Technical Manual

ARM946E-S Microprocessor Core with Cache

June 2001

LSI LOGIC®

Document DB14-000104-00, First Edition (June 2001)
This document describes Rev 0A of the LSI Logic Corporation ARM946E-S and will remain the official reference source for all revisions/releases of this product until rescinded by an update.

To receive product literature, visit us at http://www.lsilogic.com.

BM

Preface

This book is the primary reference and Technical Reference Manual for the ARM946E-S. It contains a complete functional description for the product and includes complete physical and electrical specifications for this product.

Audience

This document assumes that you have some familiarity with microprocessors and related support devices. The people who benefit from this book are:

- Engineers and managers who are evaluating the processor for possible use in a system
- Engineers who are designing the processor into a system

Organization

This document has the following chapters and appendixes:

Chapter 1, **Introduction**, provides an introduction to the ARM946E-S.

Chapter 2, **Signal Descriptions**, describes the signals used in the ARM946E-S.

Chapter 3, **Programmer's Model**, describes the programmer's model of the ARM946E-S and includes a summary of the ARM946E-S coprocessor registers.

Chapter 4, **Caches**, describes the ARM946E-S cache implementation.

Chapter 5, **Protection Unit**, describes the ARM946E-S protection unit.

Chapter 6, **Tightly Coupled SRAM**, describes the requirements and operation of the tightly coupled SRAM.

Chapter 7, **Bus Interface Unit and Write Buffer**, describes the operation of the Bus Interface Unit and write buffer.

Chapter 8, **External Coprocessor Interface**, describes the coprocessor interface and the operation of common coprocessor instructions.

Chapter 9, **Debug Interface**, describes the debug support for the ARM946E-S and the EmbeddedICE-RT logic.

Chapter 10, **ETM Interface**, describes the ETM interface, including details of how to enable the interface.

Chapter 11, **Test Support**, describes the test methodology used for the ARM946E-S synthesized logic and tightly coupled SRAM.

Appendix A, **AC Parameters**, describes the timing parameters applicable to the ARM946E-S.

Related Publications

ARM Architecture Reference Manual available from ARM Ltd. as document No. ARM DDI 0100.

ARM9E-S Technical Reference Manual available from ARM Ltd. as document No.ARM DDI 0165.

AMBA Specification (Rev 2.0) available from ARM Ltd. as document No. ARM IHI 0011.

Embedded Trace Macrocell Specification (Rev 1.0) available from ARM Ltd. as document number IHI 0014E.

Standard Test Access Port and Boundary-Scan Architecture, IEEE Std. 1149.1-1990

Conventions Used in This Manual

The first time a word or phrase is defined in this manual, it is *italicized.*

The word *assert* means to drive a signal true or active. The word *deassert* means to drive a signal false or inactive. Signals that are active LOW end in an “n.”

Hexadecimal numbers are indicated by the prefix “0x”—for example, 0x32CF. Binary numbers are indicated by the prefix “0b”—for example, 0b0011.0010.1100.1111.

Contents

Tables

Chapter 1
Introduction

This chapter introduces the ARM946E-S microprocessor core. It contains the following sections:

1.1 About the ARM946E-S

The ARM946E-S is a *synthesizable* macrocell that includes an ARM processor. The ARM9E-S processor is a member of the ARM9 Thumb family of high-performance, 32-bit system-on-a-chip processor solutions.

The ARM946E-S has a tightly coupled SRAM memory, and both instruction and data caches. It is targeted for a wide range of embedded applications where high performance, low system cost, small die size, and low power are all important.

The ARM946E-S macrocell is a Harvard architecture, cached processor that provides a complete high-performance subsystem. The ARM946E-S includes:

- ARM9E-S RISC integer processor core:

  The processor uses the ARMv5TExP 32-bit instruction set with improved ARM/Thumb code interworking and an enhanced multiplier designed for improved DSP performance.

  The processor has an ARM debug architecture with additional support for real-time debug. This capability allows critical exception handlers to execute while debugging the system.

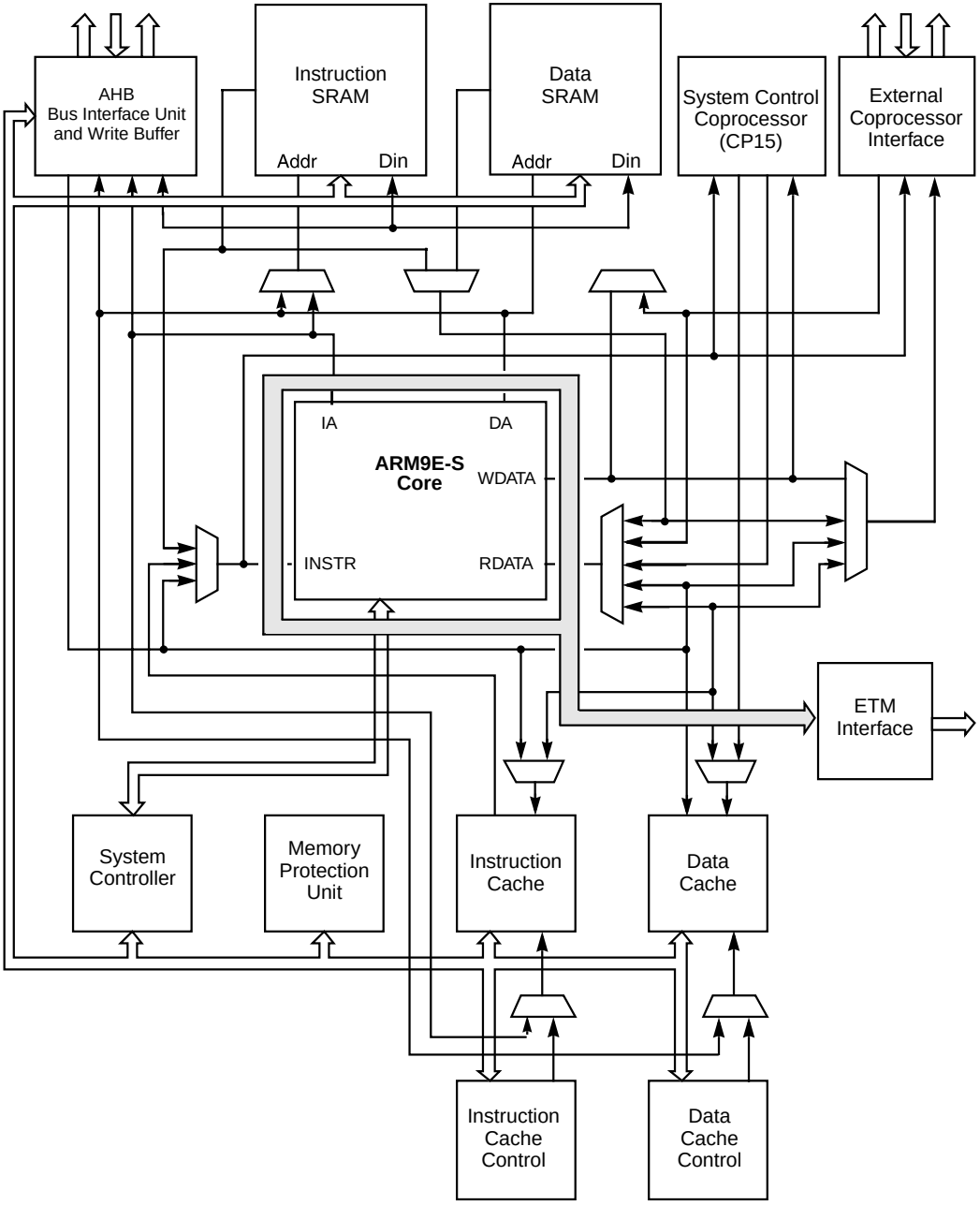
- Tightly coupled instruction and data SRAM interfaces. You can configure the size of the instruction and data SRAMs to suit the needs of your design implementation.
- Instruction and data caches. You can easily modify your design to allow any combination of caches from 4 Kbytes to 1 Mbyte.
- Protection Unit. This unit allows you to segment and protect memory in a simple manner. This feature is ideal for embedded control applications.
- AMBA AHB bus interface. ARM946E-S uses unified address and data buses to interface to the rest of the system. This interface is compatible with the AMBA AHB bus standard.
- External coprocessor support. This capability allows the addition of floating-point or other application-specific hardware accelerators. For coprocessor support, the instruction and data buses are exported along with simple handshaking signals.
- Scan testing and *Built-In Self-Test* (BIST) support. This feature provides scan test capability for the standard cell logic and BIST for the tightly coupled SRAM and caches.
- Interface to an external *Embedded Trace Macrocell* (ETM). The ETM provides support for real-time tracing of instructions and data.

Providing this complete high-frequency subsystem allows system-on-a-chip designers to concentrate on design issues unique to their system. The synthesizable nature of the device eases integration into ASIC technologies.

1.2 Microprocessor Block Diagram

The ARM946E-S block diagram is shown in Figure 1.1. A brief description of each block follows the diagram.

Figure 1.1 ARM946E-S Block Diagram



1.2.1 ARM9E-S Processor Core

The ARM9E-S processor core has a Harvard bus architecture with separate instruction and data interfaces. This design allows concurrent instruction and data accesses, and greatly reduces the cycles per instruction of the processor. For optimal performance, single-cycle memory accesses for both interfaces are required, although the core can be stalled for nonsequential accesses or slower memory systems.

The processor is implemented using a five-stage pipeline:

- Instruction Fetch (F)
- Instruction Decode (D)
- Execute (E)
- Data Memory Access (M)
- Register Write (W)

ARM implementations are fully interlocked, so that software functions identically across different implementations without concern for pipeline effects.

Refer to the *ARM9E-S Technical Reference Manual* for more information about the processor core.

1.2.2 System Controller

The system controller oversees the interactions between the Instruction RAM, Data RAM, and the Bus Interface Unit. It controls internal arbitration between the blocks and stalls the appropriate blocks when required.

1.2.3 CP15 System Control Coprocessor

The processor core uses a set of registers in the CP15 Coprocessor to control the functionality of the RAMs and the Write Buffer. These registers are accessed using the coprocessor instructions MCR and MRC.

Refer to Chapter 3 for more information about CP15.

1.2.4 Data and Instruction Caches and Control

The ARM946E-S has separate data and instruction caches. Each cache is direct mapped, or either 2-way or 4-way set associative. The D-cache and I-cache use a physical address from the processor core, and have a cache update policy of *allocate on a read miss*. The D-cache and I-cache are reloaded one cache line (eight words) at a time through the external interface.

Refer to Chapter 4 for more information about the data and instruction caches.

1.2.5 Protection Unit

The Protection Unit makes it possible to partition memory into eight regions of variable size and to set individual attributes for each memory region.

Refer to Chapter 5 for more information about the Protection Unit.

1.2.6 Instruction and Data SRAMs

The ARM946E-S incorporates internal instruction and data memories to allow high-speed operation without incurring the performance penalties of accessing the system bus. The Instruction and Data RAMs each consist of blocks of ASIC library compiled RAM. The RAM sizes can be of any size up to 1 Mbyte. The instruction and data memories can have different sizes.

Refer to Chapter 6 for more information about the Instruction and Data RAMS.

1.2.7 AHB Interface Unit and Write Buffer

The Advanced High-Performance Bus (AHB) is a new generation of AMBA bus, which meets the requirements of high-performance synthesizable designs. The AHB Interface Unit arbitrates between the sources of external bus transactions within the ARM946E-S. It stalls all other accesses until the current request has been completed.

The Write Buffer is a 16-entry FIFO. It increases system performance.

Refer to Chapter 7 for more information about the AHB interface and Write Buffer.

1.2.8 External Coprocessor Interface

The ARM946E-S supports the connection of coprocessors through the external coprocessor interface. All types of ARM coprocessor instructions are supported. Coprocessors determine the instructions they need to execute using a pipeline follower in the coprocessor.

Refer to Chapter 8 for more information about the External Coprocessor Interface.

1.2.9 JTAG and Debug Interface Port

The JTAG and Debug Interface (not shown in Figure 1.1) is based on IEEE Standard 1149.1-1990. The interface makes it possible to stop the processor core on a given instruction fetch (breakpoint), data access (watchpoint), or external debug request. The JTAG interface allows serial insertion of instructions into the pipeline of the core without using the external data bus.

Refer to Chapter 9 for more information about the JTAG and Debug Interface.

1.2.10 Embedded Trace Module Interface

This interface connects to an external Embedded Trace Module (ETM). The ETM provides a high-speed port for tracing the processor core in real time.

Refer to Chapter 10 for more information about the ETM interface.

1.3 CoreWare® Program

The CoreWare program consists of three main elements:

1. A library of cores
2. A design development and simulation package
3. Expert applications support

The CoreWare library contains a wide range of complex cores based on accepted and emerging industry standards from high-speed interconnect and digital video to DSP and microprocessors. LSI Logic provides a complete framework for device and system development and simulation. LSI Logic has advanced ASIC technologies that consistently produce Right-First-Time™ silicon. The LSI Logic in-house experts provide design support from system architecture definition through chip layout and test vector generation.

Chapter 2
Signal Descriptions

This chapter describes the ARM946E-S microprocessor signals. It contains the following sections:

2.1 Signal Properties and Requirements

The following design features ensure easier integration of the ARM946E-S into embedded applications and simplify synthesis flow:

- A single rising edge clock times all activity
- All signals and buses are unidirectional
- All inputs are required to be synchronous to the single clock

These features simplify the definition of the top-level ARM946E-S signals, because all outputs change from the rising edge and all inputs are sampled on the rising edge of the clock. In addition, all signals are either input or output only; bidirectional signals are not used.

Note: You must use external logic to synchronize asynchronous signals (for example, interrupt sources) before applying them to the ARM946E-S macrocell.

Figure 2.1 ARM946E-S Signal Diagram

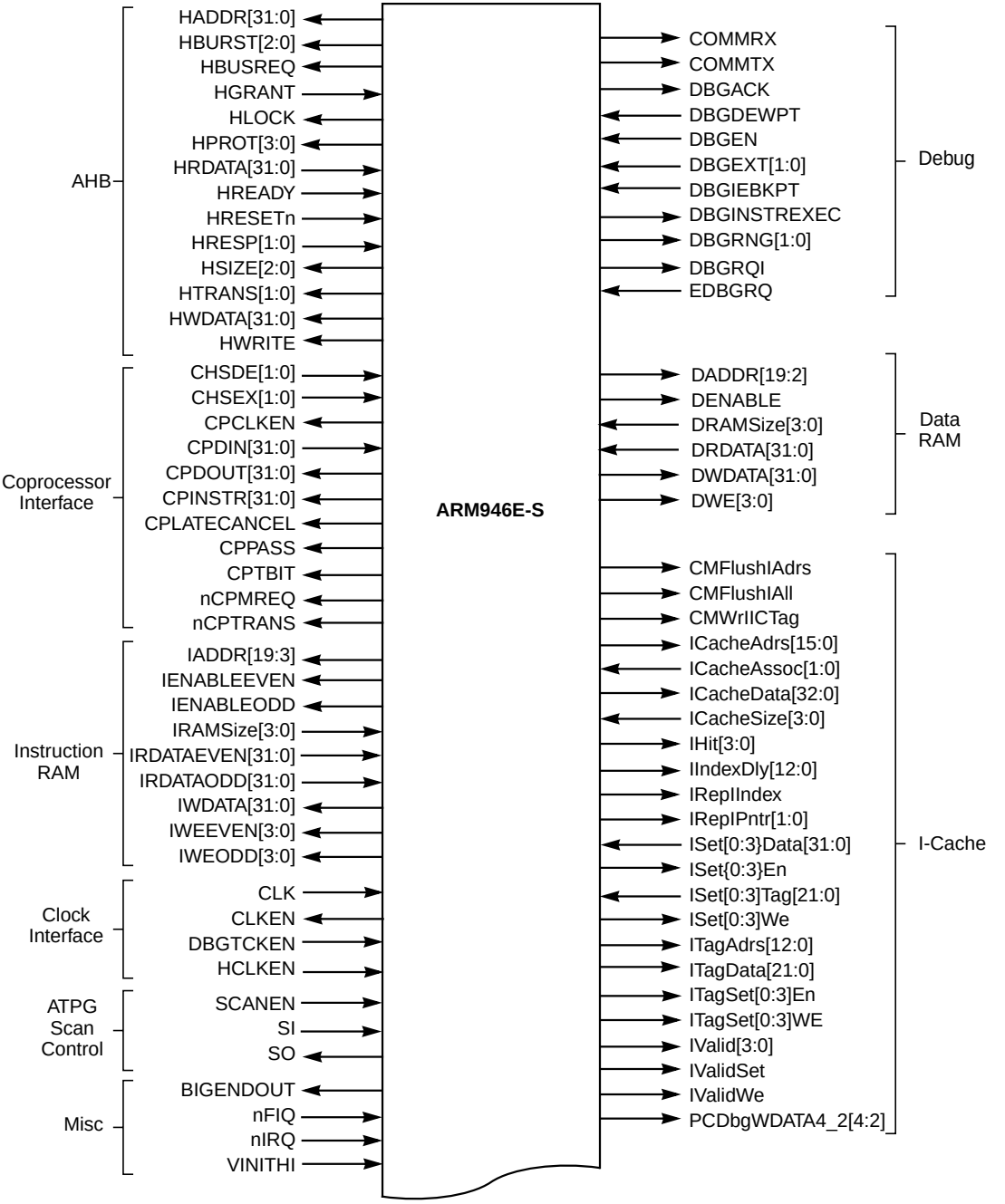
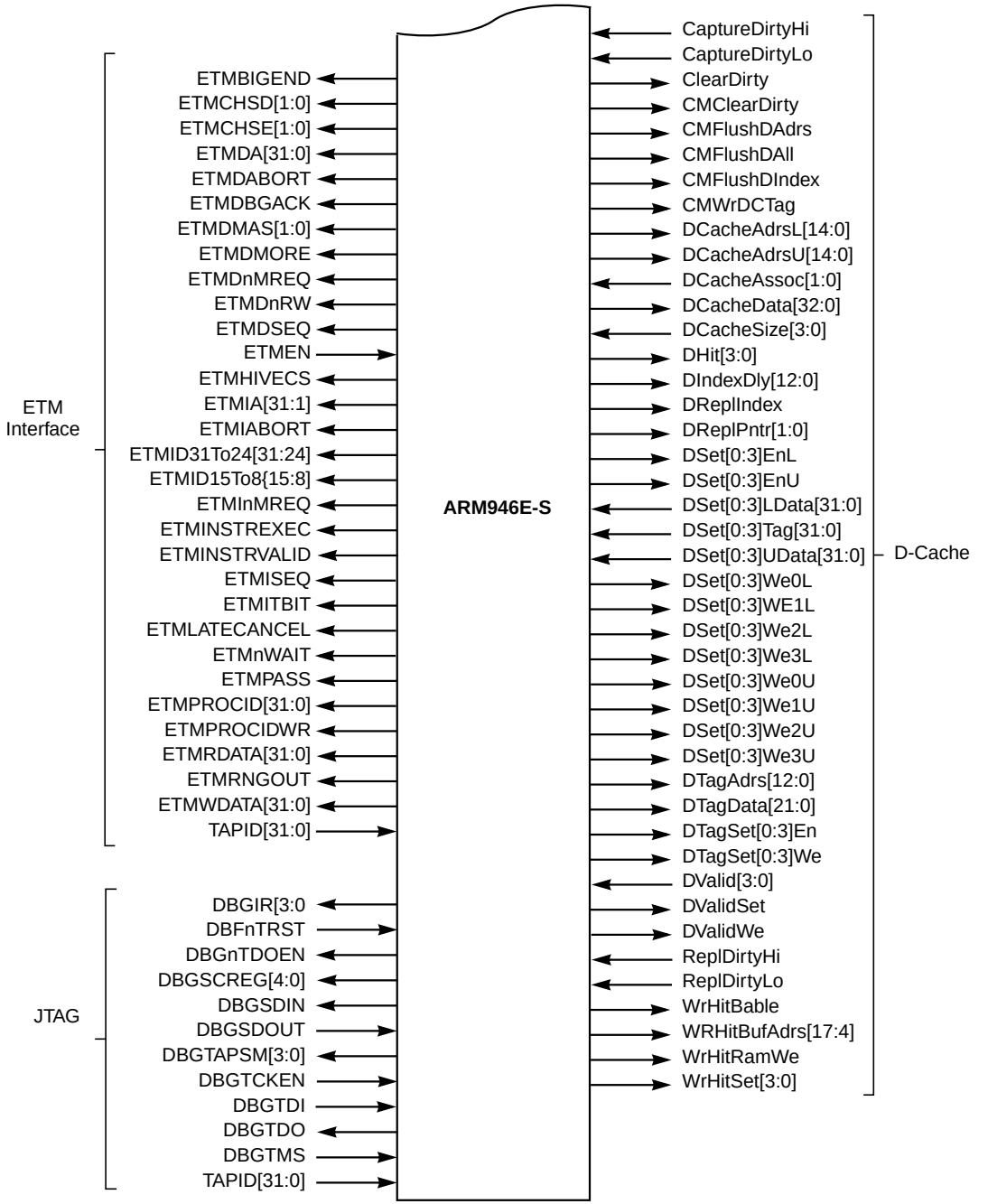


Figure 2.1 ARM946E-S Signal Diagram (cont.)



2.2 Clock Interface Signals

The following information describes the ARM946E-S clock interface signals.

CLK **System Clock** **Input**
This clock times all operations in the ARM946E-S core. All outputs change from the rising edge, and all inputs are sampled on the rising edge. The clock can be stretched in either phase.

When HCLKEN is HIGH, CLK also times AHB operations.

When DBGTCKEN is HIGH, CLK also times debug operations.

CLKEN **System Clock Enable** **Output**
CLKEN is the ARM946E-S system clock enable. CLKEN goes LOW to indicate a stall condition.

DBGTCKEN **JTAG Debug Logic Enable** **Input**
This signal provides a synchronous enable for debug logic accessed using the JTAG interface. When DBGTCKEN is HIGH, the debug logic can advance on the rising edge of CLK.

HCLKEN **AHB Clock Enable** **Input**
This signal provides a synchronous enable for AHB transfers. When HIGH, it indicates that the next rising edge of CLK is also a rising edge of HCLK in the AHB system where the ARM946E-S is embedded. HCLKEN must be tied HIGH in systems where CLK and HCLK are the same frequency.

2.3 AHB Signals

The following information describes the ARM946E-S AHB signals.

HADDR[31:0] **Address Bus** **Output**
HADDR[31:0] is the 32-bit AHB system address bus.

HBURST[2:0] Burst Type **Output**

This output indicates whether or not the transfer forms part of a burst. Both four beat and eight beat bursts are supported, where a beat is a clock tick. The ARM946E-S generates only incrementing bursts, because cache fills are zero-word first.

HBURST[2:0]	Burst Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
011	INCR4	4-beat incrementing burst
101	INCR8	8-beat incrementing burst

HBUSREQ Bus Request **Output**

When asserted, this signal indicates that the ARM946E-S requires the bus.

HGRANT Bus Grant **Input**

When asserted, this signal indicates that the ARM946E-S is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when HREADY is HIGH. When both HREADY and HGRANT are HIGH, the ARM946E-S gets access to the bus.

HLOCK Request Locked Transfers **Output**

When HIGH, this signal indicates that the ARM946E-S requires locked access to the bus and no other master should be granted access until this signal has gone LOW. The ARM946E-S asserts HLOCK when executing SWP instructions to AHB address space.

HPROT[3:0] Protection Control **Output**

This output provides information about a bus access that is useful to modules that implement some level of protection.

The HPROT[3:0] signals provide the information shown below.

HPROT3 Cacheable	HPROT2 Bufferable	HPROT1 Supervisor	HPROT0 Data/Opcode	Description
–	–	–	0	Opcode Fetch
–	–	–	1	Data Access
–	–	0	–	User Access
–	–	1	–	Supervisor Access
–	0	–	–	Not Bufferable
–	1	–	–	Bufferable
0	–	–	–	Not Cacheable
1	–	–	–	Cacheable

HRDATA[31:0]
Read Data Bus **Input**
During read operations, this 32-bit bus transfers data from a selected bus slave to the ARM946E-S.

HREADY **Transfer Done** **Input**
When HIGH, this signal indicates that a bus transfer has finished. A selected bus slave can drive this signal LOW to extend a transfer.

HRESETn **Not Reset** **Input**
This signal must be asynchronously asserted LOW to initialize the ARM946E-S system state. It must be deasserted synchronously.

HRESP[1:0] **Transfer Response** **Input**
These signals contain a transfer response from the selected slave. They provide additional status transfer information. The responses are shown below:

HRESP[1:0]	Description
0b00	Okay
0b01	Error
0b10	Retry
0b11	Split

HSIZE[2:0] **Transfer Size** **Output**

These signals indicate the size of an ARM946E-S transfer. Bit 2 is tied LOW. The transfer sizes are:

HSIZE[2:0]	Transfer Size
0b000	Byte
0b001	Halfword
0b010	Word

HTRANS[1:0] **Transfer Type** **Output**

These signals indicate the ARM946E-S transfer type. The transfer types are:

HTRANS[1:0]	Transfer Type
0b00	Idle
0b01	Nonsequential
0b10	Sequential

HWDATA[31:0]

Write Data Bus **Output**

This 32-bit bus transfers data from the ARM946E-S to a selected bus slave during write operations.

HWRITE **Transfer Direction** **Output**

When HWRITE is HIGH, it indicates this is a write transfer. When the signal is LOW, it indicates this is a read transfer.

2.4 Instruction RAM Signals

The instruction RAM is split into two banks, even and odd. Both banks get the same address and write data, but they have separate write enables so only one bank is written at a time. The muxing of the data read from the RAMs is performed inside the core.

IADDR[19:3] **Instruction RAM Address** **Output**

This 17-bit bus contains the Instruction RAM address and handles address for a 1 Mbyte address range. Addressing is performed on word boundaries, so bits 0 and 1 are not needed. Bit 2 is replaced by the even and odd select signals.

IENABLEEVEN

Word-Based Instruction Chip Enable Even **Output**

Driving this signal LOW disables the clock on the even instruction RAM and saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

IENABLEODD

Word-Based Instruction Chip Enable Odd **Output**

Driving this signal LOW disables the clock on the odd Instruction RAM.

IRAMSize[3:0]

Instruction RAM Size **Input**

These signals specify the size of the Instruction RAM.

IRAMSize[3:0]	RAM Size
0b0000	0 Kbyte
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

IRDATAEVEN[31:0]

Instruction RAM Read Data Even **Input**

This 32-bit bus contains data read from the even Instruction RAM.

IRDATAODD[31:0]

Instruction RAM Read Data Odd **Input**

This 32-bit bus contains data read from the odd Instruction RAM.

IWDATA[31:0] **Instruction RAM Write Data** **Output**

This 32-bit bus contains write data for the Instruction RAM.

IWEEVEN[3:0]

Byte-Based Instruction Write Enable Even **Output**

These signals are byte write enables. Asserting any one of the IWEEVEN[3:0] signals HIGH enables a write to the corresponding data byte in the even Instruction RAM.

