing
- post-increment addressing
- pre-decrement addressing
- post-decrement addressing.

The base address is specified by a register value (that can be optionally updated after the transfer). As the subroutine return address and the PC values are in general-purpose registers, very efficient subroutine calls can be constructed.
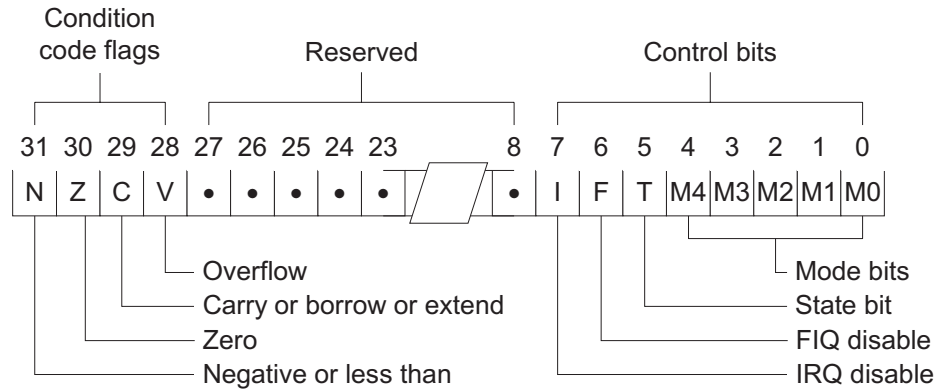

Figure 4 Program status register format

4.11 Branch instructions

As well as allowing any data processing or load instruction to change control flow (by modifying the PC) a standard branch instruction is provided with 24-bit signed offset, allowing forward and backward branches of up to 32MB. Branch with Link (BL) allows efficient subroutine calls, and preserves the address of the instruction after the branch in R14 (the Link Register or LR). This allows a move instruction to put the LR in to the PC and return to the instruction after the branch.The third type of branch (BX) switches between ARM and Thumb instruction sets. The return address can be preserved in the LR as an option.

4.12 Coprocessor

There are three types of coprocessor instructions:

- coprocessor data processing instructions invoke a coprocessor specific internal operation

- coprocessor register transfer instructions allow a coprocessor value to be transferred to or from an ARM register

- coprocessor data transfer instructions transfer coprocessor data to or from memory, where the ARM calculates the memory address of the transfer.

5   System Issues and Third Party Support

This section contains:

- *JTAG debug*
- *AMBA bus architecture*
- *AMBA Design Kit*
- *Everything you need*
- *Current support.*

5.1   JTAG debug

The internal state of the ARM7TDMI is examined through a JTAG-style serial interface. This allows instructions to be serially inserted into the pipeline of the core without using the external data bus. For example, when in debug state, a Store-Multiple (STM) instruction can be inserted into the pipeline. This exports the contents of the ARM7TDMI registers. This data can be serially shifted out without affecting the rest of the system.

5.2   AMBA bus architecture

The ARM7 Thumb family processors are designed for use with the *Advanced Microcontroller Bus Architecture* (AMBA) multi-master on-chip bus architecture. AMBA is an open standard that describes a strategy for the interconnection and management of functional blocks that makes up a *System-on-Chip* (SoC). The AMBA specification defines three buses:

- Advanced System Bus (ASB)
- Advanced High-performance Bus (AHB)
- Advanced Peripheral Bus (APB).

ASB and AHB are used to connect high-performance system modules. APB offers a simpler interface for low-performance peripherals.

5.3   AMBA Design Kit

ARM's AMBA Design Kit product is a versatile toolkit aimed at enabling the successful creation of AMBA-based SoC designs.

The AMBA Design Kit includes an AHB 'wrapper' to enable the ARM7TDMI to be connected directly to the AHB system bus.

5.4   Everything you need

ARM provides a wide range of products and services to support its processor families, including software development tools, development boards, models, applications software, training, and consulting services.The ARM architecture today enjoys broad third-party support. The ARM7 Thumb family processors' strong software compatibility with existing ARM devices ensures that users benefit immediately from existing support.

5.5   Current support

Support for the ARM Architecture today includes:

- ARM Developer Suite (ADS)
    - integrated development environment
    - C, C++, assembly, simulators and windowing source-level debugger

- — available on Windows95, Windows NT, and Unix

- ARM Multi-ICE™ JTAG interface
  - — allows EmbeddedICE software debug of ARM processor systems

- ARMulator, an instruction accurate software simulator

- Development boards

- Design Signoff Models provide signoff quality ASIC-simulation

- Software toolkits available from ARM, RedHat/GNU, Greenhills, JavaSoft, MetaWare, and WindRiver allowing software development in C, C++, Java, FORTRAN, Pascal, Ada, and assembly

- More than 40 Real Time Operating Systems including:
  - — Windriver VxWorks
  - — Mentor Graphics VRTX
  - — WindRiver pSOSystem

- The following major Operating Systems:
  - — Microsoft Windows CE
  - — Linux
  - — Palm OS
  - — Symbian OS.

- Application software components:
  - — speech and image compression
  - — software modem
  - — Chinese character input network protocols
  - — Digital AC3 decode
  - — MPEG3 encode and decode
  - — MPEG4 decode and encode.

- Hardware and software cosimulation tools from leading EDA Vendors.

For more information, see `www.arm.com`