IWEEVEN	Enables Data Bits
3	31:24
2	23:16
1	15:8
0	7:0

IWEODD[3:0] **Byte-Based Instruction Write Enable Odd** **Output**

These signals are byte write enables. Asserting one of the IWEODD[3:0] signals HIGH enables a write to the corresponding data byte in the odd Instruction RAM.

IWEODD	Enables Data Bits
3	31:24
2	23:16
1	15:8
0	7:0

2.5 Data RAM Signals

The data RAM is one logical bank of memory.

DADDR[19:2] **Data RAM Address** **Output**

This 18-bit bus contains the Data RAM address. Addressing is performed on word boundaries, so bits 0 and 1 are not needed. Bits [19:2] address a 1 Mbyte address space.

DENABLE **Word-Based Data Chip Enable** **Output**

Driving this signal LOW disables the Data RAM clock and saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

DRAMSize[3:0]

Data RAM Size **Input**

These signals specify the Data RAM size.

DRAMSize[3:0]	
0b0000	0 Kbyte
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

DRDATA[31:0]

Data RAM Read Data **Input**

This 32-bit bus contains data read from the Data RAM.

DWDATA[31:0]

Data RAM Write Data **Output**

This 32-bit bus contains write data for the Data RAM.

DWE[3:0] **Byte-Based Data Write Enable** **Output**

These bits are byte write enable signals. Asserting one of these signals HIGH enables writes to the corresponding bits in the Data RAM.

DWE	Enable Write to Data Bits
3	31:24
2	23:16
1	15:8
0	7:0

2.6 Instruction Cache Signals

The instruction cache signals support 1, 2, or 4 sets of RAMs. Each set has an Instruction RAM that contains the instructions and a tag RAM that

holds the address information needed for hit detection. There are also signal connections to the A946ESIValid module, which contains the valid bits for the instruction cache.

CMFlushIAdrs

Cache Maintenance Flush Inst. Cache Adrs. Output
Asserting this signal HIGH resets the valid bit associated with the instruciton cache address if the address presented by the CPU is a hit.

CMFlushIAll **Cache Maintenance Flush All Inst. Cache Output**
Asserting this signal HIGH synchronously resets all the instruction cache valid bits.

CMWrICTag **Cache Maintenance Write Inst. Cache Tag Output**
Asserting this signal HIGH writes the value in bit 4 of the PCDbgWDATA4_2 signal into the valid bit selected by the IIndexDly and IReplPntr signals.

ICacheAdrs[15:0]

Instruction Cache Address Output
These 16 bits hold the instruction RAM address, which is a word address that allows using RAMs that hold up to 256 Kbytes.

ICacheAssoc[1:0]

Instruction Cache Associativity Input
These bits specify the associativity of the instruction cache.

ICacheAssoc[1:0]	Encoding
00	Direct Mapped
01	Two Way
10	Four Way

ICacheData[32:0]

Instruction Cache Data Output
This bus contains data to be written to the instruction set RAMs.

ICacheSize[3:0]

Instruction Cache Size **Input**

These bits specify the size of the instruction cache.

ICacheSize[3:0]	Encoding
0b0000	0 Kbyte
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

IHit[3:0] **Instruction Cache Hit** **Output**

Each IHit[3:0] bit corresponds to instruction cache set [3:0]. When there is a hit on an instruction cache set, the corresponding IHIT[3:0] bit is driven HIGH.

IIndexDly[12:0]

Instruction Cache Index Delayed **Output**

These bits are a delayed version of the index used to select valid bits.

IReplIndex **Instruction Cache Replacement Index** **Output**

This bit is the index of a line that is being replaced.

IReplPntr[1:0] **Instruction Cache Replacement Pointer** **Output**

These bits point to the instruction cache set that will be affected when a Cache Maintenance operation uses the index/set addressing mode.

ISet0Data[31:0], ISet1Data[31:0], ISet2Data[31:0], ISet3Data[31:0]

Instruction Set 0, 1, 2, 3 Data **Input**

This 32-bit bus contains data read from the instruction cache Set 0, 1, 2, or 3 RAM.

ISet0En, ISet1En, ISet2En, ISet3En

Instruction Set 0, 1, 2, 3 Enable **Output**

Driving this signal HIGH enables the clock on the Set 0, 1, 2, or 3 instruction RAM. Driving it LOW disables the

clock, which saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

ISet0Tag[21:0] , ISet1Tag[21:0], ISet2Tag[21:0], ISet3Tag[21:0]
Instruction Set 0, 1, 2, 3 Tag **Input**
These bits contains data read from the instruction cache Set 0, 1, 2, or 3 Tag RAM.

ISet0We, ISet1We, ISet2We, ISet3We
Instruction Set 0, 1, 2, 3 Write Enable **Output**
Asserting this signal HIGH enables word writes to the Set 0, 1, 2, or 3 instruction RAM.

ITagAdrs[12:0]
Instruction Cache Tag Address **Output**
This 13-bit bus contains the address for the instruction tag RAMs.

ITagData[21:0]
Instruction Cache Tag Data **Output**
This bus contains write data for the instruction tag RAMs.

ITagSet0En, ITagSet1En, ITagSet2En, ITagSet3En
Instruction Tag Set 0, 1, 2, 3 Enable **Output**
Driving this signal HIGH enables the clock on the Set 0, 1, 2, or 3 Tag RAM. Driving it LOW disables the clock, which saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

ITagSet0We, ITagSet1We, ITagSet2We, ITagSet3We
Instruction Tag Set 0, 1, 2, 3 Word Write Enable **Output**
Asserting this signal HIGH enables word writes to the Set 0, 1, 2, or 3 Tag RAM.

IValid[3:0] **Instruction Cache Valid bits** **Input**
These bits are the instruction cache valid bits. There is one valid bit for each instruction cache set. Individual valid bits are selected by the index portion of the cache address.

IValidSet **Instruction Cache Valid Set** **Output**
This signal is driven HIGH to set the selected valid bit, and it is driven LOW to clear the valid bit. It is used when a new line is loaded into the instruction cache.

IValidWe **Instruction Cache Valid Write Enable** **Output**
This signal is the write enable for the valid bits. It is used when a new line is loaded into the instruction cache.

PCDbgWDATA4_2[4:2]
Prot/CP15 to Cache Debug Write Data **Output**
These bits write the valid bit on the instruction cache and the valid and dirty bits on the data cache during a cache maintenance operation.

2.7 Data Cache Signals

The data cache signals support 1, 2, or 4 RAM sets. Each set has an upper and lower data RAM and one tag RAM. The data RAMs hold data, and the tag RAMs hold the address information needed for a hit detection. In addition, there are signals that provide an interface to the A946ESDValid and A946ESDDirty modules. These modules contain the data cache valid and dirty bits, respectively.

CaptureDirtyHi
Capture Dirty Bit High **Input**
This is the ReplDirtyHi signal delayed by one clock. The cache maintenance state machine uses this signal.

CaptureDirtyLo
Capture Dirty Bit Low **Input**
This is the ReplDirtyLo signal delayed by one clock. The cache maintenance state machine uses this signal.

ClearDirty **Clear Dirty Bit** **Output**
Asserting this signal clears the dirty bits for a line after a new line replaces it.

CMClearDirty **Cache Maintenance Clear Dirty Bit** **Output**
This signal is a one-cycle pulse that clears the dirty bits in the addressed line.

CMFlushDAdrs

Cache Maintenance Flush Data Cache Adrs **Output**
Asserting this signal HIGH resets a valid bit if the address presented by the CPU is a hit.

CMFlushDAll

Cache Maintenance Flush All Data Cache **Output**
Asserting this signal HIGH synchronously resets all the data cache valid bits.

CMFlushDIndex

Cache Maintenance Flush Data Cache Index **Output**
Asserting this signal HIGH resets the valid bit pointed to by the Index.

CMWrDCTag

Cache Maintenance Write Data Cache Tag **Output**
Asserting this signal HIGH sets the valid bit pointed to by the Index.

DCacheAdrsL[14:0]

Data Cache Address Lower **Output**
These bits contain the word address for the lower data set RAMs. This word address range allows the use of RAMs that are up to 128 Kbytes in size.

DCacheAdrsU[14:0]

Data Cache Address Upper **Output**
These bits contain the word address for the upper data set RAMs. This word address range allows the use of RAMs that are up to 128 Kbytes in size.

DCacheAssoc[1:0]

Data Cache Associativity **Input**
These bits specify the associativity of the data cache.

DCacheAssoc[1:0]	Encoding
0b00	Direct Mapped
0b01	Two Way
0b10	Four Way

DCacheData[32:0]

Data Cache Data **Output**
This bus contains data to be written to the data set RAMs.

DCacheSize[3:0]

Data Cache Size **Input**

These bits specify the size of the data cache.

DCacheSize[3:0]	Encoding
0b0000	0 Kbyte
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Kbytes

DHit[3:0] **Data Cache Hit** **Output**

Each DHit[3:0] bit corresponds to data cache set [3:0]. When there is a hit on an data cache set, the corresponding DHit[3:0] bit is driven HIGH.

DIndexDly[12:0]

Data Cache Index Delayed **Output**

These bits are a delayed version of the index used to select valid bits.

DReplIndex **Data Cache Replacement Index** **Output**

This bit is the index of the line being replaced.

DReplPntr[1:0]

Data Cache Replacement Pointer **Output**

These bits point to the data cache set that is affected when a Cache Maintenance operation uses the index/set addressing mode.

DSet0EnL, DSet1EnL, DSet2EnL, DSet3EnL

Data Set 0, 1, 2, 3 Lower Enable **Output**

Driving this signal HIGH enables the clock on the Set 0, 1, 2, or 3 Lower RAM. Driving it LOW disables the clock, which saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

DSet0EnU, DSet1EnU, DSet2EnU, DSet3EnU

Data Set 0, 1, 2, 3 Upper Enable **Output**

Driving this signal HIGH enables the clock on the Set 0, 1, 2, or 3 Upper RAM. Driving it LOW disables the clock, which saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

DSet0LData[31:0], DSet1LData[31:0], DSet2LData[31:0], DSet3LData[31:0]

Data Set 0, 1, 2, 3 Lower Data **Input**

This 32-bit bus contains data read from the lower data cache Set 0, 1, 2, or 3 RAM.

DSet0Tag[31:0],DSet1Tag[31:0], DSet2Tag[31:0], DSet3Tag[31:0]

Data Set 0, 1, 2, or 3 Tag **Input**

This 32-bit bus contains data read from the data cache Set 0, 1, 2, or 3 Tag RAM.

DSet0UData[31:0], DSet1UData[31:0], DSet2UData[31:0], DSet3UData[31:0]

Data Set 0, 1, 2, 3 Upper Data **Input**

This 32-bit bus contains data read from the upper data cache Set 0, 1, 2. or 3 RAM.

DSet0We0L, DSet1We0L, DSet2We0L, DSet3We0L

Data Set 0, 1, 2, 3 Write Enable Byte 0, Lower **Output**

Asserting this signal HIGH enables writing to data bits [7:0] in the Lower RAM of data set 0, 1, 2, or 3.

DSet0We1L, DSet1We1L, DSet2We1L, DSet3We1L

Data Set 0, 1, 2, 3 Write Enable Byte 1, Lower **Output**

Asserting this signal HIGH enables writing to data bits [15:8] in the Lower RAM of data set 0, 1, 2, or 3.

DSet0We2L, DSet1We2L, DSet2We2L, DSet3We2L

Data Set 0, 1, 2, 3 Write Enable Byte 2, Lower **Output**

Asserting this signal HIGH enables writing to data bits [23:16] in the Lower RAM of data set 0, 1, 2, or 3.

DSet0We3L, DSet1We3L, DSet2We3L, DSet3We3L

Data Set 0, 1, 2, 3 Write Enable Byte 3, Lower **Output**

Asserting this signal HIGH enables writing to data bits [31:24] in the Lower RAM of data set 0, 1, 2, or 3.

DSet0We0U, DSet1We0U, DSet2We0U, DSet3We0U
Data Set 0, 1, 2, 3 Write Enable Byte 0, Upper **Output**
Asserting this signal HIGH enables writing to data bits [7:0] in the Upper RAM of data set 0, 1, 2, or 3.

DSet0We1U, DSet1We1U, DSet2We1U, DSet3We1U
Data Set 0, 1, 2, 3 Write Enable Byte 1, Upper **Output**
Asserting this signal HIGH enables writing to data bits [15:8] in the Upper RAM of data set 0, 1, 2, or 3.

DSet0We2U, DSet1We2U, DSet2We2U, DSet3We2U
Data Set 0, 1, 2, 3 Write Enable Byte 2, Upper **Output**
Asserting this signal HIGH enables writing to data bits [23:16] in the Upper RAM of data set 0, 1, 2, or 3.

DSet0We3U, DSet1We3U, DSet2We3U, DSet3We3U
Data Set 0, 1, 2, 3 Write Enable Byte 3, Upper **Output**
Asserting this signal HIGH enables writing to data bits [31:24] in the Upper RAM of data set 0, 1, 2, or 3.

DTagAdrs[12:0]
Data Cache Tag Address **Output**
This bus contains the address for the data tag RAMs.

DTagData[21:0]
Data Cache Tag Data **Output**
This bus contains data to be written to the data tag RAMs.

DTagSet0En, DTagSet1En, DTagSet2En, DTagSet3En,
Data Set 0, 1, 2, 3 Tag Enable **Output**
Driving this signal HIGH enables the clock on the Set 0, 1, 2, or 3 Tag RAM. Driving it LOW disables the clock, which saves power. The RAMs do not have to connect to this pin. The function is not changed if the RAMs ignore the enable and clock every cycle.

DTagSet0We, DTagSet1We, DTagSet2We, DTagSet3We,
Data Set 0, 1, 2, 3 Tag Write Enable **Output**
Asserting this signal HIGH enables writing to the Set 0, 1, 2, or 3 Tag RAM.

DValid[3:0] **Data Cache Valid Bits** **Input**
These bits are the data cache valid bits. There is one valid bit for each data cache set. Individual valid bits are selected by the index portion of the cache address.

DValidSet **Data Cache Valid Set** **Output**
Driving this signal HIGH sets the selected valid bit, and driving it LOW clears the valid bit. It is used when a new line is loaded into cache.

DValidWe **Data Cache Valid Write Enable** **Output**
This signal is the valid bit write enable. It is used when a new line is loaded into the data cache.

ReplDirtyHi **Replacement Dirty Bit High** **Input**
This signal controls the dirty bit for the upper half of the line being replaced.

ReplDirtyLo **Replacement Dirty Bit Low** **Input**
This signal controls the dirty bit for the lower half of the line being replaced.

WrHitBable **Write Hit Bufferable** **Output**
This signal is driven HIGH for write back, and it is driven LOW for write through. The dirty bits are set in write back mode only.

WrHitBufAdrs[17:4]
Write Hit Buffer Address **Output**
These bits contain the address of the write hit buffer.

WrHitRamWe
Write Hit RAM Write Enable **Output**
This signal is driven HIGH for a data cache write. Note that the dirty bits are set on a write hit.

WrHitSet[3:0]
Write Hit Set **Output**
Each WrHitSet[3:0] bit corresponds to a data cache set. When one of these bits is HIGH, the associated data cache dirty bit will set on a write hit.

2.8 Coprocessor Interface Signals

This section describes the ARM946E-S coprocessor interface signals.

CHSDE[1:0] **Coprocessor Handshake Decode** **Input**

These inputs are the handshake signals from the decode stage of the coprocessor pipeline follower.

CHSDE[1:0]	Encoding
00	Wait
01	Go
10	Absent
11	Last

CHSEX[1:0] **Coprocessor Handshake Execute** **Input**

These inputs are the handshake signals from the execute stage of the coprocessor pipeline follower.

CHSEX[1:0]	Encoding
00	Wait
01	Go
10	Absent
11	Last

CPCLKEN **Coprocessor Clock Enable** **Output**

This signal provides a synchronous enable for the coprocessor pipeline follower. When CPCLKEN is HIGH, the pipeline follower logic advances on the rising edge of CLK.

CPDIN[31:0] **Coprocessor Write Data** **Input**

This 32-bit bus is the coprocessor write data bus for transferring data from the coprocessor.

CPDOUT[31:0]

Coprocessor Read Data **Output**

This 32-bit bus is the coprocessor read data bus for transferring data to the coprocessor.

CPINSTR[31:0]

Coprocessor Instruction Data **Output**

This 32-bit bus is the coprocessor instruction data bus for transferring instructions to the coprocessor pipeline follower.

CPLATECANCEL **Output**

If this signal is HIGH during the first memory cycle of a coprocessor instruction, the coprocessor must cancel the

instruction without changing any internal state. This signal is only asserted in cycles when the previous instruction caused a data abort.

CPPASS **Coprocessor Pass** **Output**
When HIGH, this signal indicates that there is a coprocessor instruction in the execute stage of the pipeline that must be executed.

CPTBIT **Coprocessor Instruction Thumb Bit** **Output**
When CPTBIT is HIGH, the ARM946E-S is in the Thumb state. When this signal is LOW, the ARM946E-S is in the ARM state. The coprocessor pipeline follower samples this signal.

nCPMREQ **Not Coprocessor Instruction Request** **Output**
When this signal is LOW and CPCLKEN is HIGH on the rising edge of CLK, then the instruction on the CPINSTR[31:0] data bus must enter the coprocessor pipeline.

nCPTRANS **Not Coprocessor Memory Translate** **Output**
When this signal is LOW, the ARM946E-S is in User mode. When the signal is HIGH, the ARM946E-S is in Privileged mode. The coprocessor pipeline follower samples this signal.

2.9 Debug Signals

The following information describes the ARM946E-S debug signals.

COMMRX **Communications Channel Receive** **Output**
When HIGH, this signal indicates that the communications channel receive buffer contains valid data that is waiting to be read.

COMMTX **Communications Channel Transmit** **Output**
When HIGH, this signal indicates that the communications channel transmit buffer is empty.

DBGACK **Debug Acknowledge** **Output**
When HIGH, this signal indicates that the processor is in debug state.

DBGDEWPT **Data Watchpoint** **Input**
External hardware asserts this signal to halt execution of the processor for debug purposes. If this signal is HIGH at the end of a data memory request cycle, it causes the ARM946E-S to enter debug state.

DBGEN **Debug Enable** **Input**
When HIGH, this signal enables the debug features of the processor. Tie this signal LOW if debug is not required.

DBGEXT[1:0] **EmbeddedICE-RT External Input** **Input**
These inputs to the EmbeddedICE-RT logic make breakpoints/watchpoints dependent on external conditions.

DBGIEBKPT **Instruction Breakpoint** **Input**
External hardware asserts this signal to halt execution of the processor for debug purposes. If this signal is HIGH at the end of an instruction fetch, it causes the ARM946E-S to enter the debug state when the instruction reaches the execute stage of the processor pipeline.

DBGINSTREXEC
Instruction Executed **Output**
When this signal is HIGH, it indicates the processor executed the instruction in the execute stage of the processor pipeline.

DBGRNG[1:0] **EmbeddedICE-RT Range Out** **Output**
These signals indicate that the corresponding EmbeddedICE-RT watchpoint register matches the conditions currently present on the address, data, and control buses. These signals are independent of the state of the watchpoint enable control bit.

DBGRQI **Internal Debug Request** **Output**
This signal is the debug request that is presented to the core debug logic. It is a combination of EDBGRQ and bit 1 of the Debug Control Register.

EDBGRQ **External Debug Request** **Input**
An external debugger asserts this signal to force the processor to enter the debug state.

2.10 JTAG Signals

The following information describes the ARM946E-S JTAG signals.

DBGIR[3:0] **TAP Controller Instruction Register** **Output**
These four bits reflect the current instruction loaded in the TAP controller instruction register. They change when the TAP controller is in the UPDATE-IR state.

DBGnTRST **Not Test Reset** **Input**
This active-LOW input is the internally synchronized reset signal for the EmbeddedICE-RT internal state.

DBGnTDOEN **Not DBGTDO Enable** **Output**
When LOW, this signal indicates that there is serial data on the DBGTDO output. Normally, this signal is used as an output enable for a DBGTDO pin in a packaged part.

DBGSCREG[4:0]
Scan Chain ID **Output**
These five bits reflect the ID number of the scan chain currently selected by the TAP controller. They change when the TAP controller is in the UPDATE-DR state.

DBGSDIN **External Scan Chain Serial Input Data** **Output**
This output contains the serial data for an external scan chain.

DBGSDOUT **External Scan Chain Serial Data Output** **Input**
DBGSDOUT contains serial data from an external scan chain. Tie this signal LOW when an external scan chain is not connected.

DBGTAPSM[3:0]
TAP Controller State Machine **Output**
This bus reflects the current state of the TAP controller state machine.

DBGTCKEN **Test Clock Enable** **Input**
This signal is the synchronous enable test clock.

DBGTDI **Test Data In** **Input**
This signal is the test data input to the debug logic.

DBGTDO **Test Data Out** **Output**
This signal is the test data output from the debug logic.

DBGTMS **Test Mode Select** **Input**
This signal is the TAP controller test mode select.

TAPID[31:0] **Boundary Scan ID Code** **Input**
These signals specify the ID code value shifted out on DBGTDO when the IDCODE instruction is entered in the TAP controller.

2.11 Miscellaneous Signals

This section describes miscellaneous ARM946E-S signals.

BIGENDOUT **Big Endian** **Output**
When this signal is HIGH, the ARM946E-S handles memory data bytes using the big-endian format. When LOW, memory data is handled as little endian.

nFIQ **Not Fast Interrupt Request** **Input**
When an external source drives this signal LOW, it causes a Fast Interrupt Request (FIQ) exception in the processor. nFIQ must be synchronous with CLK.

nIRQ **Not Interrupt Request** **Input**
When an external source drives this signal LOW, it causes a normal Interrupt Request (IRQ) exception in the processor. nIRQ must be synchronous with CLK.

VINITHI **Exception Vector Location at Reset** **Input**
This signal determines the reset location of the exception vectors. When VINITHI is LOW, the vectors are located at 0x00000000. When it is HIGH, the vectors are located at 0xFFFF0000.

2.12 ETM Interface Signals

This section describes the ARM946E-S ETM interface signals.

ETMBIGEND **Endian Mode** **Output**

This output indicates the endian mode for the ETM. When this signal is HIGH, the mode is big endian; when ETMBIGEND is LOW, the mode is little endian.

ETMCHSD[1:0]

ETM Coprocessor Handshake Decode **Output**

These outputs are the handshake signals from the decode stage of the coprocessor's pipeline follower.

ETMCHSD[1:0]	Encoding
10	ABSENT
00	WAIT
01	GO
11	LAST

ETMCHSE[1:0]

ETM Coprocessor Handshake Execute **Output**

These outputs are the handshake signals from the execute stage of the coprocessor's pipeline follower.

ETMCHSE[1:0]	Encoding
10	ABSENT
00	WAIT
01	GO
11	LAST

ETMDA[31:0] **ETM Data Address** **Output**

This 32-bit bus contains the ETM data address.

ETMDABORT **ETM Data Abort** **Output**

Assertion of this signal indicates a data abort to the ARM9E-S core.

ETMDBGACK

ETM Debug Mode Indication **Output**

When HIGH, this signal indicates that the processor is in debug state.

ETMDMAS[1:0]
ETM Data Size Indicator **Output**
These signals indicate the data size of the ETM. They become valid in the same cycle as the data address bus.

ETMDMAS[1:0]	Transfer Size
00	Byte
01	Halfword
10	Word
11	Reserved

ETMDMORE **ETM Sequential Data Indication** **Output**
The ETMDMORE signal is active during load and store multiple instructions and only goes HIGH when ETMDnMREQ is LOW. This signal effectively gives the same information as ETMDSEQ, but one cycle ahead. This information allows external logic more time to decode sequential cycles.

ETMDnMREQ **ETM Data Memory Request** **Output**
This signal is asserted HIGH when the ARM946E-S is making a request to ETM data memory.

ETMDnRW **ETM Data R/W** **Output**
If this signal is LOW at the end of the cycle, then any data memory access in the following cycle is a read; if this signal is HIGH, then the access is a write.

ETMDSEQ **ETM Sequential Data Indication** **Output**
If this signal is HIGH at the end of the cycle, then any data memory access in the following cycle is sequential from the last data memory access.

ETMEN **ETM Enable** **Input**
When this signal is HIGH, the ETM is enabled and the ARM9E-S interface signals are driven out of this module, pipelined by one clock stage.

ETMHIVECS **Exception Vector Location** **Output**
When this output is LOW, the ARM9E-S exception vectors start at address 0x0000.0000. When this signal is HIGH, the ARM9E-S exception vectors start at address 0xFFFF.0000. This output is a static configuration signal.

ETMIA[31:1] **ETM Instruction Address Bus** **Output**
This 31-bit bus contains the address for the ETM.

ETMIABORT **ETM Instruction Abort** **Output**
This signal is asserted HIGH to abort an ETM instruction.

ETMID31To24[31:24]
Bits [31:24] of the TAPID Register **Output**
These outputs reflect the status of bits [31:24] of the Device Identification (ID) code Test Data Register.

ETMID15To8[15:8]
Bits [15:8] of the TAPID Register **Output**
These outputs reflect the status of bits [15:8] of the Device Identification (ID) code Test Data Register.

ETMInMREQ **ETM Instruction Memory Request** **Output**
The ARM946E-S drives this output LOW to indicate that an instruction fetch will take place.

ETMINSTREXEC
ETM Instruction Execute Indicator **Output**
Assertion HIGH of this output indicates that the instruction in the execute stage of the processor pipeline has been executed.

ETMINSTRVALID
ETM Valid Instruction **Output**
This signal is the valid instruction indication for the ETM.

ETMISEQ **ETM Sequential Instruction** **Output**
The ETMISEQ signal indicates whether the fetch is sequential (HIGH) or nonsequential (LOW) to the previous access.

ETMITBIT **ETM Thumb Indication** **Output**
When this signal is LOW, the processor is in ARM state, and 32-bit instructions are fetched. When ETMITBIT is HIGH, the processor is in Thumb state, and 16-bit instructions are fetched.

ETMLATECANCEL
ETM Coprocessor Late Cancel Indicator **Output**
If this output is HIGH during the first memory cycle of a coprocessor instruction, then the coprocessor should cancel the instruction without changing any internal state.

This signal is only asserted in cycles where the previous instruction accessed memory and a data abort occurred.

ETMnWAIT **ETM Clock Stall** **Output**
When this output is LOW, it indicates the processor is stalled.

ETMPASS **ETM Coprocessor Instruction Execute Indicator** **Output**
A HIGH on this signal indicates that there is a coprocessor instruction in the execute stage of the pipeline, which must be executed.

ETMPROCID[31:0]
ETM Process ID **Output**
These signals are the Process Identifier for the ETM.

ETMPROCIDWR
ETM Process ID Write Strobe **Output**
This signal is the ETMPROCID write strobe.

ETMRDATA[31:0]
ETM Read Data **Output**
This 32-bit bus contains ETM read data.

ETMRNGOUT **ETM Watchpoint Register Match** **Output**
This output indicates that the corresponding EmbeddedICE watchpoint register has matched the conditions currently present on the address, data, and control buses. This signal is independent of the state of the watchpoint's enable control bit.

ETMWDATA[31:0]
ETM Write Data **Output**
This 32-bit bus contains ETM write data.

TAPID[31:0] **Boundary Scan ID Code** **Input**
This bus specifies the ID Code value shifted out on DBGTDO when the IDCODE instruction is entered into the TAP Controller.

2.13 ATPG Scan Control Signals

SCANEN	**Scan Enable**	**Input**
	Asserting this signal HIGH enables scanning data through the scan chain.	
SI	**Scan Chain In**	**Input**
	SI is the input for the serial scan chain. There is one SI pin for each scan chain.	
SO	**Scan Chain Out**	**Output**
	SO is the output of the serial scan chain. There is one SO pin for each scan chain.	

Chapter 3
Programmer's Model

This chapter describes the programmer's model for the ARM946E-S. It contains the following sections:

3.1 About the ARM946E-S Programmer's Model

The programmer's model for the ARM946E-S primarily consists of the ARM9E-S core programmer's model (see Section 3.2, "About the ARM9E-S Programmer's Model," on page 3-2). Additions to this model are required to control the operation of the ARM946E-S internal coprocessors and any coprocessor connected to the external coprocessor interface.

There are two internal coprocessors within the ARM946E-S:

- CP14

  CP14, which is located within the ARM9E-S core, allows software access to the debug communications channel. The registers defined in CP14 are accessible with `MCR` and `MRC` instructions, and are described in Section 3.4, "CP14 Registers" on page 3-34.

- CP15

  CP15 allows configuration of the caches, tightly coupled SRAM, protection unit, write buffer, and other ARM946E-S system options. The registers defined in CP15 are accessible with `MCR` and `MRC`

instructions, and are described in Section 3.3, “CP15 Registers,” on page 3-2.

Registers and operations provided by any coprocessors attached to the external coprocessor interface are accessible with appropriate coprocessor instructions.

3.2 About the ARM9E-S Programmer’s Model

The ARM9E-S processor core implements the ARMv5TExP architecture, which includes the 32-bit ARM instruction set and the 16-bit Thumb instruction set. For a description of both instruction sets, see the *ARM Architecture Reference Manual*. Contact ARM for complete descriptions of both instruction sets.

The ARM9E-S uses the *base restored Data Abort model*, which differs from the *base updated Data Abort model* implemented by ARM7TDMI.

The difference in the ARM9E-S Data Abort model affects only a very small section of operating system code, the Data Abort handler. It does not affect user code. With the *base restored Data Abort model*, when a Data Abort exception occurs during the execution of a memory access instruction, the processor hardware always restores the base register to the value it had *before* the instruction was executed. This action eliminates the requirement that the Data Abort handler *unwind* any base register update the aborted instruction might have caused.

The *base restored Data Abort model* significantly simplifies the Data Abort handler software.

3.3 CP15 Registers

The ARM946E-S incorporates CP15 for system control. CP15 allows configuration of the caches, tightly coupled SRAM, protection unit, write buffer, and other ARM946E-S system options—such as big- or little-endian operation. Table 3.1 shows the CP15 register map.

Table 3.1 CP15 Register Map

Register	Read	Write
0	ID code[1]	Unpredictable
0	Cache type[1]	Unpredictable
0	Tightly coupled memory size[1]	Unpredictable
1	Control	Control
2	Cache configuration[2]	Cache configuration[2]
3	Write buffer control	Write buffer control
4	Unpredictable	Unpredictable
5	Access permission[2]	Access permission[2]
6	Protection region base and size[1]	Protection region base and size[1]
7	Unpredictable	Cache operations
8	Unpredictable	Unpredictable
9	Cache lockdown[2]	Cache lockdown[2]
9	Tightly coupled memory region[2]	Tightly coupled memory region[2]
10	Unpredictable	Unpredictable
11	Unpredictable	Unpredictable
12	Unpredictable	Unpredictable
13	Process ID	Process ID
14	Unpredictable	Unpredictable
15	Test state[1]	Test state[1]
15	Cache debug index[1]	Cache debug index[1]

1. Register location provides access to more than one register. The register accessed depends on the value of the opcode_2 or CRm field. See the register description for details.
2. Separate registers for instruction and data. See the register description for details.

3.3.1 Accessing CP15 Registers

Table 3.2 defines some of abbreviations and terms used in the register descriptions.

Table 3.2 CP15 Abbreviations

Abbreviation	Term	Description
UNP	Unpredictable	For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration.
UND	Undefined	An instruction that accesses CP15 in the manner indicated takes the undefined instruction trap.
RWZ	Reserved Write zero	These bits are reserved. Write zeros to all bits in this field.
RWO	Reserved Write one	These bits are reserved. Write ones to all bits in this field.

Reading from or writing any data values to the CP15 registers, including those fields specified as *unpredictable* or *RWZ/RWO*, does not cause any permanent damage.

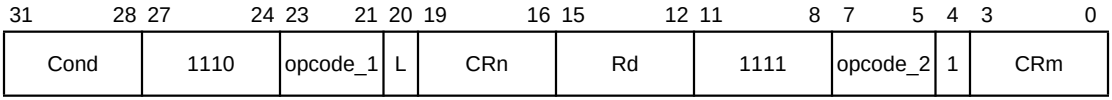
HRESETn clears to 0 all CP15 register bits that are defined and contain state, except AVS (bit 13) in Register 1. When HRESETn is asserted LOW, the the VINITHI core input pin drives the AVS bit HIGH or LOW.

The two Tightly Coupled Memory Region registers indicate the physical size of the Instruction and Data SRAMs.

You must be in privileged mode to access the CP15 registers with the `MRC` and `MCR` instructions.

Figure 3.1 shows the `MRC` and `MCR` instruction format.

Figure 3.1 MRC and MCR Instruction Format



The assembler syntax for these instructions is:

```
MCR/MRC{cond} p15, opcode_1, Rd, CRn, CRm, opcode_2
```

The processor takes an undefined instruction trap if any of the following instructions attempt to access CP15: CDP, LDC, and STC, or unprivileged MRC and MCR instructions.

The `CRn` field of the MRC and MCR instructions specifies which coprocessor register to access. The CRm field and opcode_2 field specify a particular action when addressing registers.

Attempting to read from a nonreadable register or writing to a nonwritable register causes unpredictable results.

For all instructions that access CP15, the opcode_1, opcode_2, and CRm fields should be 0, except when the values specified are used to select an operation. Using other values results in unpredictable behavior.

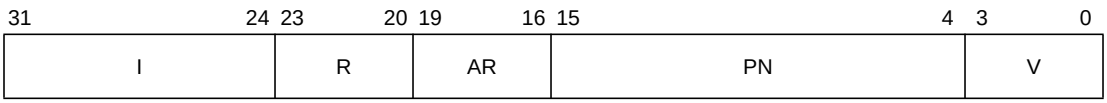
3.3.2 ID Code Register (0)

This is a read-only register that returns a 32-bit device ID code. The ID code register is accessed by reading CP15 register 0 with the opcode_2 field set to any value other than 1 or 2. For example:

```
MRC p15, 0, rd, c0, c0, {0,3–7}; returns ID register
```

Figure 3.2 shows the ID Code register format.

Figure 3.2 ID Code Register



I **Implementor** **[31:24]**
This field specifies the implementor and has a value of 0x41.

R **Reserved** **[23:20]**
This field is reserved and has a value of 0x00.

AR **Architecture Revision** **[19:16]**
This field identifies the revision level of the architecture.

PN **Part Number** **[15:4]**
This field specifies the part number. In this case, the part number is 0x946.

V **Version** **[3:0]**
This field contains the hardware version number, which is implementation specific.

3.3.3 Cache Type Register (0)

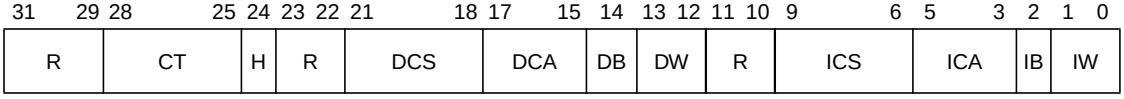
This is a read-only register that contains information about the size and architecture of the instruction cache (I-cache) and data cache (D-cache). It also allows operating systems to establish how to perform operations, such as cache cleaning and lockdown. Future ARM cached processors will contain this register, allowing RTOS vendors to produce future-proof versions of their operating systems.

The Cache Type register is accessed by reading CP15 register 0 with the opcode_2 field set to 1. For example:

```
MRC p15, 0, Rd, c0, c0, 1; returns cache details
```

Figure 3.3 shows the register format.

Figure 3.3 Cache Type Register



R **Reserved** **[31:29], [23:22], [11:10]**
These fields are reserved and have a value 0.

CT **Cache Type** **[28:25]**
This field specifies the cache type and has a value of 0b0111. This value means the cache provides:

- Cache clean-step operation
- Cache flush-step operation
- Lockdown capability

H **Harvard/Unified** **24**
This bit specifies the cache architecture. This value is 1, because the ARM946E-S uses a Harvard architecture.

DCS **Data Cache Size** **[21:18]**

This field indicates the data cache size. The actual value is implementation dependent. The encoding for the cache size bits is shown below.

Value of Bits [21:18]	Cache Size
0b0000	0 Kbytes
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

DCA **Data Cache Associativity** **[17:15]**

This field indicates the data cache associativity. The encoding is shown below.

Value	Associativity
0b000	Direct mapped
0b001	2-way set associative
0b010	4-way set associative

The actual value depends on the implementation and is defined by the implementor. If the design has a data cache, the associativity for that cache is set to 0b010 to indicate a 4-way set associative cache.

DB **Data Cache Base Size** **14**

This bit indicates the data cache base size. The value is implementation dependent. If there is a data cache, this bit is cleared to 0 to indicate that the cache type parameters are valid. If there is no data cache, this bit is set to 1 to indicate the data cache is not present.

DW **Data Cache Words per Line** **[13:12]**

This field specifies the data cache words per line. The value is 0b10, which specifies 8 words per line.

ICS **Instruction Cache Size** **[9:6]**

This field indicates the instruction cache size, and the value is implementation dependent. The bit encoding is shown below.

Value of Bits [9:6]	Cache Size
0b0000	0 Kbytes
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

ICA **Instruction Cache Associativity** **[5:3]**

This field indicates the instruction cache associativity. The encoding is shown below.

Value	Associativity
0b000	Direct mapped
0b001	2-way set associative
0b010	4-way set associative

The actual value depends on the implementation and is defined by the implementor. If the design has an instruction cache, the associativity for that cache is set to 0b010 to indicate a 4-way set associative cache.

IB **Instruction Cache Base Size** **2**

This bit specifies the instruction cache base size. The value is implementation dependent. If there is an instruction cache, this bit is cleared to 0 to indicate that the cache type parameters are valid. If there is no instruction cache, this bit is set to 1 to indicate the instruction cache is not present.

IW **Instruction Cache Word per Line** **[1:0]**
This field specifies the instruction cache words per line. The value is 0b10, which is 8 words per line.

Note: The cache base size and cache size fields are generated within the cache block so designs with different cache sizes do not have to be resynthesized.

3.3.4 Tightly Coupled Memory Size Register (0)

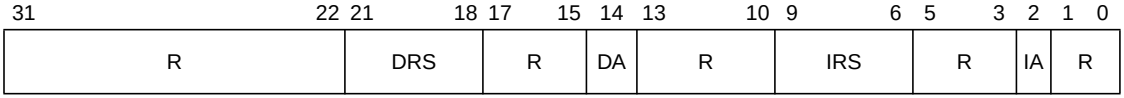
This is a read-only register that returns the size of the tightly coupled instruction and data RAMs included within the ARM946E-S.

To access the Tightly Coupled Memory Size register, read CP15 register 0 with the opcode_2 field set to 2. For example:

```
MRC p15, 0, rd, c0, c0, 2; returns tightly coupled memory
size register
```

Figure 3.4 shows the register format.

Figure 3.4 Tightly Coupled Memory Size Register



R **Reserved** **[31:22], [17:15], [13:10]. [5:3], [1:0]**
These fields are reserved and have a value 0.

DRS **Data RAM Size** **[21:18]**
This field specifies the Data RAM size. It is implementation specific. The values are generated within the memory blocks, which allows changing the memory size without resynthesizing the full design. The bit encoding is shown below.

Value of bits [21:18]	Data RAM Size
0b0000	0 Kbytes
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

DA **Data RAM Absent** **14**

If this bit is set. the Data RAM is *not* present. If it is clear, the Data RAM is present.

IRS **Instruction RAM Size** **[9:6]**

This field specifies the instruction RAM size. It is implementation specific. The values are generated within the memory blocks, which allows changing the memory size without resynthesizing the full design. The bit encoding is shown below.

Value of bits [9:6]	Instruction RAM Size
0b0000	0 Kbytes
0b0011	4 Kbytes
0b0100	8 Kbytes
0b0101	16 Kbytes
0b0110	32 Kbytes
0b0111	64 Kbytes
0b1000	128 Kbytes
0b1001	256 Kbytes
0b1010	512 Kbytes
0b1011	1 Mbyte

IA **Instruction RAM Absent** **2**
If this bit is set, the Instruction RAM is *not* present. If it is clear, the Instruction RAM is present.

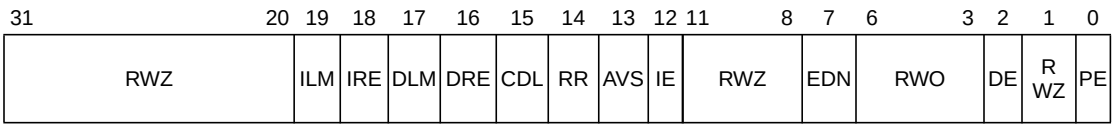
3.3.5 Control Register (1)

This register contains the ARM946E-S control bits. All reserved bits must be written with either 0 or 1, as indicated below, or written using read-modify-write. The reserved bits have an unpredictable value when read. To read and write this register:

```
MRC p15, 0, rd, c1, c0, 0; read control register
MCR p15, 0, rd, c1, c0, 0; write control register
```

Figure 3.5 shows the register format.

Figure 3.5 Control Register



RWZ **Reserved - Write Zero** **[31:20], [11:8], 1**
These bits are reserved. Write zeros to these bits.

ILM **Instruction RAM Load Mode** **19**
When this bit is set to 1, you can use the instruction RAM load mode to initialize the instruction RAM. This mode allows you to load data into ARM registers from either data cache or main memory, and then write to the same address but within the tightly coupled instruction RAM. This capability allows you to copy boot code from memory located at address 0x0 into the instruction RAM which, when enabled, also exists at address 0x0. The operation of the load mode is described in Section 6.2.3, “I-SRAM Load Mode,” on page 6-3.

At reset, this bit is cleared.

IRE **Instruction RAM Enable** **18**
When this bit is set to 1, the instruction RAM is enabled, and all instruction and data accesses to the instruction RAM address range access the instruction RAM.

At reset, this bit is cleared.

DLM **Data RAM Load Mode** **17**

When this bit is set to 1, you can use the data RAM load mode for initializing the data RAM. First, do a load data into ARM registers from either the data cache or main memory. Then store the data from the ARM registers into the tightly coupled data RAM using the same address from which the data originated. The operation of the load mode is described in Section 6.2.3, "I-SRAM Load Mode," on page 6-3.

At reset, this bit is cleared.

DRE **Data RAM Enable** **16**

When this bit is set to 1, the data RAM is enabled. Then the data RAM takes precedence over the data cache and AHB for data accesses.

At reset, this bit is cleared.

CDL **Configure Disable Loading TBIT** **15**

This bit controls the behavior of load PC instructions. When cleared to 0, the ARMv5TExP-specific behavior is enabled, and bit 0 of the loaded data controls the entry into the Thumb state when the PC (r15) is the destination register. When set to 1, this ARMv5TExP behavior is disabled.

At reset, this bit is cleared.

RR **Round-Robin Replacement** **14**

This bit controls the cache replacement algorithm. When set to 1, round-robin replacement is used. When cleared to 0, a pseudo-random replacement algorithm is used.

At reset, this bit is cleared.

AVS **Alternate Vectors Select** **13**

This bit controls the base address used for the exception vectors. When cleared to 0, the base address for the exception vectors is 0x00000000. When set to 1, the base address is 0xFFFF0000.

Note: This bit is initialized to either 1 or 0 during system reset, depending on the value of the input pin, VINITHI. This allows you to define the exception vector location during reset to suit the boot mechanism of the application. You can then reprogram this bit as required following system reset.

IE	**I-Cache Enable** This bits controls I-cache behavior. To use the instruction cache, both the protection unit enable bit (bit 0) and the I-cache enable bit must be set to 1. You can do this with a single write to register 1. At reset, this bit is cleared.	**12**
EDN	**Endian** This bit selects the endian configuration of the ARM946E-S. When this bit is set to 1, the big-endian configuration is selected. When cleared to 0, the little-endian configuration is selected. At reset, this bit is cleared.	**7**
RWO	**Reserved - Write Ones** This field is reserved. Write ones to these bits.	**[6:3]**
DE	**D-Cache Enable** This bit controls the behavior of the D-cache. To use the data cache, both the protection unit enable bit (bit 0) and the D-cache enable bit must be set to 1. This can be done with a single write to register 1. At reset, this bit is cleared.	**2**
PE	**Protection Unit Enable** This bit controls the operation of the ARM946E-S Protection Unit. At reset, this bit is cleared, which disables the Protection Unit. It also disables the instruction cache, data cache, and the write buffer. At least one protection region must be programmed before the Protection Unit is enabled. See Section 3.3.9, “Protection Region/Base Size (PR/BS) Registers (6),” on page 3-19 and Chapter 3, “Programmer’s Model.”	**0**

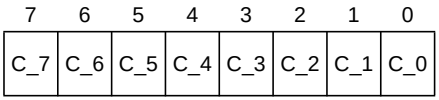
3.3.6 Cache Configuration Registers (2)

These registers contain the cacheable attributes for the eight memory regions. Individual control is provided for the I and D caches. If the opcode_2 field = 0, then the data cache bits are programmed. If the opcode_2 field = 1, then the instruction cache bits are programmed. To read and write these registers:

```
MRC p15, 0, rd, c2, c0, 0; read data cacheable bits
MRC p15, 0, rd, c2, c0, 1; read instruction cacheable bits
MCR p15, 0, rd, c2, c0, 0; write data cacheable bits
MCR p15, 0, rd, c2, c0, 1; write instruction cacheable bits
```

The format for the cacheable bits in the data and instruction areas is the same. Figure 3.6 shows the register format.

Figure 3.6 Instruction/Data Cacheable Bits Register



C_n **Cacheable bits for Memory Region 7:0 [7:0]**
These bits allow you to individually enable or disable the cacheable attribute for each memory region. Setting a bit to 1 makes the corresponding memory region cacheable; clearing the bit makes that memory region uncacheable.

Bit	Corresponds to Memory Region
C_7	7
C_6	6
C_5	5
C_4	4
C_3	3
C_2	2
C_1	1
C_0	0

3.3.7 Write Buffer Control Register (3)

This register contains the write buffer control (bufferable) attribute for the eight memory regions.

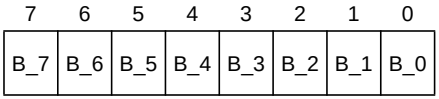
Note: This register only applies to data accesses.

To read and write the write buffer control register:

```
MCR p15, 0, rd, c3, c0, 0; write data bufferable bits
MRC p15, 0, rd, c3, c0, 0; read data bufferable bits
```

Figure 3.7 shows the register format.

Figure 3.7 Write Buffer Control Register



B_n **Bufferable bits for Memory Area 7:0 [7:0]**
These bits allow you to individually enable or disable the bufferable attribute for each memory region. Setting a bit to 1 makes the corresponding memory area bufferable; clearing the bit makes that memory region not bufferable.

Bit	Corresponds to Memory Data Region
B_7	7
B_6	6
B_5	5
B_4	4
B_3	3
B_2	2
B_1	1
B_0	0

3.3.8 Access Permission Registers (5)

There are four access permission registers:

- Instruction Access Permission (Extended)
- Data Access Permission (Extended)
- Instruction Access Permission (Standard)
- Data Access Permission (Standard)

These registers contain the access permission bits for the instruction and data protection regions. The opcode_2 field of the `MCR/MRC` instruction determines whether to access the standard or extended instruction or data access permission registers.

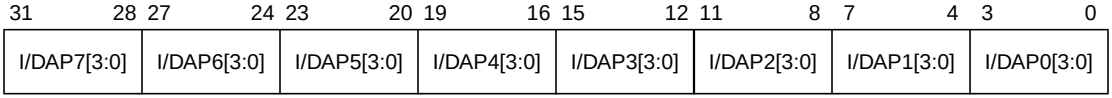
3.3.8.1 Extended Instruction/Data Access Permission Registers

To read and write the extended registers:

```
MRC p15, 0, rd, c5, c0, 2; read data access permission bits
MRC p15, 0, rd, c5, c0, 3; read instruction access permission bits
MCR p15, 0, rd, c5, c0, 2; write data access permission bits
MCR p15, 0, rd, c5, c0, 3; write instruction access permission bits
```

Figure 3.8 shows the extended instruction/data register format. The same format applies to both the Instruction and Data Access Permission areas.

Figure 3.8 Instruction/Data Access Permission (I/DAP) Register (Extended)



I/DAP7[3:0] Instruction/Data Access Permission 7 [31:28]

This field contains the access permission bits for area 7. The bit encoding is shown below.

	Access Permission	
I/DAP7[3:0]	**Privileged**	**User**
0b0000	No access	No access
0b0001	Read/write access	No access
0b0010	Read/write access	Read-only
0b0011	Read/write access	Read/write access
0b0100	Unpredictable	Unpredictable
0b0101	Read-only	No access
0b0110	Read-only	Read-only
0b0111	Unpredictable	Unpredictable
0b1xxx	Unpredictable	Unpredictable

I/DAP6[3:0] Instruction/Data Access Permission 6 [27:24]

This field contains the access permission bits for area 6. The bit encoding for these bits is the same as I/DAP7[3:0].

I/DAP5[3:0] Instruction/Data Access Permission 5 [23:20]

This field contains the access permission bits for area 5. The bit encoding for these bits is the same as I/DAP7[3:0].

I/DAP4[3:0] **Instruction/Data Access Permission 4** **[19:16]**
This field contains the access permission bits for area 4. The bit encoding for these bits is the same as I/DAP7[3:0].

I/DAP3[3:0] **Instruction/Data Access Permission 3** **[15:12]**
This field contains the access permission bits for area 3. The bit encoding for these bits is the same as AP7[3:0].

I/DAP2[3:0] **Instruction/Data Access Permission 2** **[11:8]**
This field contains the access permission bits for area 2. The bit encoding for these bits is the same as I/DAP7[3:0].

I/DAP1[3:0] **Instruction/Data Access Permission 1** **[7:4]**
This field contains the access permission bits for area 1. The bit encoding for these bits is the same as I/DAP7[3:0].

I/DAP0[3:0] **Instruction/Data Access Permission 0** **[3:0]**
This field contains the access permission bits for area 0. The bit encoding for these bits is the same as I/DAP7[3:0].

3.3.8.2 Standard Instruction/Data Access Permission Registers

The following instructions are supported for backward compatibility with existing ARM processors with memory protection, and they access the standard registers:

```
MRC p15, 0, rd, c5, c0, 0; read data access permission bits
MRC p15, 0, rd, c5, c0, 1; read instruction access permission bits
MCR p15, 0, rd, c5, c0, 0; write data access permission bits
MCR p15, 0, rd, c5, c0, 1; write instruction access permission bits
```

Figure 3.9 shows the extended instruction/data register format. The same format applies to both the Instruction and Data Access Permission registers.

Figure 3.9 Instruction/Data Access Permission (I/DAP) Register (Standard)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
I/DAP7[1:0]	I/DAP6[1:0]	I/DAP5[1:0]	I/DAP4[1:0]	I/DAP3[1:0]	I/DAP2[1:0]	I/DAP1[1:0]	I/DAP0[1:0]

I/DAP7[1:0] Instruction/Data Access Permission 7 [15:14]
This field contains the access permission bits for area 7. The bit encoding is shown below.

	Access Permission	
I/DAP7[1:0]	**Privileged**	**User**
0b00	No access	No access
0b01	Read/write access	No access
0b10	Read/write access	Read-only
0b11	Read/write access	Read/write access

I/DAP6[1:0] Instruction/Data Access Permission 6 [13:12]
This field contains the access permission bits for area 6. The bit encoding for these bits is the same as I/DAP7[1:0].

I/DAP5[1:0] Instruction/Data Access Permission 5 [11:10]
This field contains the access permission bits for area 5. The bit encoding for these bits is the same as I/DAP7[1:0].

I/DAP4[1:0] Instruction/Data Access Permission 4 [9:8]
This field contains the access permission bits for area 4. The bit encoding for these bits is the same as I/DAP7[1:0].

I/DAP3[1:0] Instruction/Data Access Permission 3 [7:6]
This field contains the access permission bits for area 3. The bit encoding for these bits is the same as I/DAP7[1:0].

I/DAP2[1:0] Instruction/Data Access Permission 2 [5:4]
This field contains the access permission bits for area 2. The bit encoding for these bits is the same as I/DAP7[1:0].

I/DAP1[1:0] **Instruction/Data Access Permission 1** **[3:2]**
This field contains the access permission bits for area 1. The bit encoding for these bits is the same as I/DAP7[1:0].

I/DAP0[1:0] **Instruction/Data Access Permission 0** **[1:0]**
This field contains the access permission bits for area 0. The bit encoding for these bits is the same as I/DAP7[1:0].

3.3.8.3 Programming the Access Permission Registers

At reset, the value of the I/DAPn bits is undefined. However, because the protection unit is disabled on reset, in effect all areas are set to privileged mode with user read/write access. Therefore, you must program the access permission registers before you enable the protection unit.

If the access permissions are initially programmed using the extended access permissions and then reprogrammed using the standard access permissions, the access permissions are applied as if I/DAPn[3:2] are 0b00.

3.3.9 Protection Region/Base Size (PR/BS) Registers (6)

There are eight Protection Region/Base Size registers. You can define eight programmable regions using these registers. The values are ignored when the protection unit is disabled, and on reset only the region enable bit for each region is reset to 0. All other bits are undefined. You must program at least one memory region before you enable the protection unit.

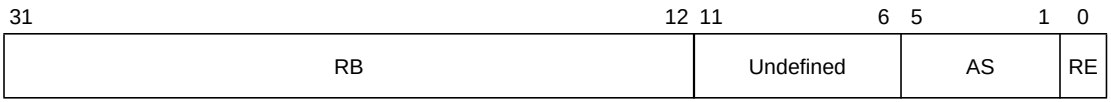
The instructions that access the eight protection region/base size registers are listed in Table 3.3.

Table 3.3 Accessing PR/BS Registers

ARM Instruction	PR/BS Register	Memory Region
MCR/MRC p15, 0, rd, c6, c7, 0	7	7
MCR/MRC p15, 0, rd, c6, c6, 0	6	6
MCR/MRC p15, 0, rd, c6, c5, 0	5	5
MCR/MRC p15, 0, rd, c6, c4, 0	4	4
MCR/MRC p15, 0, rd, c6, c3, 0	3	3
MCR/MRC p15, 0, rd, c6, c2, 0	2	2
MCR/MRC p15, 0, rd, c6, c1, 0	1	1
MCR/MRC p15, 0, rd, c6, c0, 0	0	0

Figure 3.10 shows the PR/BS register format.

Figure 3.10 PR/BS Register



RB **Region Base** **[31:12]**
This field specifies the region base.

U **Undefined** **[11:6]**
These bits are undefined.

AS **Area Size** **[5:1]**
This field determines the area size. The bit encoding is shown below.

Bit Encoding[1]	Area Size
0b00000 to 01010	Reserved (UNP)
0b01011	4 Kbytes
0b01100	8 Kbytes
0b01101	16 Kbytes
0b01110	32 Kbytes
0b01111	64 Kbytes

Bit Encoding[1]	Area Size
0b10000	128 Kbytes
0b10001	256 Kbytes
0b10010	512 Kbytes
0b10011	1 Mbyte
0b10100	2 Mbytes
0b10101	4 Mbytes
0b10110	8 Mbytes
0b10111	16 Mbytes
0b11000	32 Mbytes
0b11001	64 Mbytes
0b11010	128 Mbytes
0b11011	256 Mbytes
0b11100	512 Mbytes
0b11101	1 Gbyte
0b11110	2 Gbytes
0b11111	4 Gbytes

1. Using any value less than 0b01011 causes unpredictable behavior.

RE **Region Enable** **0**

When this bit is set, it enables the memory region associated with this register. When cleared, it disables the memory region.

You must align the region base to an area size boundary, where the area size is defined in its respective protection region register. If this is not done, the behavior is unpredictable.

3.3.9.1 Example Base Setting

An 8 Kbyte-size region aligned to an 8 Kbyte boundary at 0x0000.2000 (covering the address range 0x0000.2000 to 0x0000.3FFF) is programmed as 0x0000.2019.

The following instruction allows the protection region registers to be read, and it provides backward compatibility with other ARM processors that use a memory protection unit.

```
MRC p15, 0, rd, c6, CRm, 1; returns protection region
register
```

Writes to the Protection Region/Base Size registers with opcode_2 set to 1 are unpredictable.

3.3.10 Cache Operations Register (7)

You can perform the following cache operations by writing to the Cache Operations register:

- Flush I-cache and D-cache
- Prefetch an I-cache line
- Clean and flush the D-cache
- Drain the write buffer
- Wait for interrupt

The ARM946E-S uses a subset of the ARM architecture v4 functions that are defined in the ARM Architecture Reference Manual. Table 3.4 summarizes the ARM946E-S cache operations.

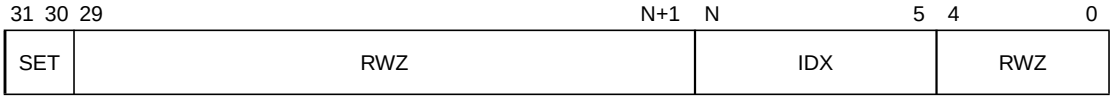
Table 3.4 Cache Operations

ARM Instruction	Cache Operation	Written to Register 7
MCR p15, 0, rd, c7, c5, 0	Flush I-cache	Zero[1]
MCR p15, 0, rd, c7, c5, 1	Flush I-cache single entry	Address
MCR p15, 0, rd, c7, c13, 1	Prefetch I-cache line	Address
MCR p15, 0, rd, c7, c6, 0	Flush D-cache	Zero[1]
MCR p15, 0, rd, c7, c6, 1	Flush D-cache single entry	Address
MCR p15, 0, rd, c7, c10, 1	Clean D-cache entry	Address
MCR p15, 0, rd, c7, c14, 1	Clean and flush D-cache entry	Address
MCR p15, 0, rd, c7, c10, 2	Clean D-cache entry	Index/Set
MCR p15, 0, rd, c7, c14, 2	Clean and flush D-cache entry	Index/Set

1. The Rd value transferred to Register 7 should be 0.

Figure 3.11 shows the format for operations that transfer index/set information to the Cache Operations register.

Figure 3.11 Index and Set Format



SET **Set** **[31:30]**
This field specifies the cache set.

RWZ **Reserved - Write Zero** **[29:N+1]**
These bits are reserved. Write zeros to these bits.

IDX **Index** **[N:5]**
This field specifies the cache index.

The size of the index varies depending on the implemented cache size. Table 3.5 shows how the index size changes for the cache sizes supported by the ARM946E-S.

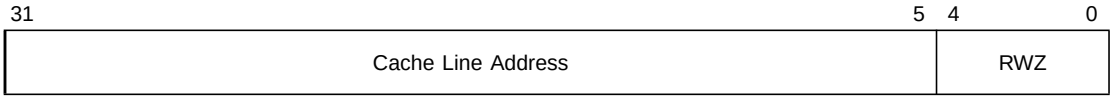
Table 3.5 Index Fields for Supported Cache Sizes

Cache Size	Index
4 Kbytes	Addr[9:5]
8 Kbytes	Addr[10:5]
16 Kbytes	Addr[11:5]
32 Kbytes	Addr[12:5]
64 Kbytes	Addr[13:5]
128 Kbytes	Addr[14:5]
256 Kbytes	Addr[15:5]
512 Kbytes	Addr[16:5]
1 Mbyte	Addr[17:5]

Table 3.5 assumes a 4-way set associative cache. As associativity decreases the cache needs more address bits. For a 2-way cache, each index address entry in the table requires one additional address bit. For a direct-mapped cache, two additional bits are required.

For cache operations that write an address to Register 7 (see Table 3.4), Figure 3.12 shows the register format.

Figure 3.12 Address Format



3.3.10.1 Cache Clean and Flush Operations

Cache clean and flush operations can occur during instruction and data line fetches. In these circumstances, the line fetch completes before the clean or flush operation is executed.

3.3.10.2 Noncache Operations

Writing to the Cache Operations register is also used for two noncache operations:

- Drain Write Buffer
- Wait for Interrupt

Drain Write Buffer – This operation stalls instruction execution until the write buffer is emptied. Stalling is useful in real-time applications where the processor must be sure that a write to a peripheral has finished before program execution continues. For example, if a peripheral in a bufferable region is the source of an interrupt, then after the interrupt is serviced, the request must be removed before interrupts are enabled again. This requirement is ensured if there is a drain write buffer operation between the store to the peripheral and the interrupt enable.

Writing to Register 7 invokes the drain write buffer operation. Use the following ARM instruction:

```
MCR cp15, 0, rd, c7, c10, 4; drain write buffer
```

This write transfer stalls the processor core until all outstanding accesses in the write buffer are completed. In other words, it stalls the processor until all data in the buffer is written to external memory.

Wait for Interrupt – This operation allows the ARM946E-S to enter a low-power standby mode. When you invoke the operation, the CLKEN

signal to the processor core is negated and the cache and tightly coupled memories are placed in a low-power state until either an interrupt or a debug request occurs. This Wait for Interrupt operation is invoked by writing to Register 7 using the following ARM instruction:

```
MCR p15, 0, rd, c7, c0, 4; wait for interrupt
```

This encoding is preferred for new software. For compatibility with existing software, ARM946E-S also supports the following ARM instruction, which has the same effect:

```
MCR p15, 0, rd, c15, c8, 2; wait for interrupt
```

This instruction stalls the processor from the time that the instruction is executed until either nFIQ, nIRQ, or EDBGRQ are asserted. If the debugger sets the debug request bit in the EmbeddedICE-RT logic control register, it causes the *wait for interrupt* condition to terminate.

In the case of nFIQ and nIRQ, the processor core wakes up regardless of whether the interrupts are enabled or disabled (that is, independent of the I and F bits in the processor CPSR). The debug related wake up only occurs if DBGEN is HIGH, that is, only when debug is enabled.

If interrupts are enabled, the ARM9E-S core is guaranteed to take the interrupt before executing the instruction after the wait for interrupt operation. If a debug request is used to wake up the system, the processor enters the debug state before executing any more instructions.

The write buffer continues to drain until empty while the wait for interrupt operation is executing.

3.3.11 Cache Lockdown Registers (9)

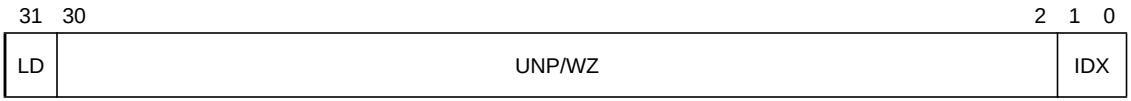
The Cache Lockdown registers allow you to lock down regions of the cache. There are separate registers for the instruction and data cache.

To read and write the registers:

```
MCR p15, 0, rd, c9, c0, 0; write data lockdown control
MRC p15, 0, rd, c9, c0, 0; read data lockdown control
MCR p15, 0, rd, c9, c0, 1; write instruction lockdown
control
MRC p15, 0, rd, c9, c0, 1; read instruction lockdown control
```

Figure 3.12 shows the register format.

Figure 3.13 Cache Lockdown Register



LD **Load** **31**
When this bit is set to 1, it indicates this is a lockdown load operation.

UNP/WZ **Unpredictable - Write Zero** **[30:2]**
These bits are unpredictable, so write all zeros to them.

IDX **Index Field** **[1:0]**
This field specifies the cache set.

For a description of Lockdown, refer to *Section 4.4, "Cache Lockdown."*

3.3.12 Tightly Coupled Memory Region Registers (9)

These registers allow you to modify the visible size of the instruction and data tightly coupled memories.

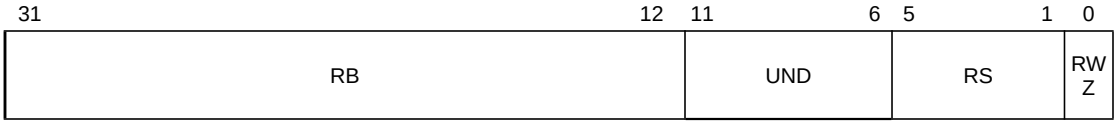
You can either increase or decrease the size of the tightly coupled memories from the physical sizes specified in Register 0. (See Section 3.3.4, "Tightly Coupled Memory Size Register (0)," on page 3-9.) Increasing the visible size of the tightly coupled memories above the physical size allows aliasing within the tightly coupled memory space. This feature is useful for debugging multitasking systems.

There are two memory region registers, one for each of the tightly coupled memories:

```
MRC p15, 0, rd, c9, c1, 0; read data tightly coupled memory
MCR p15, 0, rd, c9, c1, 0; write data tightly coupled memory
MRC p15, 0, rd, c9, c1, 1; read instruction tightly coupled memory
MCR p15, 0, rd, c9, c1, 1; write instruction tightly coupled memory
```

Figure 3.12 shows the register format.

Figure 3.14 Tightly Coupled Memory Region Register Format



RB **Region Base Address** **[31:12]**
This field contains the Region Base address.

UND **Undefined** **[11:6]**
These bits are undefined.

RS **Region Size** **[5:1]**
This field specifies the region size, which can range from 4 Kbytes (minimum) to 4 Gbytes (maximum). The bit encodings are shown below.

Bit Encoding	Tightly Coupled Memory Region Size
0b00011	4 Kbytes
0b00100	8 Kbytes
0b00101	16 Kbytes
0b00110	32 Kbytes
0b00111	64 Kbytes
0b01000	128 Kbytes
0b01001	256 Kbytes
0b01010	512 Kbytes
0b01011	1Mbyte
0b01100	2 Mbytes
0b01101	4 Mbytes
0b01110	8 Mbytes
0b01111	16 Mbytes
0b10000	32 Mbytes
0b10001	64 Mbytes

Bit Encoding	Tightly Coupled Memory Region Size
0b10010	128 Mbytes
0b10011	256 Mbytes
0b10100	512 Mbytes
0b10101	1 Gbyte
0b10110	2 Gbytes
0b10111	4 Gbytes

RWZ **Reserved - Write Zero** **0**
This bit is reserved. Write a zero to this bit.

For a given number of aliases for the physical memory size, use the following function:

Area size = Physical size + N

where 2^N is the required number of aliases.

You must align the region base to an area size boundary, where the area size is defined in its respective protection region register. The behavior is unpredictable if this is not done.

The instruction tightly coupled memory base address is fixed at 0x00000. For the instruction tightly coupled memory, the region base returns the value 0x00000 when read.

When writing to the instruction tightly coupled memory, you must set the region base to 0x00000. Writes with the region base set to any other value are unpredictable.

At reset, the region base for both the instruction and data Tightly Coupled Memory Region registers is cleared to 0x00000.

At reset, the area size for the instruction and data Tightly Coupled Memory Region registers takes the value defined in the Tightly Coupled Memory Size register. (See Section 3.3.4, “Tightly Coupled Memory Size Register (0),” on page 3-9.)

You must program the Data Tightly Coupled Memory Region registers before you set the data RAM enable bit (bit 16) in Register 1. (See Section 3.3.5, “Control Register (1),” on page 3-11.) If this is not done,

the data tightly coupled memory resides at the same location, which causes unpredictable behavior.

Note: If the data tightly coupled memory is located at the same address as the instruction tightly coupled memory, then the instruction memory takes precedence for data accesses.

If the data tightly coupled memory is located at the same address as the instruction tightly coupled memory, and the instruction RAM is in load mode, data accesses are read from the data RAM and written to the instruction RAM.

3.3.13 Trace Process Identifier Register (13)

This register allows you to identify the currently executing process in multitasking environments using the real-time trace tools.

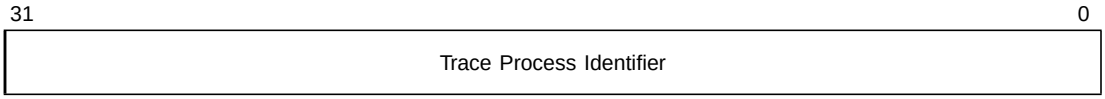
The contents of this register are output on the ETMPROCID pins of the ARM946E-S.

The following ARM instructions are used for accessing the Process ID register:

```
MRC p15, 0, rd, c13, c1, 1; read process ID register
MCR p15, 0, rd, c13, c1, 1; write process ID register
```

Figure 3.15 shows the register format.

Figure 3.15 Trace Process ID Register



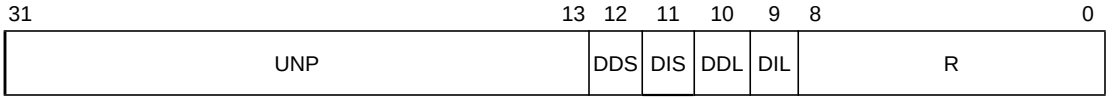
3.3.13.1 Register 15, Test State Register

Register 15 gives you access to the test features included within the ARM946E-S. Use these instructions to access the register:

```
MCR {cond} p15, 0, rd, c15, c0, 0; write test state register
MRC {cond} p15, 0, rd, c15, c0, 0; read test state register
```

Figure 3.15 shows the register format.

Figure 3.16 Test State Register



UNP **Unpredictable** **[31:13]**
When read, these bits return unpredictable data. Writing to these bits can cause unpredictable behavior or changes to device configuration.

DDS **Disable Data Cache Streaming** **12**
When this bit is 1, it prevent the data cache from streaming data to the ARM9E-S during a cache line fill. When the bit is 0, data cache streaming is permitted.

DIS **Disable Instruction Cache Streaming** **11**
When this bit is 1, it prevents the instruction cache from streaming data to the ARM9E-S during a cache line fill. When the bit is 0, instruction cache streaming is permitted.

DDL **Disable Data Cache Line Fill** **10**
When this bit is 1, it prevents the data cache from doing a line fill on a cache miss. When the bit is 0, line fills are permitted.

DIL **Disable Instruction Cache Line Fill** **9**
When this bit is 1, it prevents the data cache from doing a line fill on a cache miss. When the bit is 0, line fills are permitted.

R **Reserved** **[8:0]**
These bits are reserved.

Reading the Test State register returns bits [12:0] in the least significant bits. The 19 most significant bits are unpredictable. Writing the Test State register updates only bits [12:9].

In debug mode, you must be able to execute code without causing line fills to update the caches, primarily to load new code into memory. This means that STRs, if they hit the cache, must update the memory and the cache, and that for LDRs or instruction prefetches that miss, a line fill is not performed. When set, bits [10:9] prevent the respective cache from performing a line fill on a cache miss. The memory mapping, as seen by

the ARM9E-S or by the programmer, is unchanged. This feature improves the performance of single-stepping when in debug mode. There is one side effect from this capability. For cached accesses, the BIU returns aborts from the AHB. However, when line fills are disabled, the access on the AHB is treated as uncacheable, so aborts may return that otherwise would have been blocked.

When set, bits [12:11] prevent the respective cache from streaming data to the ARM9E-S while the line fill is performed to the cache. The line fill still occurs, but the prefetched instruction or load data is returned to the processor at the end of a line fill.

3.3.14 Cache Debug Index Register (15)

The Cache Debug Index register allows access to any location within the instruction or data cache for debugging purposes.

Table 3.6 lists cache debugging operations that use this register.

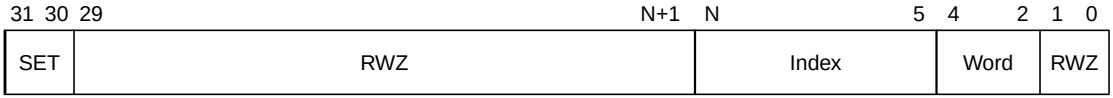
Table 3.6 Cache Debug Operations

Instruction	Operation	Data
MCR p15, 3, rd, c15, c0, 0	Write CP15 cache debug index register	Index/Set
MRC p15, 3, rd, c15, c0, 0	Read CP15 cache debug index register	Index/Set
MCR p15, 3, rd, c15, c1, 0	Instruction tag write[1]	Data
MRC p15, 3, rd, c15, c1, 0	Instruction tag read[1]	Data
MCR p15, 3, rd, c15, c2, 0	Data tag write[1]	Data
MRC p15, 3, rd, c15, c2, 0	Data tag read[1]	Data
MCR p15, 3, rd, c15, c3, 0	Instruction cache write[1]	Data
MRC p15, 3, rd, c15, c3, 0	Instruction cache read[1]	Data
MCR p15, 3, rd, c15, c4, 0	Data cache write[1]	Data
MRC p15, 3, rd, c15, c4, 0	Data cache read[1]	Data

1. You must program the Cache Debug Index register before using any of the tag or cache read/write operations.

The Cache Debug Index register provides an index into the cache memories and uses the format shown in Figure 3.17.

Figure 3.17 Cache Debug Index Register - Index/Set Format



SET **Set** **[31:30]**
This field specifies which one of the four cache sets to access.

RWZ **Reserved - Write Zero** **[29:N+1], [1,0]**
These bits are reserved. Write zeros to these bits.

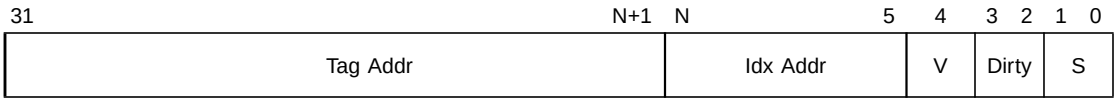
Index **Index** **[N:5]**
This field specifies which cache index to access during a cache debug operation. The size of the index varies depending on the implemented cache size. Table 3.7 on page 3-33 shows how much the index address field size differs for each cache size the ARM946E-S supports. Note that the table is for a 4-way set associative cache and the index value must be adjusted for 2-way and direct-mapped caches.

Word **Word Address** **[4:2]**
This field specifies which cache word to access in a cache debug operation.

Note: For tag operations, the word address field in the Cache Debug Index register is ignored.

Figure 3.18 shows the data format for tag read/write operations.

Figure 3.18 Data Format for Tag Read/Write Operations



Tag Addr **Tag Address** **[31:N+1]**
This field contains the tag address to be read from or written into the cache tag RAM. The size of the tag address varies according to the implemented cache size (see Table 3.7).

Idx Addr **Index Address** **[N:5]**
This field contains the Index address to be read from or written to the cache tag RAM. The size of the index varies according to the implemented cache size (see Table 3.7).

V **Valid Bit** **4**
This bit is the validity bit associated with a given tag location. When set, it indicates the cache data associated with this tag location holds valid data.

Dirty **Dirty Bits** **[3:2]**
When set, these bits indicate the data associated with a given tag location differs from external memory.

S **Set** **[1:0]**
This field indicates which data or instruction cache set to access.

Table 3.7 shows how the index and tag address field sizes differ for each cache size the ARM946E-S supports.

Table 3.7 Tag and Index Fields for Supported Cache Sizes

Cache Size	Tag	Index
4 Kbytes	Addr[31:10]	Addr[9:5]
8 Kbytes	Addr[31:11]	Addr[10:5]
16 Kbytes	Addr[31:12]	Addr[11:5]
32 Kbytes	Addr[31:13]	Addr[12:5]
64 Kbytes	Addr[31:14]	Addr[13:5]
128 Kbytes	Addr[31:15]	Addr[14:5]
256 Kbytes	Addr[31:16]	Addr[15:5]
512 Kbytes	Addr[31:17]	Addr[16:5]
1 Mbyte	Addr[31:18]	Addr[17:5]

The values shown in Table 3.7 are for a 4-way set associative cache. The index must be increased by 1 and the tag decreased by 1 for a 2-way set associative cache. For a direct-mapped cache, the tag and index must be adjusted by 2.

3.4 CP14 Registers

CP14 contains four registers, which are listed in Table 3.8.

Table 3.8 Coprocessor 14 Register Map

Register Name	Register Number	Notes
Comms channel status	C0	Read-only
Comms channel data read	C1	For reads
Comms channel data write	C1	For writes
Debug status	C2	Read/write

3.4.1 Debug Comms Channel Status Register (C0)

The Debug Comms Channel Status register is read-only. It controls synchronized handshaking between the processor and the debugger. Figure 3.19 shows the register format.

Figure 3.19 Debug Comms Channel Status Register

Each register bit functions as follows:

VER **Version Number** **[31:28]**
This field contains a fixed pattern that denotes the EmbeddedICE-RT version number (in this case 0b0011).

RWZ **Reserved - Write Zero** **[27:2]**
These bits are reserved. Write zeros to these bits.

W **Write Register** **1**
This bit indicates whether or not the Comms Data Write register is available to the processor. If the bit is 0, the register is available and the processor can write new data to it. If the bit is 1, the register is not free and the processor must poll until W = 0. From the point of view of the debugger, when W = 1, new data has been written that can then be scanned out.

R	**Read Register** This bit indicates whether or not there is new data in the Comms Data Read register. If this bit is 1, from the processor's viewpoint there is new data that can be read using an `MRC` instruction. If this bit is 0, from the debugger's viewpoint the Comms Data Read register is free, and new data can be placed there through the scan chain. When this bit is 1, data previously placed there through the scan chain has not been collected by the processor yet, so the debugger must wait.	**0**

From the viewpoint of the debugger, the registers are accessed using the scan chain. From the viewpoint of the processor, the registers are accessed using the coprocessor register transfer instructions. It is recommended that you use the following instructions:

1. This instruction returns the Debug Comms Control register into Rd:
   `MRC p14, 0, Rd, c0, c0`
2. This instruction writes the value in Rn to the Comms Data Write register:
   `MCR p14, 0, Rn, c1, c0`
3. This instruction returns the Debug Data Read register into Rd:
   `MRC p14, 0, Rd, c1, c0`

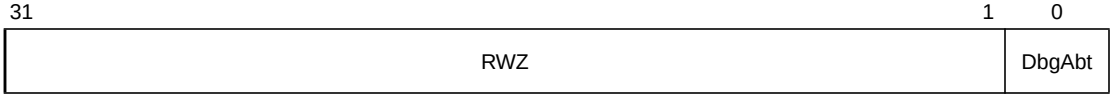
Access this data using SWI instructions when in the Thumb state, because the Thumb instruction set does not contain coprocessor instructions.

3.4.2 Debug Status Register (C2)

A debug monitor can use the CP14 Debug Status register when the ARM9E-S is in real-time debug mode.

The CP14 Debug Status register is essentially a one-bit read/write register with the format shown in Figure 3.20.

Figure 3.20 Coprocessor 14 Debug Status Register



RWZ **Reserved - Write Zero** **[31:1]**
These bits are reserved. Write zeros to these bits.

DbgAbt **Debug Abort** **0**
This bit indicates whether the processor took a Prefetch or Data Abort in the past because of a breakpoint or watchpoint.

This bit is set to 1 if the ARM9E-S core takes a Prefetch Abort as a result of a breakpoint or watchpoint. If on a particular instruction or data fetch, both the debug abort and external abort signals are asserted, the external abort takes priority and the DbgAbt bit is not set. You can read/write the DbgAbt bit using MRC/MCR instructions.

A typical use of this bit is by a real-time debug-aware abort handler. This handler examines the DbgAbt bit to determine whether the abort has been externally or internally generated. If the DbgAbt bit is set, the abort handler initiates communication with the debugger over the comms channel.

Chapter 4
Caches

This chapter describes the features and behavior of each of these blocks. It contains the following sections:

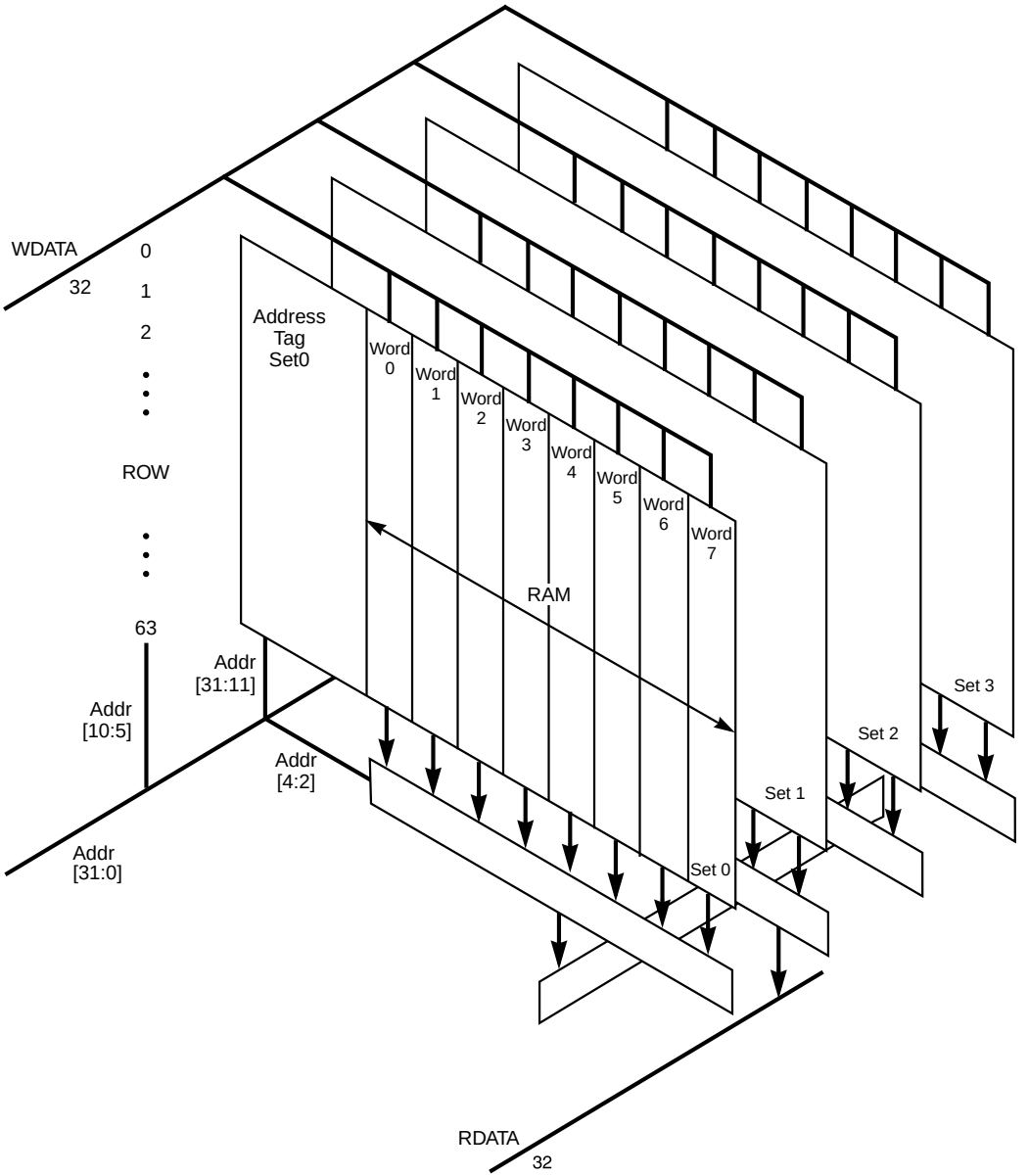
4.1 Cache Architecture

The ARM946E-S uses an Instruction Cache (I-cache) and a Data Cache (D-cache) to reduce the effective memory access time. You can tailor the cache size to suit your individual application, and you can set the I-cache and D-cache sizes independently. The ARM946E-S supports the following cache sizes:

- 0 Kbytes
- 4 Kbytes
- 8 Kbytes
- 16 Kbytes
- 32 Kbytes
- 64 Kbytes
- 128 Kbytes
- 256 Kbytes
- 512 Kbytes
- 1 Mbyte

The I-cache and D-cache are formed from synchronous SRAM, and have similar architectures. Figure 4.1 shows an example 8 Kbyte cache.

Figure 4.1 Example 8 Kbyte Cache



The I-cache and D-cache can be direct mapped, or they can be 2-way or 4-way set associative with a cache line length of 8 words (32 bytes). Each cache supports single-cycle read access.

Each cache set includes a tag RAM for storing the cache line address and a data RAM for storing the instructions or data.

During a cache access, all tag RAMs are accessed, and the tag address is compared with the access address. If a match (or cache hit) occurs, the data from that set is selected for return to the ARM9E-S core. If none of the Tags match (a cache miss), then external memory must be accessed, unless the write buffer is enabled and this access is a buffered write.

If a read access from a cacheable memory region misses, new data is loaded into one of the sets. This method is an allocate on read miss replacement policy. Selection of the set is performed by a set counter that can be clocked in a pseudo-random manner, or in a predictable manner based on the replacement algorithm selected.

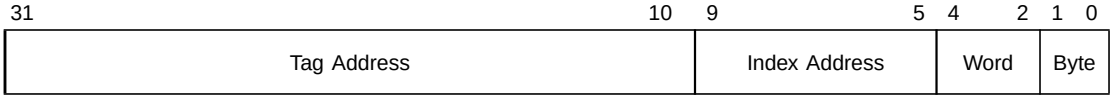
Critical or frequently accessed instructions or data can be locked into the cache by restricting the range of the replacement counter. You cannot replace locked lines. They remain in the cache until they are unlocked or flushed. The cache cannot be locked if it is direct mapped.

The cache access address from the ARM9E-S core has four parts:

- Byte Address (Addr[1:0])
- Word Address (Addr[4:2])
- Index Address (Addr[N:5])
- Tag Address (Addr[31:N+1])

For example, for a 4 Kbyte, 4-way set associative cache, the cache access address is as shown in Figure 4.2.

Figure 4.2 Access Address for a 4 Kbyte Cache



The size of the index and address Tags vary depending on the cache size. Table 4.1 shows how the index and tag sizes change for the cache sizes supported by the ARM946E-S.

Table 4.1 Tag and Index Fields for Supported Cache Sizes

Cache Size	Tag	Index
4 Kbytes	Addr[31:10]	Addr[9:5]
8 Kbytes	Addr[31:11]	Addr[10:5]
16 Kbytes	Addr[31:12]	Addr[11:5]
32 Kbytes	Addr[31:13]	Addr[12:5]
64 Kbytes	Addr[31:14]	Addr[13:5]
128 Kbytes	Addr[31:15]	Addr[14:5]
256 Kbytes	Addr[31:16]	Addr[15:5]
512 Kbytes	Addr[31:17]	Addr[16:5]
1 Mbyte	Addr[31:18]	Addr[17:5]

Table 4.1 is for a 4-way set associative cache. A direct-mapped cache requires two more address bits for the index and two less for the tag. A 2-way set associative cache uses one more bit for the index and one less for the tag.

Each entry in the tag RAM contains an Index and tag address (or cache line address) plus three additional bits that indicate the status or validity of the cache data associated with a given tag address. The three bits are:

- Valid Bit (bit 4)

  The valid bit is set when a cache line is written with valid data. Only a valid line can return a hit during a cache lookup. Upon reset, all valid bits are cleared.

- Dirty Bits (bits [3:2])

  The two dirty bits are associated with write operations in the D-cache. They indicate whether or not the data in a cache line differs from the data in external memory.

Note: Data can be marked as dirty only if it resides in a write back protection region.

4.2 I-Cache

The ARM946E-S has a direct-mapped, 2-way, or 4-way set-associative I-cache. You can choose the size of the I-cache from any of the supported cache sizes. The I-cache uses the physical address generated by the processor core. It uses a policy of *allocate on read-miss*, and is always reloaded one cache line (eight words) at a time, through the external interface.

4.2.1 Enabling and Disabling the I-Cache

To enable the I-cache, set bit 12 of the CP15 Control register. The cache is only enabled if the protection unit is already enabled, or if they are enabled simultaneously. When the I-cache is enabled, a cacheable read-miss places lines in the I-cache.

You can enable the I-cache and protection unit simultaneously with a single write to the CP15 control register, although you must program at least one protection region before you enable the protection unit.

You can lock critical or frequently accessed instructions into the I-cache.

4.2.2 I-Cache Operation

When enabled, the I-cache operation is also controlled by the Cacheable instruction (Ci) bit, which is stored in the Protection unit. This bit selectively enables or disables caching for different memory regions. The Ci bit affects I-cache operation as follows:

Successful Cache Read – Data is returned to the core only if the Ci bit is 1.

Unsuccessful Cache Read – If the Ci bit is 1, a line fetch of eight words is performed. The line fetch starts with the requested address aligned to an eight-word boundary (that is, the line fetch starts with word 0). If the Ci bit is 0, a single-word external access is performed to fetch the requested instruction. The cache is not updated.

Clearing bit 12 of the CP15 Control register disables the I-cache. This action prevents all I-cache look ups and line fills, and forces all instruction fetches to be performed as single external accesses.

4.2.3 I-Cache Validity

The ARM946E-S does not support external memory snooping. Therefore, if you write self-modifying code, the instructions in the I-cache can become incoherent with external memory. Similarly, if you reprogram the protection regions, code might exist in the cache that should be in a noncacheable region. In either of these cases, you must flush the I-cache.

4.2.4 I-Cache Flush

You can flush the entire I-cache by software in one operation, or you can flush individual cache lines by writing to the CP15 Cache Operations register (register 7). The I-cache is automatically flushed during reset. The I-cache never has to be cleaned because its only source of data is from external memory. (The ARM9E-S processor only performs reads from the I-cache, except during debug operations.)

Flushing the Entire Cache – As shown in Table 3.4 on page 3-22, you can flush the entire I-cache using an MCR instruction. In this case, the contents of the ARM register transferred to CP15 must be 0. You can use the following code to do this:

```
MOV   r0, #0            ; Clear r0
MCR   p15, r0, c7, c5, 0; Flush entire instruction cache
```

Note: The use of r0 is arbitrary.

Flushing the entire cache also flushes any locked-down code. If you want to preserve locked-down code, you must flush cache lines individually and avoid the locked-down lines.

Flushing a Single Cache Line – You can flush single cache lines. To do this, you must specify in Rd the address to be flushed from the cache. You can use the following code to do this:

```
      LDR r0, = FlushAddress; Load r0 with address
FlushAddress
      MCR p15, r0, c7, c5, 1; Flush single cache line
```

4.3 D-Cache

The ARM946E-S has a direct mapped, 2-way, or 4-way set-associative D-cache. You can choose the size of the D-cache from any of the supported cache sizes. The D-cache uses the physical address generated by the processor core. It uses an *allocate on read-miss* policy, and is always reloaded one cache line (eight words) at a time, through the external memory interface.

The Cacheable data (Cd) and Bufferable data (Bd) bits, which reside in the Protection Unit, control the behavior of the D-cache. For this reason, the Protection Unit must be enabled when the D-cache is enabled.

4.3.1 Enabling and Disabling the D-Cache

You can enable the D-cache by setting bit 2 of the CP15 control register. The cache is only enabled if the Protection Unit is already enabled, or is enabled simultaneously.

You can enable the D-cache and Protection Unit simultaneously with a single write to the CP15 control register, although you must program at least one protection region before you enable the protection unit.

To disable the D-cache, clear bit 2 of the CP15 control register.

The D-cache is automatically disabled and flushed on reset.

When the D-cache is disabled, cache searches are prevented. This marks all data accesses as noncacheable, forcing the ARM946E-S to perform external accesses. The write buffer control is still decoded from the Bd and Cd bits. The Cd bit is forced to 0 (noncacheable).

4.3.2 D-Cache Operation

When the D-cache is enabled, it is searched when the processor performs a load or store.

The D-cache supports both *write back* (WB) and *write through* (WT) modes. For data stores that hit in the D-cache in WB mode, the cache line is updated and the dirty bit is set for the associated cache half line. Setting the dirty bit indicates that the cache version of the data differs from external memory. In WT mode, a store that hits in the D-cache

causes the cache line to be updated. But the cache line is not marked as dirty, because the data store is also written to external memory (through the write buffer). This action keeps external memory consistent with the cache. In both WB and WT modes, a store that misses in the cache is sent to the write buffer. When a line fetch causes a cache line to be evicted from the D-cache, the dirty bit for each half of the line is read, and, if the half line contains valid and dirty data, it is written back to the write buffer before the line fill replaces it.

4.3.2.1 Cd Bit - Cache Loads and Stores

The Cd bit determines whether data being read must be placed in the D-cache and used for subsequent reads. Typically, main memory is marked as cacheable to reduce memory access time and therefore increase system performance. It is usual to mark input/output space as noncacheable. For example, if a processor is polling a memory-mapped register in input/output space, it is important that the processor is forced to read data direct from the peripheral, and not a copy of initial data held in the D-cache.

If the cache hits on a load, data is returned to the cache if the Cd bit is 1. If the cache read misses, the Cd bit is examined. Table 4.2 shows the function of the Cd bit.

Table 4.2 Cd Bit Function

Cd Bit Value	Function
1	Cacheable data area and protection unit enabled. A line fill of eight words is performed, and the data is written into a randomly chosen segment of the D-cache.
0	A single or multiple external access is performed and the cache is not updated.

Stores that hit in the cache update the cache line if the Cd bit is 1. Stores that miss the cache use the Cd and Bd bits to determine whether the write is buffered. A write miss is not loaded into the cache as a result of that miss.

4.3.2.2 Cd and Bd Bits - Cache Stores

The Bd and Cd bits affect writes that both hit and miss in the D-cache. If the Bd and Cd bits are both 1, the area of memory is marked as write back, and stores that hit in the D-cache only update the cache, not external memory. If the Bd bit is 0 and the Cd bit is 1, the area of memory is marked as write through, and stores that hit in the D-cache update both the cache and external memory.

4.3.2.3 Load and Store Multiples

Load and store multiples are divided at 4 Kbyte boundaries (the minimum protection region size), allowing a protection check to be performed in case the *Load Multiple* (LDM) or *Store Multiple* (STM) crosses into a region with different protection properties.

4.3.3 D-Cache Validity

The ARM946E-S does not support memory translation, so you can always consider the data in the D-cache as valid within the context of the ARM946E-S. However, if you use external memory translation and the mappings are changed, the D-cache is no longer consistent with external memory, and you must flush it.

The ARM946E-S does not support external memory snooping. Any shared data memory space, therefore, must not be cacheable. Additionally, if you reprogram the data protection regions, data already in the cache might now be in a noncacheable region, and you must flush it.

4.3.4 D-Cache Clean and Flush

The D-cache has flexible cleaning and flushing utilities that allow the following operations:

- You can invalidate the whole D-cache (*flush D-cache*) in one operation without writing back dirty data.
- You can invalidate individual lines without writing back any dirty data (*flush D-cache single entry*).
- You can perform cleaning on a line-by-line basis. The data is only written back through the write buffer when a dirty line is encountered, and the cleaned line remains in the cache (*clean D-cache single*

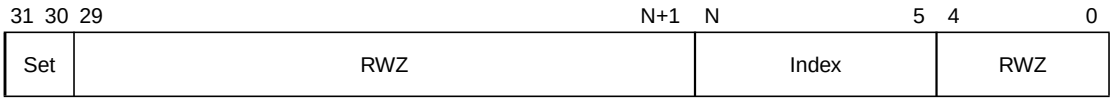
entry). You can clean cache lines using either their index within the D-cache or their address within memory.

- You can clean and flush individual lines in one operation (*clean and flush D-cache entry*). You can clean and flush individual lines using either their index within the D-cache or their address within memory.

You perform the cleaning and flushing operations using CP15 register 7, in a similar way to the I-cache.

The format of Rd transferred to CP15 for all register 7 operations is shown in Figure 4.3.

Figure 4.3 Register 7, Rd Format



The value of N is dependent on the cache size, as shown in Table 4.3.

Table 4.3 Calculating Index Addresses

Cache Size	Value of N
4 Kbytes	9
8 Kbytes	10
16 Kbytes	11
32 Kbytes	12
64 Kbytes	13
128 Kbytes	14
256 Kbytes	15
512 Kbytes	16
1 Mbyte	17

The value of N is derived from the following equation:

$$N = \log_2\left(\frac{\text{cache size}}{\text{number of sets} \times \text{line length in bytes}}\right) + 4$$

Where the number of sets multiplied by the line length in bytes is 128 for a 4-way associative cache. This would be 64 for a 2-way associative cache and 32 for a direct-mapped cache. The table above is correct for a 4-way associative cache. A direct-mapped or 2-way cache would have larger values for N.

It is usual to clean the cache before flushing it, so that external memory is updated with any dirty data. The following code shows how you can clean and flush the entire cache (assuming a 4 Kbyte D-cache).

```
    MOV     r1, #0                      ; Initialize set counter
outer_loop
    MOV     r0, #0                      ; Initialize line counter
inner_loop
    ORR     r2, r1, r0                  ; Generate set and line address
    MCR     p15, 0, r2, c7, c14, 2      ; Clean and flush the line
    ADD     r0, r0, #0x20               ; Increment to next line
    CMP     r0, #0x400                  ; Complete all entries in one
set?
    BNE     inner_loop                  ; If not branch back to inner_loop
    ADD     r1, r1, #0x40000000         ; Increment set counter
    CMP     r1, #0x0                    ; Complete all sets
    BNE     outer_loop                  ; If not branch back to outer_loop
```

4.4 Cache Lockdown

To provide predictable code behavior in embedded systems, a mechanism is provided for locking code into the I-cache and D-cache. For example, you can use this feature to hold high-priority interrupt routines where there is a hard real-time constraint, or to hold the coefficients of a DSP filter routine in order to reduce external bus traffic.

You can lockdown a region of the I-cache or D-cache by executing a short software routine, taking note of these requirements:

- The program must be held in a noncacheable area of memory.
- The cache must be enabled and interrupts must be disabled.
- Software must ensure that the code or data to be locked down is not already in the cache.
- If the caches have been used after the last reset, the software must ensure that the cache in question is cleaned, if appropriate, and then flushed.

You can carry out lockdown in the D-cache using CP15 register 9. I-cache lockdown uses both CP15 registers 7 and 9.

As described in Section 4.1, “Cache Architecture,” page 4-1, the ARM946E-S I-cache and D-cache each consist of from one to four sets. You can perform lockdown with a granularity of one set. The smallest space that you can lockdown is one set (one quarter of cache size). Lockdown starts at set zero, and can continue until only one set is left unlocked. At least one set must always be unlocked, so lockdown is not available for direct-mapped caches.

4.4.1 Locking Down the Caches

The procedures for locking down a set in the I-cache and D-cache are slightly different. In both cases you must:

1. Put the cache into lockdown mode by programming register 9.
2. Force a line fill.
3. Lock the corresponding data in the cache.

4.4.1.1 D-Cache Lockdown

For the D-cache, the lockdown procedure is as follows:

1. Write to CP15 register 9, setting LD = 1 (LD is bit 31, the load bit) and IDX[1:0] = 0 (IDX bits specify the cache set).
2. Initialize the pointer to the first of the words to be locked into the cache.
3. Execute an LDR from that location. This forces a line fill from that location, and the resulting eight words are captured in the cache.
4. Increment the pointer by 32 (number of bytes in a cache line).
5. Execute an LDR from that location. The resulting line fill is captured in the cache.
6. Repeat steps 4 and 5 until all words are loaded in the cache or one quarter of the cache has been loaded.
7. Write to CP15 register 9, setting LD = 0 and IDX[1:0] = 1.

If there is more data to lockdown, at the final step, the LD bit must remain set and the process repeated. The LD bit must only be cleared when all

the lockdown data has been loaded. The IDX[1:0] bits must be set to the next available set.

Note: The write to CP15 register 9 must not be executed until the line fill has completed. This is achieved by aligning the LDR to the last address of the line.

4.4.1.2 I-Cache Lockdown

For the I-cache, the lockdown procedure is as follows:

1. Write to CP15 register 9, setting LD = 1 (the load bit) and IDX[1:0] = 0 (the cache set bits).
2. Initialize the pointer to the first of the words to be locked into the cache.
3. Force a line fill from that location by writing to CP15 register 7 (I-cache preload).
4. Increment the pointer by 32 (number of bytes in a cache line).
5. Force a line fill from that location by writing to CP15 register 7. The resulting line fill is captured in the I-cache.
6. Repeat steps 4 and 5 until all words are loaded in the cache or one set of the cache has been loaded.
7. Write to CP15 register 9, setting LD = 0 and IDX[1:0] = 1.

If there are more instructions to lockdown, at the final step, the LD bit must remain set and the process repeated. The LD bit must only be cleared when all the lockdown instructions have been loaded. The IDX[1:0] bits must be set to the next available set.

The only significant difference between the sequence of operations for the D-cache and I-cache is that an `MCR` instruction must be used to force the line fill in the I-cache, instead of an LDR. The rest of the sequence is the same as for D-cache lockdown.

The MCR to perform the I-cache fetch is a CP15 register 7 operation:

```
MCR    p15, 0, Rd, c7, c13, 1
```

4.4.1.3 Example I-Cache Lockdown Subroutine

A subroutine that you can use to lockdown code in the I-cache is:

```
; Subroutine lock_i_cache
; r1 contains the start address
; r2 contains the end address
; Assumes that r2 - r1 fits within one cache set
; The subroutine performs a lockdown of instructions in the
; instruction cache
; It first reads the current lock_down index and then locks
; down the number of sets required
; Note - This subroutine must be located in a noncacheable
;        region of memory
;      - Interrupts must be disabled
;      - Subroutine must be called using the BL instruction
;      - r1-r3 can be corrupted in line with ARM/Thumb
;        Procedure Call Standards (ATPCS)
;      - Returns final I-Cache lockdown index in r0 if successful
;      - Returns 0xFFFFFFFF in r0 if an error occurred

lock_I_cache
    BIC     r1, r1, #0x7f                       ;Align address to cache line
    MRC     p15, 0, r3, c9, c0, 1               ;Get current I-Cache index
    AND     r3, r3, #0x3                        ;Mask unwanted bits
    CMP     r3, #0x3                            ;Check for available set
    BEQ     error                               ;If no sets available,
                                                ;generate an error
    ORR     r3, r3, #0x8000000                  ;Set the lockdown bit
    MCR     p15, 0, r3, c9, c0, 1               ;Write lockdown register

lock_loop
    MCR     p15, 0, r1, c7, c13, 1              ;Force an instruction fetch
                                                ;from address r1
    ADD     r1, r1, #0x20                       ;Increment address by a
                                                ;cache line length
    CMP     r2, r1                              ;Reached our end address yet?
    BLT     lock_loop                           ;If not, repeat loop
    ADD     r3, r3, #0x1                        ;Increment I-Cache index
    BIC     r0, r3, #0x8000000                  ;Clear lockdown bit and
                                                ;Write index into r0
    MCR     p15, 0, r3, c9, c0, 1               ;Write lockdown register
    MOV     pc, lr                              ;Return from subroutine

error
    MVN     r0, #0                              ;Move 0xFFFFFFFF into r0
    MOV     pc, lr                              ;Return from subroutine
```

Chapter 5
Protection Unit

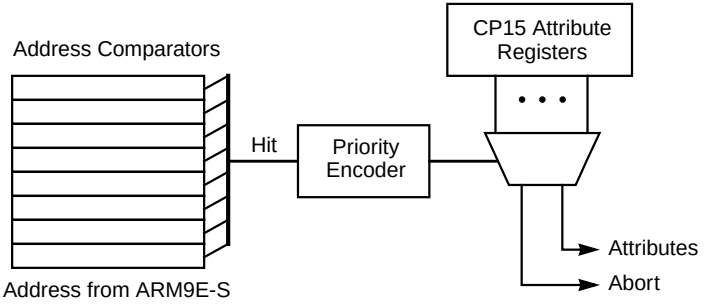
This chapter describes the ARM946E-S Protection Unit. It contains the following sections:

5.1 About the Protection Unit

The protection unit allows you to partition memory and set individual protection attributes for each protection region. You can divide the address space into eight regions with different sizes for each region.

Figure 5.1 shows a simplified block diagram of the Protection Unit.

Figure 5.1 ARM946E-S Protection Unit



The Protection Unit is programmed using CP15 registers 1, 2, 3, 5, and 6 (see Section 3.3, “CP15 Registers,” page 3-2).

5.2 Enabling the Protection Unit

Before the Protection Unit is enabled, you must program at least one valid protection region. Otherwise, the ARM946E-S could enter a state that requires using reset to recover.

To enable the Protection Unit, set bit 0 of CP15 Register 1, the Control register.

When the Protection Unit is disabled, all instruction fetches are noncacheable, and all data accesses are noncacheable and nonbufferable.

5.3 Memory Regions

You can partition the address space into a maximum of eight regions. Each region has the following specifications:

- Region base address
- Region size
- Cache and write buffer configuration
- Read and write access permissions

The ARM architecture uses constants known as *inline literals* to perform address calculations. These constants are automatically generated by the assembler and compiler and are stored inline with the instruction code. To ensure correct operation, you must define an area of memory from where code is to be executed that allows both data and instruction accesses.

The base address and size properties are programmed using CP15 register 6. For the format and bit descriptions of this register, refer to Section 3.3.9, “Protection Region/Base Size (PR/BS) Registers (6),” page 3-19.

5.3.1 Region Base Address

The base address defines the start of the memory region. You must align this to a region-sized boundary. For example, if a region size of 8 Kbytes

is programmed for a given region, the base address must be a multiples of 8 Kbytes.

Note: If the region is not aligned correctly, it causes unpredictable behavior.

5.3.2 Region Size

The region size is specified as a five-bit value, encoding a range of values from 4 Kbytes to 4 Gbytes. For a detailed list of the bit encodings, refer to page 3-20.

5.3.3 Partition Attributes

Each region has a number of attributes associated with it. These control how a memory access is performed when the processor core issues an address that falls within a given region. The attributes are:

- Cacheable
- Bufferable (for data regions only)
- Read/write permissions

To specify this information, program CP15 registers 2, 3, and 5 (see Chapter 3, “Programmer’s Model”). If an access fails its protection check (for example, if a User mode application attempts to access a *Privileged mode access only* region), a memory abort occurs. The processor enters the abort exception mode, branching to the Data Abort or Prefetch Abort vector accordingly.

The cacheable and bufferable bits in CP15 registers 2 and 3 are used together to select one of four cache and write buffer configurations. These are described in Section 7.5, “Write Buffer,” page 7-10.

5.4 Overlapping Regions

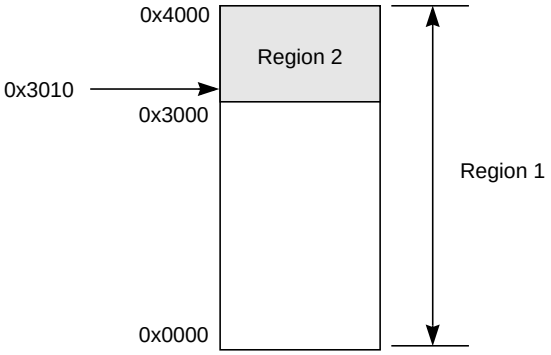
You can program the Protection Unit with two or more overlapping regions. When overlapping regions are programmed, a fixed priority scheme is applied to determine the overlapping region attribute that is applied to the memory access. Attributes for region 7 have the highest priority, and those for region 0 have the lowest priority.

For example, if region 1 and 2 are programmed as follows:

Region	Programmed For:
2	- 4 Kbyte size, starting from 0x3000 - Access permission bits, Dap[3:0] = 0b0010 - Privileged mode has full access - User mode has read only access
1	- 16 Kbyte size, starting from 0x0000 - Access permission bits, Dap[3:0] = 0001 - Privileged mode access only

When the processor performs a data load from address 0x3010 while in User mode, the address falls within both Region 1 and 2 (see the shaded area in Figure 5.2). Since there is a conflict, the attributes associated with Region 2 apply. This situation causes a Data Abort to occur, because in User mode only reads are allowed from Region 2.

Figure 5.2 Overlapping Memory Regions



5.5 Background Regions

Overlapping regions increase the flexibility of how the eight regions can be mapped onto physical memory devices in the system. You can also use the overlapping properties to specify a background region. For example, you might have a number of physical memory areas sparsely distributed across the 4 Gbyte address space. If a programming error

occurs therefore, it might be possible for the processor to issue an address that does not fall into any defined region.

If the address issued by the processor falls outside any of the defined regions, the ARM946E-S protection unit is hardwired to abort the access. To override this behavior, program region 0 to be a 4 Gbyte background region. In this way, if the address does not fall into any of the other seven regions, the access is controlled by the attributes you have specified for region 0.

Chapter 6
Tightly Coupled SRAM

This chapter describes the tightly coupled SRAM in the ARM946E-S. It contains the following sections:

- Section 6.1, “ARM946E-S SRAM Requirements”
- Section 6.2, “Using CP15 Control Register”

For details of the ARM9E-S interface signals referenced in this chapter, see the *ARM9E-S Technical Reference Manual*.

6.1 ARM946E-S SRAM Requirements

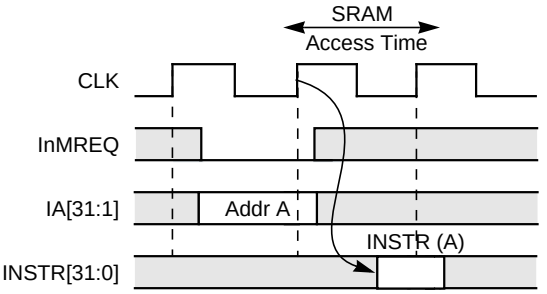
The ARM946E-S tightly coupled SRAM is built using compiled SRAM blocks from an ASIC library. The instruction SRAM (I-SRAM) and data SRAM (D-SRAM) can differ in size, and they can be any size the Protection Unit supports, from 0 bytes to 1 Mbyte. However, to ease implementation, the size must be an integer power of two.

ARM946E-S supports synchronous SRAM for the tightly coupled SRAM. The memory must be capable of returning data to the ARM9E-S processor core in a single cycle. This requirement applies to both the I-SRAM and D-SRAM.

To initialize the I-SRAM and to access literal tables during execution, the ARM9E-S processor core data interface requires I-SRAM access. To provide this access, the ARM946E-S multiplexes the instruction and data addresses before they enter the I-SRAM, and routes instruction data to both the instruction and data interfaces of the core. See Figure 1.1 on page 1-3 for details of this data and address multiplexing.

Figure 6.1 shows a typical read cycle (I-SRAM shown).

Figure 6.1 SRAM Read Cycle



The I-SRAM is located at address 0x00000000 in the memory map. Using this location simplifies the implementation of the design by removing the need for complex address comparators on both the instruction and data interfaces of the ARM9E-S core to generate the chip select logic for the SRAM. Fixing the SRAM location at 0x0 allows an address decode to control the chip selects for greater power efficiency.

6.2 Using CP15 Control Register

Except during reset, the CP15 control register controls the behavior of the tightly coupled SRAM.

6.2.1 Enabling the I-SRAM

To enable the I-SRAM, set bit 18 of the CP15 Control register to 1. To preserve the bits that are not being modified, you must use *read-modify-write* when accessing this register. See Section 3.3.5, “Control Register (1)” on page 3-11 for details of how to read and write the CP15 control register.

After the I-SRAM is enabled, all ARM9E-S instruction fetches and data accesses to the I-SRAM address space access the I-SRAM.

Enabling the I-SRAM greatly increases the performance of the ARM946E-S, because most accesses to the I-SRAM occur without stall cycles. Accessing the AHB, however, can cause several stall cycles for *each* access.

Note: Make sure you initialize the I-SRAM before attempting to enable or use it. Otherwise, behavior is unpredictable.

6.2.2 Disabling the I-SRAM

To disable the I-SRAM, clear bit 18 of the CP15 control register. See Section 3.3.5, “Control Register (1),” page 3-11 for details of how to read and write the CP15 control register.

After you disable the I-SRAM, all ARM9E-S instruction fetches access the AHB.

Note: The contents of the SRAM are preserved when it is disabled. If it is re-enabled, accesses to previously initialized SRAM locations return the preserved data.

6.2.3 I-SRAM Load Mode

You must initialize the I-SRAM with the required code image before executing from the I-SRAM.

To initialize the I-SRAM, write a 1 to it from the ARM9E-S processor core data interface.

Using the I-SRAM Load Mode allows you to initialize the I-SRAM more efficiently. With load mode, you can directly copy data into an I-SRAM location from the corresponding address in the data cache or external memory.

When the I-SRAM Load Mode bit of CP15 Register 1 is set to 1, it inhibits reads from the I-SRAM. This action forces I-SRAM reads to access either external main memory or the data cache. Writes to the I-SRAM address range are not affected when instruction load mode is set.

The procedure for initializing the I-SRAM using the load mode is as follows:

1. Enable the I-SRAM and instruction load mode.
2. Load ARM registers from main memory, data cache, or data RAM.
3. Store ARM registers into I-SRAM.
4. Increment address pointers and repeat load/store steps until the code image is copied.

A suggested assembler code sequence for this procedure is shown below:

```
MOV R0, #0                ; Initialize pointer
LDR R1, =ImageTop         ; Define end of code image
MRC p15, 0, R2, c1, c0, 0 ; Read Control Register
ORR R2, R2, #&C0000
MCR p15, 0, R2, c1, c0, 0 ; Enable Instruction RAM and Load Mode CopyLoop
LDMIA R0, {R2 - R9}       ; Load 8 registers from main memory
STMIA R0!, {R2 - R9}      ; Store 8 regs into instruction SRAM
CMP R1, R0                ; Check if limit reached
BGT CopyLoop              ; Repeat if more to do
```

The read (LDMIA) accesses external memory or the data cache, and the write (STMIA) updates the tightly coupled I-SRAM.

Do not use SWP or SWPB to access I-SRAM addresses while in load mode. Doing this produces unpredictable results.

6.2.4 Enabling and Disabling the D-SRAM

To enable the D-SRAM, set bit 16 of the CP15 Control register. See Section 3.3, “CP15 Registers,” page 3-2 for details of how to read and write this register. After the D-SRAM is enabled, all read and write accesses to the D-SRAM address space access the D-SRAM.

To disable the D-SRAM, clear bit 16 of the CP15 control register. After you disable the D-SRAM, all reads and writes to the D-SRAM address space access the AHB.

Reads and writes to D-SRAM address space either use the D-SRAM or access the AHB depending on whether D-SRAM is enabled or not.

For more information, see Section 3.3.12, “Tightly Coupled Memory Region Registers (9),” page 3-26.

6.2.5 D-SRAM Load Mode

You must initialize the D-SRAM with the required data image before use.

To initialize the D-SRAM, write to it from the ARM9E-S processor core data interface.

The D-SRAM load mode allows you to initialize the D-SRAM more efficiently. With load mode, you can copy into the D-SRAM directly from the corresponding address in the data cache or external memory.

When the D-SRAM Load Mode bit of CP15 Register 1 is set to 1, it inhibits reads from the D-SRAM. This action forces reads to access either main memory or the data cache. The Load Mode bit does not affect writes to the D-SRAM address range.

The procedure for initializing the D-SRAM using the load mode is as follows:

1. Enable the D-SRAM and data load mode.
2. Load ARM registers from main memory or data cache.
3. Store ARM registers into D-SRAM.
4. Increment address pointers and repeat load/store steps until the data image is copied.

A suggested assembler code sequence for this procedure is shown below:

```
    LDR R0, #ImageStart        ; Initialized pointer
    LDR R1, =ImageTop          ; Define end of data space
    MRC p15, 0, R2, c1, c0, 0  ; Read Control Register
    ORR R2, R2, #&30000
    MCR p15, 0, R2, c1, c0, 0  ; Enable Data RAM and Load Mode
CopyLoop
    LDMIA R0, {R2 - R9}        ; Load 8 registers from main memory
    STMIA R0!, {R2 - R9}       ; Store 8 regs into instruction SRAM
    CMP R1, R0                 ; Check if limit reached
    BGT CopyLoop               ; Repeat if more to do
```

The read (LDMIA) accesses external memory or the data cache, and the write (STMIA) updates the tightly coupled D-SRAM.

Do not use SWP or SWPB to access D-SRAM addresses while in load mode. Doing this produces unpredictable results.

SWP and SWPB operations to the D-SRAM while it is in load mode produce unpredictable results. The read accesses external memory or the data cache, and the write updates the D-SRAM.

Do not perform SWP or SWPB operations to locations in the I-SRAM address space while the I-SRAM is in load mode.

Chapter 7
Bus Interface Unit and Write Buffer

This chapter describes the ARM946E-S Bus Interface Unit (BIU) and write buffer. It contains the following sections:

7.1 About the BIU and Write Buffer

The ARM946E-S supports an *Advanced Microprocessor Bus Architecture* (AMBA) *Advanced High-performance Bus* (AHB) interface. The AHB is a new generation of AMBA interface that addresses the requirements of high-performance synthesizable designs, including:

- Single clock edge operation (rising edge)
- Unidirectional (non 3-state) buses
- Burst transfers
- Split transactions
- Single-cycle bus master handover

See the *AMBA Rev 2.0 AHB Specification* for details about this bus architecture.

The ARM946E-S BIU implements a fully compliant AHB bus master interface and incorporates a write buffer to increase system performance. The BIU links external memory with the ARM9E-S processor core, the caches, and the tightly coupled SRAMs. External memory is accessed

through the AHB interface for cache line fills and for initializing the tightly coupled SRAMs. The AHB interface is also used to access code and data that are not within the cacheable or tightly coupled memory address regions.

When an AHB access occurs, the BIU and system controller handshake to ensure that the ARM9E-S processor core is stalled until the access is finished. If you are using the write buffer, you might be able to allow the processor core to continue program execution. The BIU controls the write buffer and related stall behavior.

7.2 AHB Bus Master Interface

The ARM946E-S implements a fully compliant AHB bus master interface as defined in the *AMBA Specification, Rev 2.0*. See this document for a detailed description of the AHB protocol.

7.2.1 About the AHB

The AHB architecture is based on separate cycles for address and data rather than separate clock phases, as in the AMBA Advanced System Bus (ASB). The address and control for an access are broadcast from the rising edge of HCLK in the cycle before the data is expected to be read or written. During this data cycle, the address and control for the next transfer are driven out, providing a fully pipelined address architecture.

When an access is in its data cycle, a slave can extend an access by driving the HREADY signal LOW. This action stretches the current data cycle, and the pipelined address and control for the next transfer also stretches. With this system, all AHB masters and slaves sample HREADY on the rising edge of HCLK to determine when an access is complete and whether a new address can be sampled or driven out.

7.2.2 ARM946E-S Transfer Descriptions

The ARM946E-S supports three of the four possible transfer types defined in the *AMBA Specification*. The supported transfer types are:

IDLE HTRANS[1:0] = 00

NONSEQ HTRANS[1:0] = 10

SEQ HTRANS[1:0] = 11

Note: The ARM 946E-S does not support the Busy transfer type, where HTRANS[1:0] = 01.

7.2.3 Burst Sizes

The ARM946E-S supports the burst types listed in Table 7.1.

Table 7.1 Supported Burst Types

Burst Type	HBURST[2:0] Encoding	Use
SINGLE	000	Single writes (STR/STRH/STRB) Uncached single reads Uncached instruction fetches
INCR	001	Store multiple (STM) Uncached burst reads (LDM)
INCR4	011	Dirty half-cache line write back
INCR8	101	Dirty cache line write back Cache line fetches

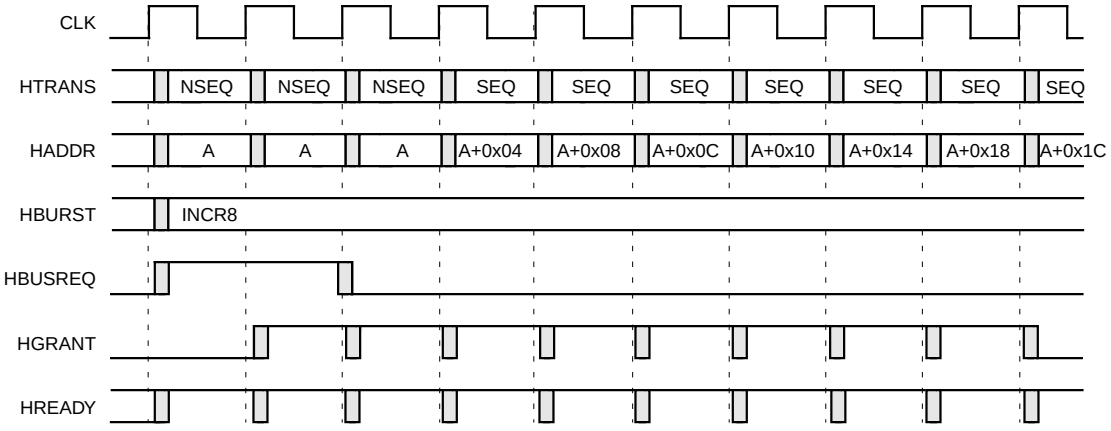
Incrementing bursts have an address increment of four (that is, a word increment).

7.2.4 Line Fetch Transfers

The ARM946E-S is optimized to run with both the I-cache and D-cache enabled. If a memory request (either instruction or data) to a cacheable area misses in the cache, the ARM946E-S does a line fetch.

A line fetch transfer is shown in Figure 7.1.

Figure 7.1 Line Fetch Transfer

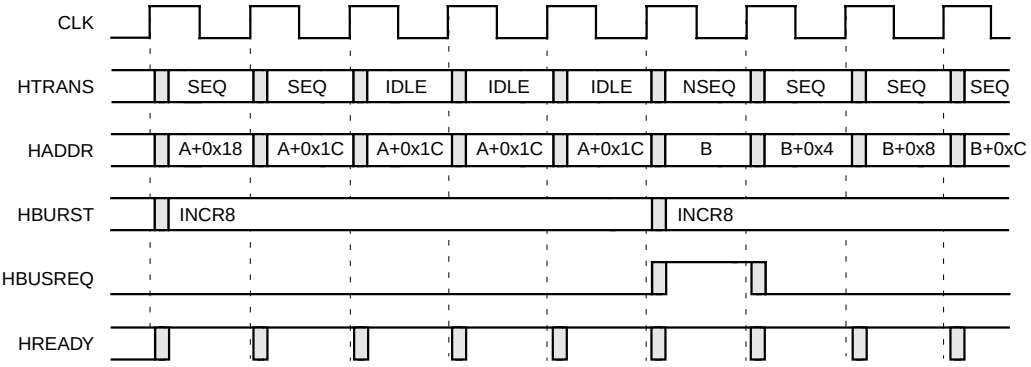


A line fetch is a fixed length burst of eight words. The start address of a line fetch is aligned to an eight-word boundary. The ARM946E-S asserts the bus request HBUSREQ until the arbiter grants the AHB bus (HGRANT asserted). The bus request is then negated. This method allows optimum system performance as the arbiter can accurately predict the end of the defined burst length.

7.2.5 Back-to-Back Line Fetches

The ARM946E-S supports streaming of data and instructions (core execution is advanced during the line fetch). To allow for cache look ups when crossing a cache line boundary, the ARM946E-S must insert IDLE cycles onto the AHB bus. The effect of this is shown in Figure 7.2. It is assumed in Figure 7.2 that HGRANT is asserted throughout, and that the HCLK frequency is the same as CLK.

Figure 7.2 Back-to-Back Line Fetches

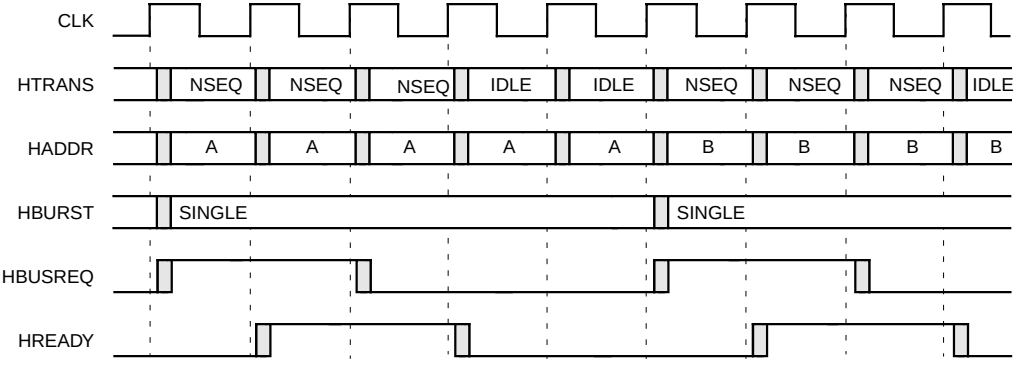


7.2.6 Uncached Transfers

If a memory request is made to an uncacheable region or the ARM946E-S cache is not enabled, the memory requests are serviced by the AHB interface. Sequential instruction fetches are treated as nonsequential reads.

Figure 7.3 shows uncached instruction fetches. Nonsequential uncached data operations have similar bus timing.

Figure 7.3 Nonsequential Uncached Accesses

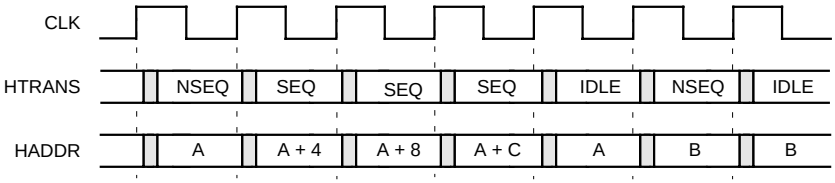


7.2.7 Burst Accesses

The AHB handles uncached burst operations (STM/LDM) as incrementing bursts of undefined length on the AHB.

Figure 7.4 shows a data burst followed by an uncached instruction fetch.

Figure 7.4 Data Burst Followed by Instruction Fetch

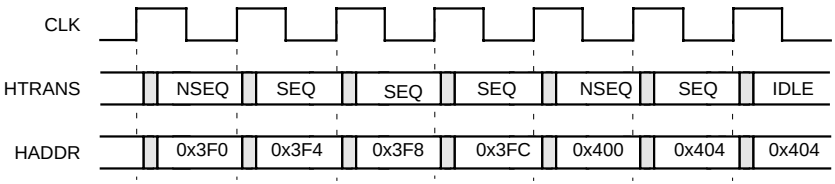


7.2.8 Bursts Crossing 1 Kbyte Boundary

The AHB specification requires that bursts must not continue across a 1 Kbyte boundary. Line Fetches and cache line write backs cannot cross a 1 Kbyte boundary because the start address is aligned to either a four- or eight-word boundary, and the burst length is fixed.

Uncached data bursts can cross a 1 Kbyte boundary (see example in Figure 7.5). The burst is restarted by inserting a nonsequential transfer as the boundary is crossed.

Figure 7.5 Crossing a 1 Kbyte Boundary



7.3 Noncached Thumb Instruction Fetches

The AHB interface performs Thumb instruction fetches as 32-bit accesses. To minimize bus loading, AHB transfers are only performed for nonsequential addresses and for sequential addresses that cross a word boundary. The word returned from main memory is latched, so both halfwords are available for the processor core.

7.4 AHB Clocking

The ARM946E-S design uses a single rising-edge clock (CLK) to time all internal activity. Some systems in which the ARM946E-S is embedded might need to run the AHB at a lower rate. To support this requirement, the ARM946E-S requires a clock enable (HCLKEN) to time AHB transfers.

The HCLKEN input is driven HIGH coincident with a rising edge of the ARM946E-S CLK. This action indicates that this particular rising-edge is also an HCLK rising-edge. HCLK must be synchronous with the ARM946E-S CLK.

When the ARM9E-S processor is running from tightly coupled SRAM or performing writes using the write buffer, the ARM946E-S HCLKEN and HREADY inputs are not used to stall the ARM9E-S processor. The processor only stalls when there are SRAM stall cycles or if the write buffer overflows. This means that the ARM9E-S is executing instructions at the faster CLK rate and is effectively decoupled from the HCLK domain.

However, when an AHB read access or unbuffered write occurs, the core does stall until the AHB transfer is completed. While the lower rate HCLK clocks the AHB system, HCLKEN is examined to determine when to drive out the AHB address and control that start an AHB transfer. HCLKEN then must detect the next rising edge of HCLK, so the BIU knows when the access is complete.

If the slave being accessed at the HCLK rate has a multicycle response, the HREADY input to the ARM946E-S is driven LOW until the data is ready to return. The BIU must do a logical AND on the HREADY response with HCLKEN to detect when the AHB transfer is completed. When the transfer is completed, SYSCLKEN is reasserted and enables the processor.

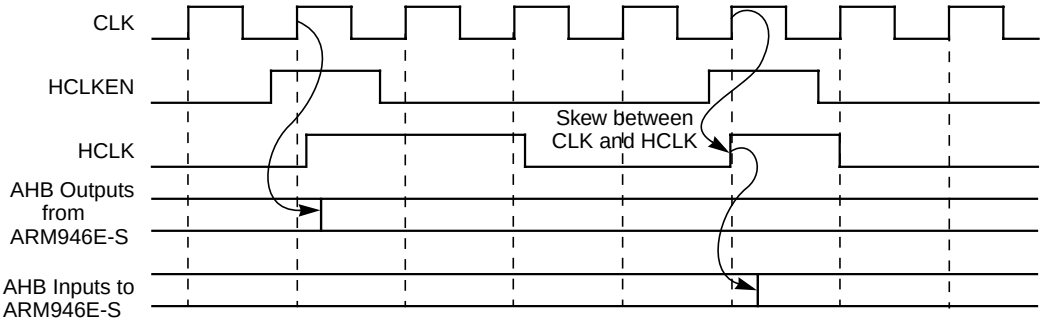
Note: When an AHB access is required, the processor core is stalled until the next HCLKEN pulse is received. The processor is stalled twice, once before it starts the access and while waiting for the access to finish. The stall before the start of the access is a synchronization penalty, and the worst case can be expressed in CLK cycles as the HCLK-to-CLK ratio minus 1.

7.4.1 CLK-to-HCLK Skew

The ARM946E-S drives out the AHB address on the rising edge of CLK when the HCLKEN input is TRUE. The AHB outputs therefore have output hold and delay values relative to CLK. However, these outputs are used in the AHB system where transfers are timed using HCLK. Similarly, inputs to the ARM946E-S are timed relative to HCLK but are sampled within the ARM946E-S with CLK. This leads to hold-time issues, from CLK to HCLK on outputs, and from HCLK to CLK on inputs. To minimize this effect, you must minimize the skew between HCLK and CLK.

Figure 7.6 shows the AHB clock relationships.

Figure 7.6 AHB Clock Relationships

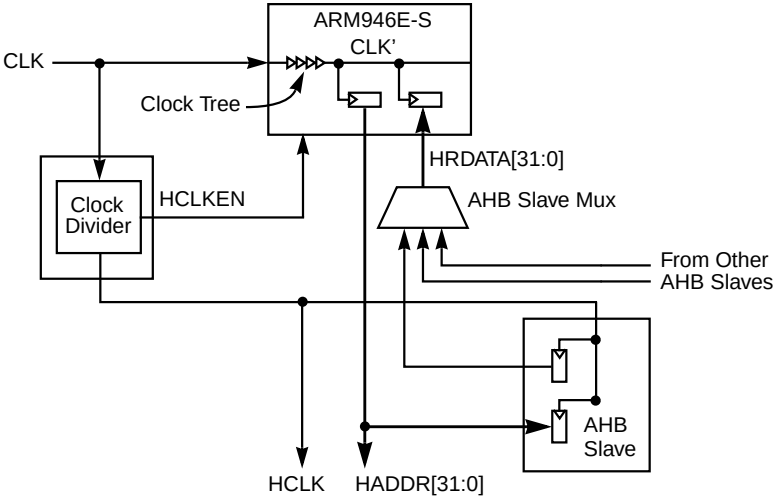


7.4.1.1 Clock Tree Insertion at Top Level

To ensure the clock is evenly distributed to all registers in the design, the ARM946E-S requires insertion of a clock tree. The registers that drive AHB outputs and sample AHB inputs are timed off CLK at the bottom of the inserted clock tree and subject to the clock tree insertion delay. To maximize performance, when the ARM946E-S is embedded in an AHB system, the HCLK clock generation logic must be constrained so it matches the insertion delay of the ARM946E-S clock tree. You can achieve this using a clock tree insertion tool provided you insert the clock tree in both the ARM946E-S and the embedded system at the same time (top level insertion).

Figure 7.7 shows an example of an AHB slave connected to the ARM946E-S.

Figure 7.7 ARM946E-S CLK to AHB HCLK Sampling



In Figure 7.7, the slave peripheral has input setup and hold times and output hold and valid times relative to HCLK. The ARM946E-S has input setup and hold times and output hold and valid times relative to CLK', which is the clock at the bottom of the clock tree. For optimal performance, use clock tree insertion to position HCLK to match CLK'.

7.4.1.2 Hierarchical Clock Tree Insertion

If you perform clock tree insertion on the ARM946E-S before it is embedded, you can add buffers on input data to match the clock tree, so that the setup and hold times are relative to the top-level CLK. This is safe at the expense of extra buffers in the data input path.

The HCLK domain AHB peripherals must still meet the ARM946E-S input setup and hold requirements. Since the ARM946E-S inputs and outputs are now relative to CLK, the outputs appear comparatively later by the value of the insertion delay. This ultimately leads to lower AHB performance.

7.5 Write Buffer

The ARM946E-S provides a write buffer to improve system performance. The write buffer has a 16-entry FIFO. Each entry can be either address or data. The type of entry is determined by the setting of an address/data flag. Each address entry is tagged with the transfer size, as indicated by the ARM9E-S core (byte, halfword, or word).

Write buffer behavior is controlled by the protection region attributes of the store being performed and the D-cache and Protection Unit enable status. This control is indicated by the Cacheable data bit (Cd) and the Write Buffer Control bit (Bd) from the Protection Unit.

The state of the Cd bit is based on three factors: the cacheable attribute for the particular protection region, the D-cache enable, and the Protection Unit enable.

The state of the Bd bit is based on two factors: the bufferable attribute for the particular protection region and the Protection Unit enable.

All accesses are initially noncacheable and nonbufferable until you have programmed and enabled the Protection Unit. You cannot use the write buffer while the Protection Unit is disabled.

On reset, all entries in the write buffer are invalidated.

7.5.1 Write Buffer Operation

The write buffer is used when the D-cache hits and/or misses, depending on the mode of operation. Table 7.2 shows how the Cd and Bd bits control the behavior of the write buffer.

Table 7.2 Data Write Modes

Cd	Bd	Access Mode	Description
0	0	NCNB (noncacheable, nonbufferable)	Data reads and writes are not cached, and they can be aborted externally. Writes are not buffered, so the processor is stalled until the external access is performed. NCNB reads bypass the write buffer.
0	1	NCB (noncacheable, bufferable)	Data reads and writes are not cached. Writes are buffered, and so they cannot be aborted externally. Reads can be aborted externally. Reads cause the write buffer to drain. If the D-cache hits for this type of access, there has been a programming error. D-cache hits are ignored, and the D-cache line is not updated for a read. Swap instruction operations on data in an NCB region are made to perform NCNB type accesses and are *not* buffered.
1	0	WT (write through)	Searches the D-cache for reads and writes. Reads that miss in the D-cache cause a line fill. Reads that hit in the D-cache do not perform an external access. All writes are buffered, regardless of whether they hit or miss in the D-cache. Writes that hit in the D-cache update the cache, but do not mark the cache line as dirty, because the write is also sent to the write buffer. Writes cannot be externally aborted. D-cache line fills cause the write buffer to drain before the line fill starts.
1	1	WB (write back)	Searches the D-cache for reads and writes. Reads that miss in the D-cache cause a line fill. Reads that hit in the D-cache do not perform an external access. Writes that miss in the D-cache are buffered. Writes that hit in the D-cache update the cache line, mark it as dirty, and do not send the data to the write buffer. D-cache write-backs are buffered. Writes (write-miss and write-back) cannot be externally aborted. D-cache line fills cause the write buffer to drain before the line fill starts.

7.5.2 Enabling and Disabling the Write Buffer

You cannot directly enable or disable the write buffer. However, you can prevent the write buffer from being used by setting the properties of a memory region to NCNB, or by disabling the Protection Unit.

7.5.3 Using Self-Modifying Code

Instruction fetches and NCNB reads bypass the write buffer. If you write self-modifying code to a bufferable or cacheable region, then it is essential that you drain the write buffer before fetching instructions from these addresses.

Chapter 8
External Coprocessor Interface

This chapter describes the ARM946E-S pipelined external coprocessor interface. It contains the following sections:

8.1 About the External Coprocessor Interface

The ARM946E-S fully supports the connection of on-chip coprocessors through an external coprocessor interface. All types of coprocessor instructions are supported. For a description of all the interface signals referred to in this chapter, see the *ARM9E-S Technical Reference Manual*.

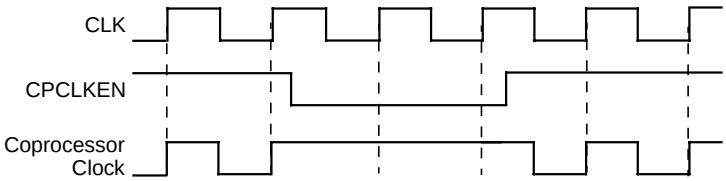
Coprocessors determine the instructions they must execute using a *pipeline follower* in the coprocessor. As each instruction arrives from memory, it enters both the ARM9E-S pipeline and the coprocessor pipeline. To avoid being a critical path for the instruction, the coprocessor pipeline operates one clock cycle behind the ARM9E-S pipeline. However, there is a mechanism inside the ARM946E-S that stalls the ARM9E-S pipeline, so the external coprocessor pipeline can catch up.

For that reason, consider the two pipelines synchronized. The ARM9E-S processor informs the coprocessor when instructions move from decode to execute, and whether the instruction must be executed or not.

To enable coprocessors to continue executing data operations while the ARM9E-S pipeline is stalled (for example, when waiting for a cache line fill to occur), the coprocessor receives a clock (CLK) and a clock enable signal (CPCLKEN).

If CPCLKEN is LOW on the rising edge of CLK, then the ARM9E-S pipeline is stalled and the coprocessor pipeline must not advance. Figure 8.1 indicates the timing for these signals and when the coprocessor pipeline must advance its state.

Figure 8.1 Coprocessor Clocking



The Coprocessor clock is the result of ORing CLK with the inverse of CPCLKEN. This is one technique for generating a clock that reflects the ARM9E-S core pipeline advancing.

8.2 Coprocessor Instructions

There are three classes of coprocessor instructions:

LDC/STC Load from memory to coprocessor or store from coprocessor to memory

MCR/MRC Register transfer between coprocessor and ARM processor core

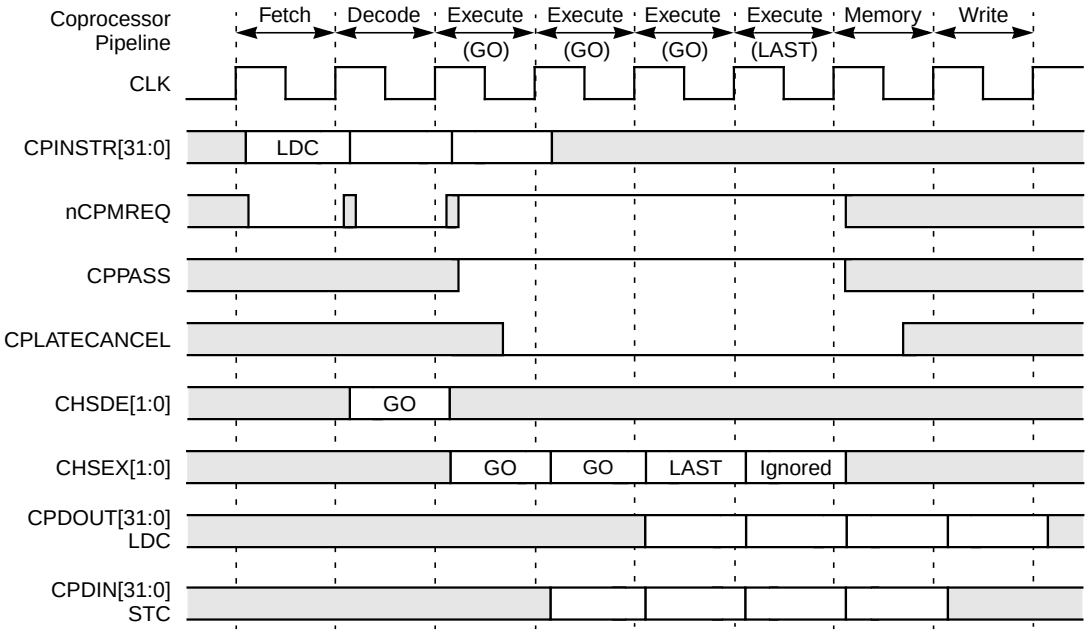
CDP Coprocessor data operation

8.3 LDC/STC Instructions

The Load/Store Coprocessor from/to Memory (LDC/STC) instructions are used to transfer data to and from external coprocessor registers and memory. For the ARM946E-S, the memory can be either internal memory (cache or tightly coupled SRAM) or the AHB depending on the address range of the access and the Protection Unit settings.

Figure 8.2 shows the cycle timing for the LDC/STC operations.

Figure 8.2 LDC/STC Cycle Timing



In the example shown in Figure 8.2, four data words are transferred. What the coprocessor drives on the CHSDE[1:0] and CHSEX[1:0] buses determines the number of words transferred.

As with all other instructions, the ARM9E-S performs the main decode off the rising edge of the clock during the Decode stage. From this, the core commits to executing the instruction, so it does an instruction fetch. The coprocessor instruction pipeline keeps in step with the ARM9E-S processor by monitoring nCPMREQ. This is a registered version of the ARM9E-S instruction memory request signal (InMREQ).

At the rising edge of CLK, if CPCLKEN is HIGH and nCPMREQ is LOW, an instruction fetch is taking place. On the next rising edge of the clock, when CPCLKEN is HIGH, the coprocessor instruction bus (CPINSTR[31:0]) contains the fetched instruction.

In this case, the following occurs:

1. The last instruction fetched enters the decode stage of the coprocessor pipeline.
2. The instruction in the decode stage of the coprocessor pipeline enters its execute stage.
3. The fetched instruction is sampled.

In all other cases, the ARM9E-S pipeline is stalled, and the coprocessor pipeline does not advance.

During the execute stage, the condition codes are compared with the status flags to determine whether the instruction can actually execute. The output CPPASS is asserted (HIGH) if the instruction in the execute stage of the coprocessor pipeline:

- Is a coprocessor instruction
- Has passed its condition codes

If a coprocessor instruction busy-waits, CPPASS is asserted on every cycle until the coprocessor instruction is executed. If an interrupt occurs during busy-waiting, CPPASS is driven LOW, and the coprocessor stops execution of the coprocessor instruction.

Another output, CPLATECANCEL, cancels a coprocessor instruction when the instruction preceding it causes a Data Abort. This output is valid on the rising edge of CLK on the cycle that follows the first execute cycle of the coprocessor instruction. This is the only cycle in which CPLATECANCEL can be asserted.

On the rising edge of the clock, the ARM9E-S processor examines the coprocessor handshake signals, CHSDE[1:0] or CHSEX[1:0], based on the following criteria:

- If a new instruction is entering the execute stage in the next cycle, it examines CHSDE[1:0].

- If the currently executing coprocessor instruction requires another execute cycle, it examines CHSEX[1:0].

8.3.1 Coprocessor Handshake States

The handshake signals encode one of four states: ABSENT, WAIT, GO, and LAST. Table 8.1 describes these four handshake states.

Table 8.1 Coprocessor Handshake States

State	Description
ABSENT	If there is no coprocessor attached that can execute the coprocessor instruction, the handshake signals indicate the ABSENT state. In this case, the ARM9E-S takes the undefined instruction trap.
WAIT	If there is a coprocessor attached that can handle the instruction, but not immediately, the coprocessor handshake signals are driven to indicate that the ARM9E-S processor core must stall until the coprocessor can catch up. This is known as the *busy-wait* condition. In this case, the ARM9E-S processor core loops in an IDLE state waiting for CHSEX[1:0] to be driven to another state, or for an interrupt to occur. If CHSEX[1:0] changes to ABSENT, the undefined instruction trap is taken. If CHSEX[1:0] changes to GO or LAST, the instruction proceeds as described below. If an interrupt occurs, the ARM9E-S processor is forced out of the busy-wait state. This is indicated to the coprocessor by the CPPASS signal going LOW. The instruction is restarted later and so the coprocessor must not commit to the instruction (it must not change any coprocessor state) until it has seen CPPASS HIGH at the same time as the handshake signals indicate the GO or LAST condition.
GO	The GO state indicates that the coprocessor can execute the instruction immediately, and that it requires another cycle of execution. Both the ARM9E-S processor core and the coprocessor must also consider the state of the CPPASS signal before actually committing to the instruction. For an LDC or STC instruction, the coprocessor instruction drives the handshake signals with GO when two or more words still have to be transferred. When only one more word remains to be transferred, the coprocessor drives the handshake signals with LAST. During the Execute stage, the ARM9E-S processor core outputs the address for the LDC/STC. Also in this cycle, DnMREQ is driven LOW, indicating to the ARM946E-S memory system that a memory access is required at the data end of the device. The timing for the data on CPDOUT and CPDIN is shown in Figure 8.2 on page 8-3.
LAST	You can use an LDC or STC for more than one item of data. If this is the case, possibly after busy-waiting, the coprocessor drives the coprocessor handshake signals with a number of GO states, and in the penultimate cycle it drives LAST. The LAST state indicates that the next transfer is the final one. If there is only one transfer, the sequence is WAIT, WAIT, ..., LAST.

8.3.2 Coprocessor Handshake Encoding

Table 8.2 shows the encoding for the CHSDE[1:0] and CHSEX[1:0] handshake signals.

Table 8.2 Handshake Encoding

CHSDE/CHSEX[1:0]	Definition
10	ABSENT
00	WAIT
01	GO
11	LAST

Note: If an external coprocessor is not attached in the ARM946E-S embedded system, the CHSDE[1:0] and CHSEX[1:0] handshake inputs must be tied off to indicate ABSENT.

8.3.3 Multiple External Coprocessors

If multiple external coprocessors are attached to the ARM946E-S interface, you can combine the handshaking signals by ANDing bit 1 and ORing bit 0. In the case of two coprocessors that have handshaking signals (where coprocessor 1 signals are CHSDE1/CHSEX1 and coprocessor 2 signals are CHSDE2/CHSEX2), combine the signals as shown below:

CHSDE[1] = CHSDE1[1] AND CHSDE2[1**]**

CHSDE[0] = CHSDE1[0] OR CHSDE2[0**]**

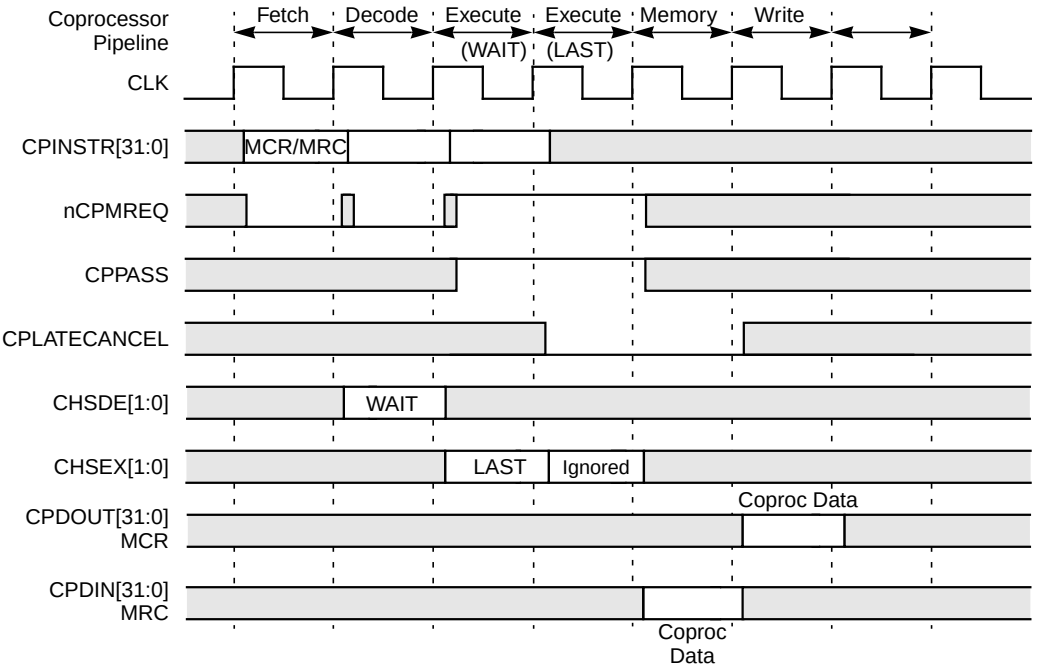
CHSEX[1] = CHSEX1[1] AND CHSEX2[1]

CHSEX[0] = CHSEX1[0] OR CHSEX2[0].

8.4 MCR/MRC Instructions

The MCR (Move CPU Register to Coprocessor Register) instruction and the MRC (Move from Coprocessor Register to CPU Register) instruction have timing cycles that are very similar to the STC/LDC instructions. Figure 8.3 provides an example with a busy-wait state.

Figure 8.3 MCR/MRC Transfer Timing with Busy-Wait



First, nCPMREQ is driven LOW to indicate that the instruction on CPINSTR[31:0] is entering the Decode stage of the pipeline. This causes the coprocessor to decode the new instruction and drive CHSDE[1:0] as required. In the next cycle, nCPMREQ is driven LOW to indicate that the instruction has now been issued to the execute stage. If the condition codes pass, and therefore, the instruction is to be executed, then the CPPASS signal is driven HIGH and the CHSDE[1:0] handshake bus is examined. It is ignored in all other cases.

For any successive execute cycles, the CHSEX[1:0] handshake bus is examined. When the LAST condition is observed, the instruction is

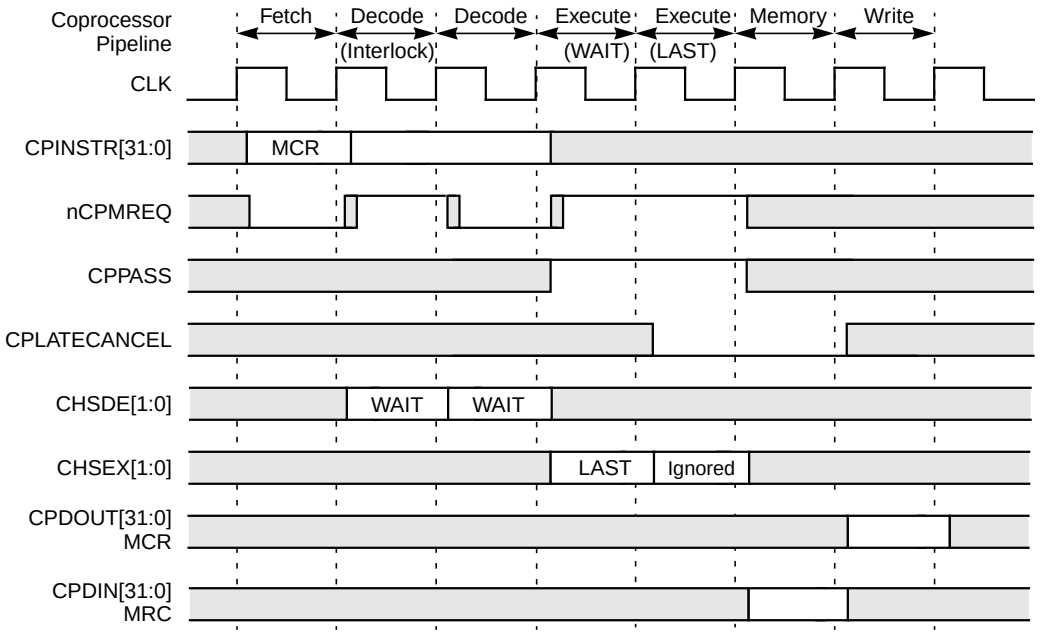
committed. In the case of an MCR instruction, the CPDOUT[31:0] bus is driven with the registered data during the coprocessor write stage. In the case of an MRC instruction, CPDIN[31:0] is sampled at the end of the ARM9E-S processor Memory stage and written to the destination register during the next cycle.

8.5 Interlocked MCR Instructions

If the data for an MCR instruction is not available inside the ARM9E-S processor pipeline during its first decode cycle, then the ARM9E-S processor pipeline interlocks for one or more cycles until the data is available. For example, this interlocking applies when the register being transferred is the destination from a preceding LDR instruction. In this situation, the MCR instruction enters the decode stage of the coprocessor pipeline, and then remains there a number of cycles before entering the execute stage.

Figure 8.4 gives an example of an interlocked MCR that also has a busy-wait state.

Figure 8.4 Interlocked MCR Timing with Busy-Wait

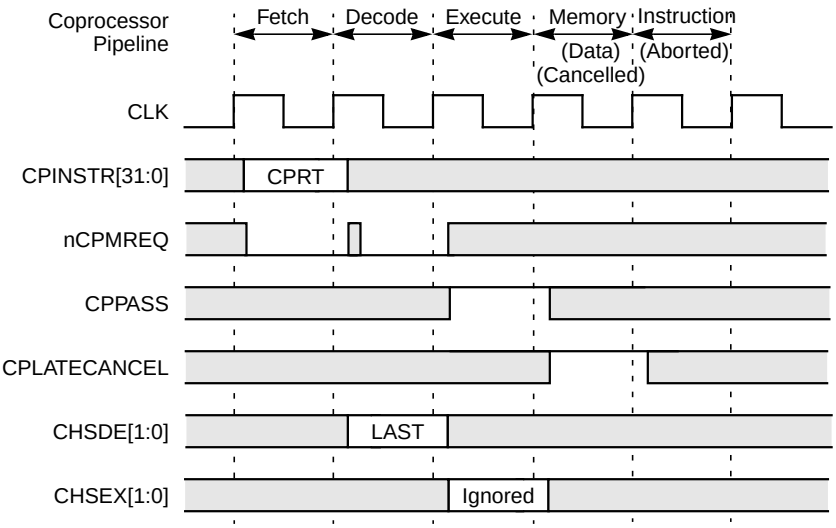


8.6 CDP Instructions

Coprocessor Data Processing (CDP) instructions normally execute in a single cycle. As with all the previous cycles, nCPMREQ is driven LOW to indicate when an instruction is entering the decode stage and then the execute stage of the pipeline. If the instruction is to be executed, the CPPASS signal is driven HIGH during the execute cycle. If the coprocessor can execute the instruction immediately, it drives CHSDE[1:0] with LAST. If the instruction requires a busy-wait cycle, the coprocessor drives CHSDE[1:0] with WAIT and then CHSEX[1:0] with LAST.

Figure 8.5 shows a CDP instruction that is cancelled because the previous instruction caused a Data Abort.

Figure 8.5 Late Cancelled CDP Instruction

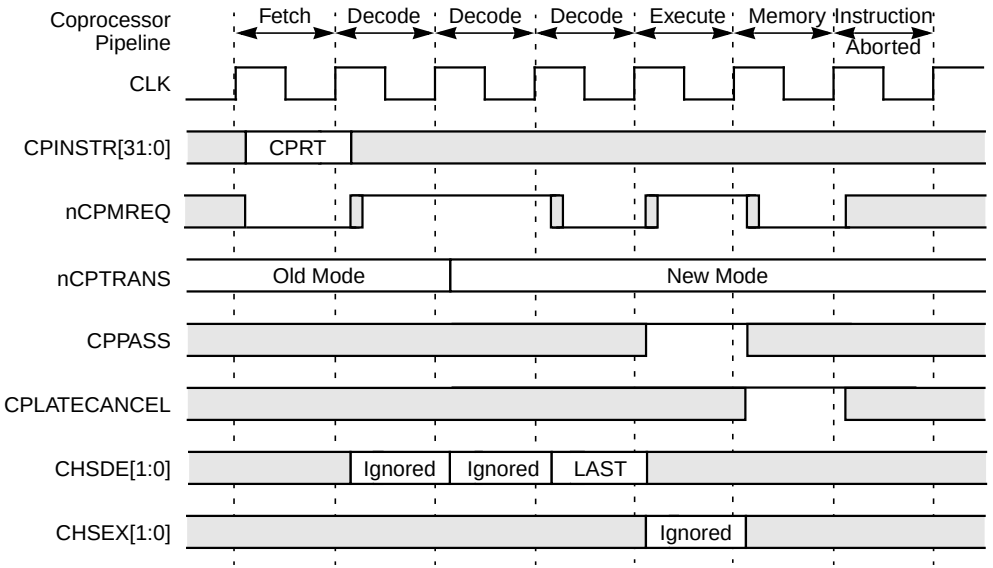


In this example, the CDP instruction enters the execute stage of the pipeline and is signaled to execute by CPASS. In the following cycle, CPLATECANCEL is asserted. This event forces the coprocessor to terminate execution of the CDP instruction and prevents the instruction from causing any coprocessor state changes.

8.7 Privileged Instructions

The coprocessor can restrict some instructions for use in Privileged mode only. To do this, the coprocessor tracks the nCPTRANS output. When nCPTRANS is LOW, the processor is in User mode. When it is HIGH, the processor is in Privileged mode. Figure 8.6 shows how nCPTRANS changes after a mode change.

Figure 8.6 Privileged Instructions



The first two CHSDE[1:0] responses are ignored by the ARM9E-S processor, because only the final CHSDE[1:0] response counts (when the instruction moves from decode into execute). This method allows the coprocessor to change its response as nCPTRANS changes.

8.8 Busy-Waiting and Interrupts

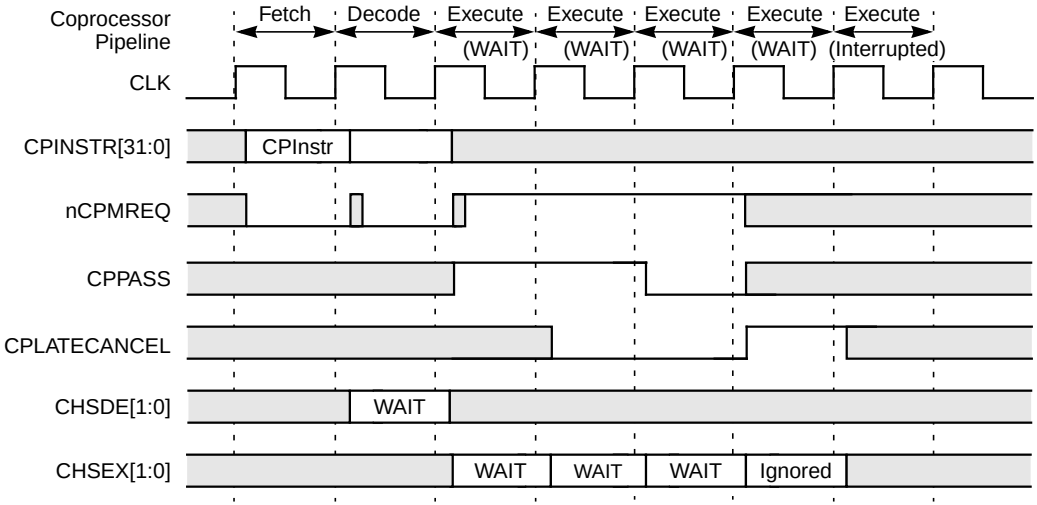
The coprocessor is permitted to stall, or busy-wait, the processor during the execution of a coprocessor instruction. For example, the coprocessor can do this if it is busy with an earlier coprocessor instruction. To do so, the coprocessor associated with the decode stage instruction drives WAIT onto CHSDE[1:0]. When the instruction concerned enters the

execute stage of the pipeline, the coprocessor can drive WAIT onto CHSEX[1:0] for as many cycles as necessary to keep the instruction in the busy-wait loop.

For interrupt latency reasons, the coprocessor can be interrupted while busy-waiting. This causes the instruction to be abandoned. Abandoning execution is done through CPPASS. The coprocessor must monitor the state of CPPASS during every busy-wait cycle. If it is HIGH, the instruction must still be executed. If it is LOW, the instruction must be abandoned.

Figure 8.7 shows a busy-waiting coprocessor instruction abandoned due to an interrupt. CPLATECANCEL is also asserted as a result of the execute interruption.

Figure 8.7 Busy-Waiting and Interrupts



Chapter 9
Debug Interface

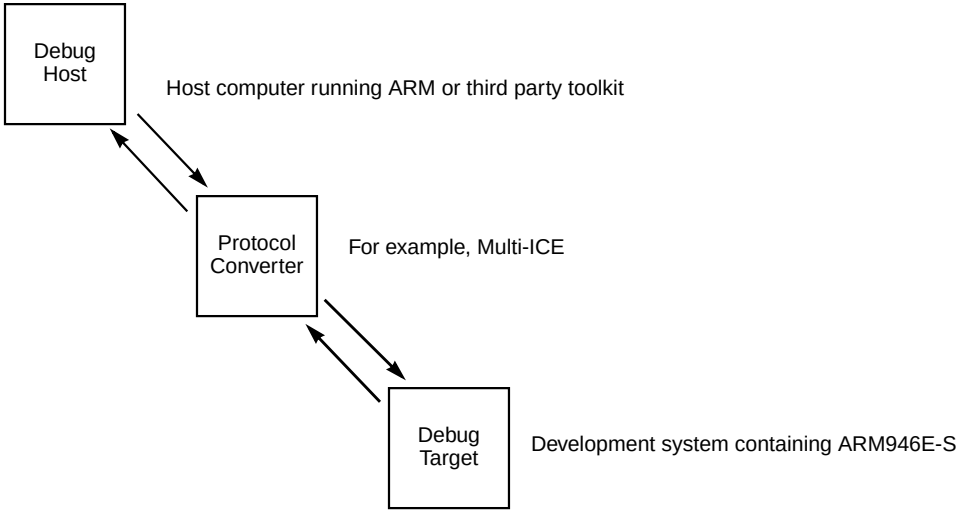
This chapter describes the ARM946E-S debug interface. It contains the following sections:

A more detailed description of the ARM9E-S debug features and JTAG interface is provided in the *ARM9E-S Technical Reference Manual,* Appendix D, “Debug in Depth.”

9.1 Debug Systems

The ARM946E-S forms one component of a debug system, a system that ranges from a high-level, user debugging capability to the low-level hardware interfaces in the ARM946E-S. Figure 9.1 shows a typical debug system.

Figure 9.1 Typical Debug System



A debug system typically has three parts:

- Debug Host
- Protocol Converter
- ARM946E-S Debug Target

The debug host and the protocol converter are system-dependent. The following subsections describe each of the debug system blocks.

9.1.1 Debug Host

The debug host is a computer that is running a software debugger, such as *armsd*. The debug host allows you to issue high-level commands, such as setting breakpoints or examining memory contents.

9.1.2 Protocol Converter

An interface, such as a parallel port, connects the debug host to the ARM946E-S development system. The messages broadcast over this connection must be converted to the interface signals of the ARM946E-S. The protocol converter does the conversion.

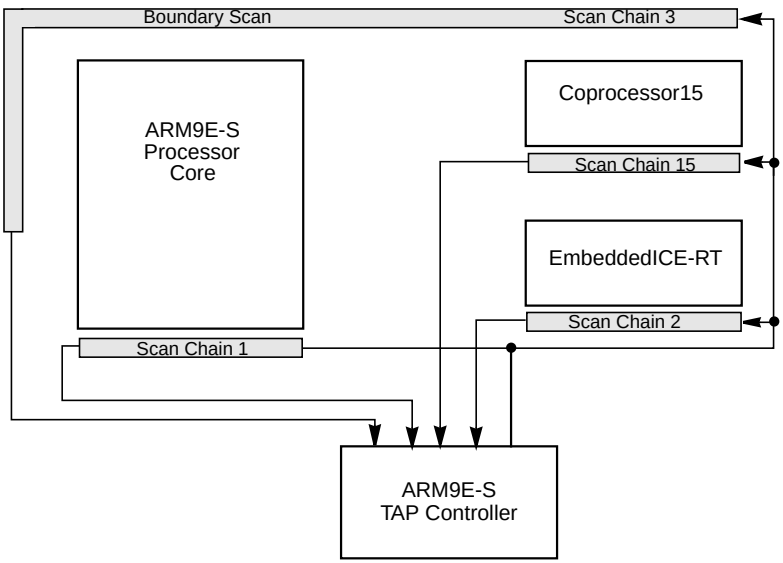
9.1.3 ARM946E-S Debug Target

The ARM9E-S processor core within the ARM946E-S has hardware extensions that ease debugging at the lowest level. The debug extensions make it possible to:

- Stall the processor program execution
- Examine the core internal state
- Examine the state of the memory system
- Resume program execution

Figure 9.2 shows the major blocks of the ARM9E-S processor and associated debug logic.

Figure 9.2 ARM9E-S Processor and Debug Logic



The blocks in Figure 9.2 are briefly described as follows:

ARM9E-S Processor Core Includes debug support hardware

EmbeddedICE-RT Logic Contains a set of registers and comparators used to generate debug exceptions (such as breakpoints)

TAP Controller	Controls the scan chains using the JTAG serial interface
CP15	Contains system configuration and control state
Boundary Scan	Includes the optional external scan chain
Scan Chain 1	Provides access to the processor instruction and data buses
Scan Chain 2	Provides access to the registers in the EmbeddedICE-RT
Scan Chain 3	Provides control of the optional external boundary scan chain
Scan Chain 15	Provides access to the CP15 register set and the cache

9.2 Debug Operations Overview

The embedded ARM9E-S processor core provides debug support capabilities for the ARM946E-S. The ARM946E-S debug interface is based on IEEE Std. 1149.1- 1990, *Standard Test Access Port and Boundary-Scan Architecture*. See this standard for an explanation of the terms used in this chapter.

The ARM9E-S processor core, which contains hardware extensions for advanced debugging features, makes it easier to develop the hardware, the application software, and operating systems.

These debug extensions allow you to force the processor to stop for a particular:

- instruction fetch - using breakpoints
- data access - using watchpoints
- external debug request

The condition when stopped is known as the *debug state*. In the debug state, the processor core and ARM946E-S memory system are effectively stopped and isolated from the rest of the system. This condition is known as *halt mode* operation and allows you to examine the internal state of the ARM9E-S core, ARM946E-S system, and external AHB state, while all other system activity continues normally. When

debug has been completed, the ARM9E-S restores the processor and system state, and resumes program execution.

The examination of the internal state of the ARM946E-S uses a JTAG-style interface that allows the serial insertion of instructions into the instruction pipeline, and exports the contents of the ARM9E-S core registers. The exported data is serially shifted out without affecting the rest of the system.

In addition, the ARM9E-S supports a real-time debug mode, where instead of generating a breakpoint or watchpoint, an internal Instruction Abort or Data Abort is generated. This mode is known as *monitor mode* operation.

When used in conjunction with a debug monitor program activated by the abort exception entry, you can debug the ARM946E-S while allowing the execution of critical interrupt service routines. The debug monitor program typically communicates with the debug host over the ARM946E-S debug communication channel. Real-time debug is described in Section 9.7, “Real-Time Debug,” page 9-28.

9.3 Debug Using the Serial Interface and TAP Controller

The JTAG Interface includes six serial registers and a TAP controller state machine.

9.3.1 Serial Registers

The JTAG interface includes the following serial registers:

- Boundary Scan Register

  Contains boundary scan data.

- Bypass Register

  A one-bit shift register that contains test data.

- Device ID Code Register

  The content of this register identifies the device. TAPID[31:0] drive this register. Tie these signals to a constant value that represents the unique device ID code.

- TAP Instruction Register

  The content of this four-bit register selects the register (boundary scan, bypass, device ID, scan path select, scan) to be read and written when the TAP controller is in the Shift-DR state.

  The TAP instruction register does not include a parity bit. During the CAPTURE-IR state, a fixed value of 0b0001 is loaded into this register.

- Scan Path Select Register

  This register selects the scan path.

- Scan Register

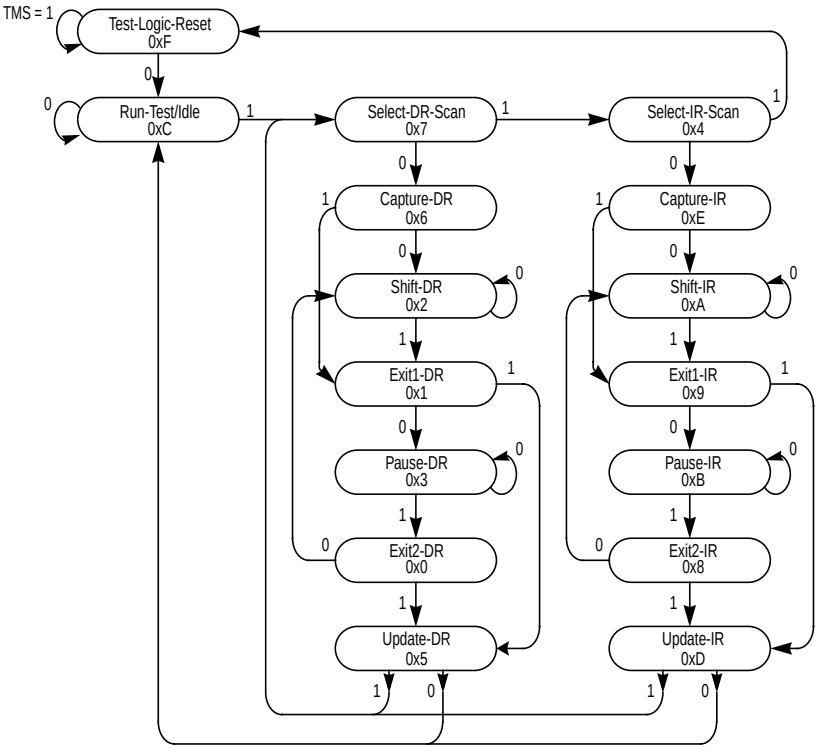
  Contains data shifted in from the associated scan path.

9.3.2 TAP Controller State Machine

Figure 9.3 shows the state transitions that occur in the TAP controller.

Each state is encoded as a hexadecimal value, which is output on TAPSM[3:0]. These values are shown in the diagram.

Figure 9.3 TAP Controller State Diagram



Note:

1. The state transition values (0 and 1) in this diagram correspond to the Test Mode Select input signal, DBGTMS.
2. The hexadecimal value in each bubble represents the DBGTAPSM[3:0] output.

The TAP controller states are briefly defined as follows:

- Test-Logic-Reset

  Resets the debug interface. See Section 9.3.2.1, “Resetting the TAP Controller” for more information.

- Run-Test/Idle

  The TAP controller is idle and can be left in this state when not used.

- Select-IR-Scan and Select-DR-Scan

  These states serve as intermediate states on the path to selecting either instruction or data register update.

- Capture-DR

  The action taken during this state depends on the TAP instruction being executed. See Table 9.1 for TAP instruction descriptions.

- Capture-IR

  During this state, 0x1 is loaded into the TAP instruction register.

- Shift-DR

  The data register is inserted in the TDI/TDO shift path and shifted on each rising edge of TCK.

- Shift-IR

  The instruction register is inserted in the TDI/TDO shift path and shifted on each rising edge of TCK.

- Exit1-DR and Exit1-IR

  These are temporary states with no effect.

- Pause-DR and Pause-IR

  This state is temporary with no effect, except pausing.

- Exit2-DR and Exit2-IR

  These states are temporary with no effect.

- Update-DR

  The action taken during this state depends on the TAP instruction being executed. See Table 9.1 for TAP instruction descriptions.

- Update-IR

  During this state, the value in the TAP instruction register is the current instruction.

9.3.2.1 Resetting the TAP Controller

To force the TAP controller into the correct state after power-up of the device, apply a reset pulse to the DBGnTRST signal or cycle the JTAG state machine through the TEST-LOGIC-RESET state. Before using the JTAG interface, drive DBGnTRST LOW, and then HIGH again. If you do not intend to use the boundary scan interface, tie the DBGnTRST input permanently LOW.

Note: A clock on TCK is not necessary to reset the device.

Reset causes the following action:

1. Reset forces exit from the debug state. The boundary scan chain cells do not intercept any of the signals passing between the external system and the core.
2. The IDCODE instruction is selected. If the TAP controller is put into the SHIFT-DR state and TCK is pulsed, the contents of the ID register are clocked out of TDO.

9.3.2.2 JTAG Interface Signals and Pull-Up Resistors

The IEEE 1149.1 standard effectively requires TDI and TMS to have internal pull-up resistors. To minimize static current draw, these resistors are not included in the ARM9E-S processor core. Accordingly, the four inputs to the test interface (the TDO, TDI, TMS, and TCK) must all be driven to valid logic levels to achieve normal circuit operation.

9.3.2.3 Test Access Port Instructions

This section describes how the TAP controller state machine controls the serial JTAG interface when the following instructions are executed:

- EXTEST - External test
- SCAN_N - Scan-in
- INTEST - Internal test
- IDCODE - Device ID code
- BYPASS - Bypass
- SAMPLE/PRELOAD - Sample/preload
- RESTART - Restart

In this section, it is assumed that 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.

Table 9.1 provides a description of each TAP instruction.

Table 9.1 Test Access Port Instruction Descriptions

Instruction	Description
EXTEST (0000)	The EXTEST instruction puts the selected scan chain and all scan cells in test mode, and it connects the selected scan chain between TDI and TDO. In the CAPTURE-DR state, inputs from the system logic and outputs from the output scan cells to the system are captured by the scan cells. In the SHIFT-DR state, previously captured test data is shifted out of the scan chain on TDO, while new test data is shifted in through TDI. This data is applied immediately to the system logic and system pins.
SCAN_N (0010)	This instruction connects the Scan Path Select register between TDI and TDO. During the CAPTURE-DR state, a fixed value of 0b10000 is loaded into the Scan Path Select register. During the SHIFT-DR state, the ID number of the desired scan path is shifted into the Scan Path Select register. In the UPDATE-DR state, the Scan register of the selected scan chain is connected between TDI and TDO, and it remains connected until a subsequent SCAN_N instruction is issued. On reset, scan chain 3 is selected by default. The Scan Path Select register is 5 bits long in this implementation, although no finite length is specified.
INTEST (1100)	The INTEST instruction puts the selected scan chain and all scan cells in test mode, and it connects the selected scan chain between TDI and TDO. In the CAPTURE-DR state, the output and input scan cells capture the value of the data applied from the core logic and the system logic, respectively. In the SHIFT-DR state, previously captured test data is shifted out of the scan chain on TDO, while new test data is shifted in through TDI.

Table 9.1 Test Access Port Instruction Descriptions (Cont.)

Instruction	Description
IDCODE (1110)	The IDCODE instruction connects the Device Identification (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. The ID register is loaded from the TAPID[31:0] input bus. This input must be tied to a constant value that provides a unique device ID code. When the instruction register is loaded with the IDCODE instruction, all the scan cells are placed in their normal (system) mode of operation. In the CAPTURE-DR state, the ID register captures the device identification code. In the SHIFT-DR state, the previously captured device identification code is shifted out of the ID register on TDO, while data is shifted into the ID register through the TDI pin. In the UPDATE-DR state, the ID register is not changed.
BYPASS (1111)	The BYPASS instruction connects the 1-bit shift Bypass register between TDI and TDO. When the BYPASS instruction is loaded into the instruction register, all the 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, the Bypass register captures a logic 0. In the SHIFT-DR state, test data shifts into the Bypass register on TDI, and then after a delay of one TCK cycle, it shifts out on TDO. The first bit shifted out is a 0. The Bypass register is not affected by the UPDATE-DR state. Note: All unused instruction codes default to the BYPASS instruction.

Table 9.1 Test Access Port Instruction Descriptions (Cont.)

Instruction	Description
SAMPLE/PRELOAD (0011)	When the TAP instruction register is loaded with the SAMPLE/PRELOAD instruction, all the scan cells of the selected scan chain are placed in the normal mode of operation. In the CAPTURE-DR state, a snapshot of the signals of the boundary scan is taken on the rising edge of TCK. Normal system operation is not affected. In the SHIFT-DR state, sampled test data shifts out of the boundary scan on TDO, while new data shifts in on TDI to preload the boundary scan parallel input latch. This data is not applied to the system logic or system pins while the SAMPLE/PRELOAD instruction is active. You must use this instruction to preload the Boundary Scan register with known data prior to executing INTEST or EXTEST instructions.
RESTART (0100)	This instruction restarts the processor on exit from the debug state. The RESTART instruction connects the Bypass register between TDI and TDO, and the TAP controller behaves as if the BYPASS instruction is loaded. The processor re-synchronizes back to the memory system when the RUN-TEST/IDLE state is entered.

9.3.3 Scan Chains

ARM946E-S supports 32 scan chains. Three of the scan chains are located inside the ARM946E-S. The scan chains allow testing, debugging, and programming of the EmbeddedICE macrocell watchpoint units.

Table 9.2 lists the scan chains and their functions.

Table 9.2 ARM946E-S Scan Chain Functions

Scan Chain	Function
0	Reserved
1	Debug
2	EmbeddedICE-RT logic programming
3	External boundary scan

Table 9.2 ARM946E-S Scan Chain Functions (Cont.)

Scan Chain	Function
4–14	Reserved
15	Control coprocessor
16–31	Unassigned

9.3.3.1 Scan Chain 1

Scan chain 1 is 67 bits long. Its primary function is to provide debugging support and access to the core instruction and data buses.

Table 9.3 shows the bit allocation for scan chain 1.

Table 9.3 Scan Chain 1 Bit Allocation

Bit	Function
67:35	Data values
34	SYSPEED Control
33	WPTANDBKPT Control
32	Reserved
31:0	Instruction values

During debugging operations, the value of the SYSSPEED control bit determines whether the ARM9E-S core executes an instruction at system speed or not.

After the ARM946E-S enters the debug state, the first time SYSSPEED is captured and scanned out tells the debugger whether the core has entered debug state due to a breakpoint (SYSSPEED LOW) or a watchpoint (SYSSPEED HIGH). A watchpoint and a breakpoint can occur simultaneously. When a watchpoint condition occurs, the debugger must examine the WPTANDBKPT bit to determine whether the instruction currently in the Execute stage of the pipeline is breakpointed. If it is, WPTANDBKPT is HIGH, otherwise it is LOW.

9.3.3.2 Scan Chain 2

Scan chain 2 is 37 bits long. This scan chain allows access to the EmbeddedICE-RT logic registers. The order of the scan chain, from DBGTDI to DBGTDO, is [37:0].

Table 9.4 shows the bit allocation for scan chain 2.

Table 9.4 Scan Chain 2 Bit Allocation

Bit	Function
37	Read/write control. Read = 0. Write = 1.
36:32	Register address
31:0	Data value

During CAPTURE-DR, no action occurs for scan chain 2.

During SHIFT-DR, a data value is shifted into the serial register. Bits 36:32 specify the address of the EmbeddedICE-RT register to be accessed.

During UPDATE-DR, either read or write the EmbeddedICE-RT register depending on the value of bit 37.

9.3.3.3 Scan Chain 3

Scan chain 3 allows ARM946E-S to control an optional external boundary scan chain. You determine the length of scan chain 3.

9.3.3.4 Scan Chain 15

Scan chain 15 provides debug access to CP15 registers. This access allows you to control the system state within the ARM946E-S while in the debug mode. For example, you can use it to enable or disable the SRAM before performing a debug load or store.

You can also use scan chain 15 to interrogate the cache.

Scan chain 15 is 39 bits long. Table 9.5 shows the order of the scan chain bits from the DBGTDI input to the DBGTDO output.

Table 9.5 Scan Chain 15 Bit Allocation

Bits	Contents
38	Read/write control. Read = 0. Write = 1.
37:32	CP15 register address
31:0	CP15 data value

Table 9.6 shows the address mapping of scan chain 15 to CP15 registers.

Table 9.6 Mapping of Scan Chain 15 Address Field to CP15 Registers

Register Address			Register Number	CP15 Register	
37	[36:33]	32		Name	Type
0	0000	0	C0.ID	ID register	Read
0	0000	1	C0.C	Cache type	Read
0	0001	0	C1	Control	Read/write
0	0010	0	C2.D	Data cacheable bits	Read/write
0	0010	1	C2.I	Instruction cacheable bits	Read/write
0	0011	0	C3	Write buffer control	Read/write
0	0100	0	C0.M	Tightly coupled memory size	Read
0	0101	0	C5.D	Data space access permissions	Read/write
0	0101	1	C5.I	Instruction address access permissions	Read/write
1	<Crm>[1]	0	C6.[7:0]	Memory region protection	Read/write
0	0111	0	C7.FD	Flush data cache	Write
0	0111	1	C7.FI	Flush instruction cache	Write
0	1110	0	C7.FD.s	Flush D-cache single (uses C15.C.Ind)	Write
0	1110	1	C7.FI.s	Flush I-cache single (uses C15.C.Ind)	Write

Table 9.6 Mapping of Scan Chain 15 Address Field to CP15 Registers (Cont.)

Register Address			Register Number	CP15 Register	
37	[36:33]	32		Name	Type
1	1010	1	C7.CD.s	Clean D-cache single (uses C15.C.Ind)	Write
0	1001	0	C9.D	Data cache lockdown	Read/write
0	1001	1	C9.I	Instruction cache lockdown	Read/write
1	1000	1	C9.Dram	Data SRAM size/location	Read/write
1	1001	1	C9.Iram	Instruction SRAM size/location	Read/write
0	1101	1	C13.TPID	Trace process identifier	Read/write
0	1111	0	C15.State	Test state	Read/write
0	1111	1	C15.Tag	Tag BIST control	Read/write
1	1111	1	C15.RAM	Cache RAM BIST control	Read/write
1	1101	0	C15.C.Ind	Cache index (address/segment)	Read/write
0	1010	0	C15.DC	Data cache read/write (uses C15.C.Ind)	Read/write
0	1010	1	C15.IC	Instruction cache read/write (uses C15.C.Ind)	Read/write
0	1011	0	C15.DT	Data tag read/write (uses C15.C.Ind)	Read/write
0	1011	1	C15.IT	Instruction tag read/write (uses C15.C.Ind)	Read/write
1	1110	1	C15.Mem	Memory RAM BIST control	Read/write

1. For CP15 register 6, CRm corresponds to memory region number [7:0].

When the TAP controller is in the SHIFT-DR state, the scan chain 15 read/write bit, register address, and register value (for writing) shift in from TDI.

For a write operation, the register value is updated when the TAP controller reaches the UPDATE-DR state.

For a read operation, return to SHIFT-DR through CAPTURE-DR to shift out the register value.

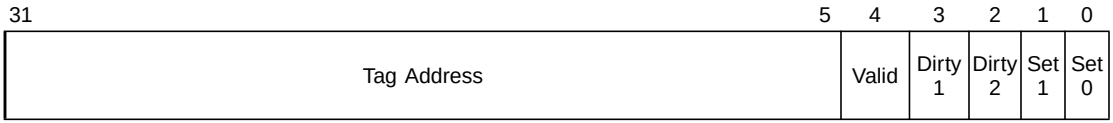
9.3.4 Debug Access to the Caches

It is useful for the debugger to examine the instruction and data cache contents during debug operations. This examination requires two steps:

1. The debugger determines if valid addresses are stored in the cache and forms tag addresses from the tag contents and the tag index.
2. The debugger uses the generated addresses either to access main memory or to read individual entries using the CP15 scan chain.

Step 1 – To do this step, the debugger reads the I-cache and D-cache tag arrays using scan chain 15. The debugger must do this for each entry set within the cache. Figure 9.4 shows the format of the return data.

Figure 9.4 Tag Address Format



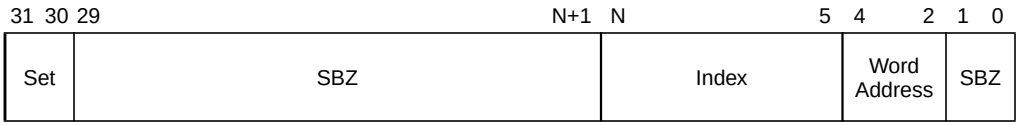
The tag address is formed from the tag contents and the tag index. This combination ensures that the format of the return data is consistent regardless of cache size.

Step 2 – Reading individual entries using the CP15 scan chain is useful if an entry is marked dirty, because this indicates an inconsistency between the cache contents and main memory.

For the D-cache, the debugger can execute system speed accesses that hit in the cache and return the cache contents. Writes to the D-cache from the processor core using this method cause the dirty bits to set for write-back regions, and main memory is updated for write-through regions.

If the CP15 scan chain is used for updating the D-cache, only the cache contents are updated. Writes are not made to main memory. With this method, you must first program the index/set register with the required cache index, set, and word values. Figure 9.5 shows the Cache Index register format.

Figure 9.5 Cache Index Register Format



Note: Although 27 bits are specified for the tag address, only those bits required for the particular tag implementation are used.

The cache index register is also used for writing to the instruction cache. This is useful for setting software breakpoints within code already in the cache. It means that you do not have to flush the cache and reload the entry.

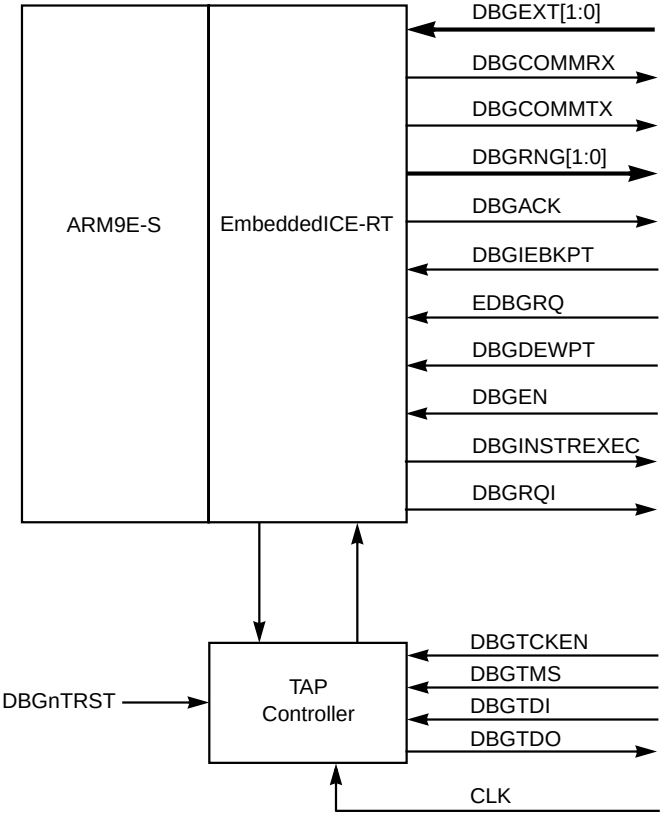
Note: There is no mechanism for detecting that the I-cache has been updated in this way. The debugger must restore the original cache contents after executing the breakpoint.

9.4 Debug Using the EmbeddedICE-RT

The ARM9E-S EmbeddedICE-RT logic provides integrated on-chip debug support for the ARM9E-S core within the ARM946E-S.

EmbeddedICE-RT is programmed serially using the ARM9E-S TAP controller. Figure 9.6 illustrates the relationship between the core, EmbeddedICE-RT, and the TAP controller, showing only the signals that are pertinent to the EmbeddedICE-RT.

Figure 9.6 The ARM9E-S, Tap Controller, and EmbeddedICE-RT



The EmbeddedICE-RT logic consists of:

- Two real-time watchpoint units
- Two independent registers:
  - Debug Control register
  - Debug Status register
- Debug comms channel

The Debug Control register and the Debug Status register provide overall control of EmbeddedICE-RT operation. For more detailed information about these registers, refer to the *ARM9E-S Technical Reference Manual,* Appendix D, “Debug in Depth.”

You can program one or both watchpoint units to halt the execution of instructions by the core. Execution halts when the values programmed into EmbeddedICE-RT match the values currently appearing on the address bus, data bus, and various control signals.

Note: You can mask bits so that their values do not affect the comparison.

You can configure each watchpoint unit to be either a watchpoint (monitoring data accesses) or a breakpoint (monitoring instruction fetches). Watchpoints and breakpoints can be data-dependent in halt mode debug.

9.4.1 Disabling EmbeddedICE-RT

To disable EmbeddedICE-RT, set the DBGEN input LOW.

Note: Hardwiring the DBGEN input LOW *permanently* disables debug access.

When DBGEN is LOW, it inhibits DBGDEWPT, DBGIEBKPT, and EDBGRQ to the core, and DBGACK from the ARM946E-S is always LOW.

9.4.2 Debug Communications Channel

The ARM9E-S EmbeddedICE-RT logic contains a communications channel for passing information between the target and the host debugger. This channel is implemented as coprocessor 14.

The communications channel consists of:

- A 32-bit comms data read register
- A 32-bit comms data write register
- A 6-bit comms control register for synchronized handshaking between the processor and the asynchronous debugger

These registers are located in fixed locations in the EmbeddedICE-RT logic register map and are accessed from the processor using MCR and MRC instructions to coprocessor 14.

In addition to the comms channel registers, the processor can access a 1-bit debug status register for use in the real-time debug configuration.

9.4.3 Debug Comms Channel Registers

There are four Debug Comms Channel registers. Table 9.7 lists the registers.

Table 9.7 Coprocessor 14 Register Map

Register Name	Register Number	Notes
Comms Channel Status	C0	Read-only
Comms Channel Data Read	C1	For reads
Comms Channel Data Write	C1	For writes
Debug Status	C2	Read/write

For a description of each register and its format, including field and bit definitions, refer to Section 3.4, “CP14 Registers” in this manual.

9.4.4 Communications Using the Comms Channel

You can send and receive messages using the comms channel.

9.4.4.1 Sending a Message to the Debugger

When the processor has to send a message to the debugger, it must check that the Comms Data Write register is free for use by finding out whether the W bit of the Debug Comms Control register is clear.

The processor reads the Debug Comms Control register to check the status of the W bit:

- If the W bit is clear, the Comms Data Write register is clear.
- If the W bit is set, previously written data has not been read by the debugger. The processor must continue to poll the control register until the W bit is clear.

When the W bit is clear, a message is written by a register transfer to coprocessor 14. As the data transfer occurs from the processor to the Comms Data Write register, the W bit is set in the Debug Comms Control register.

The debugger sees both the R and W bits when it polls the Debug Comms Control register through the JTAG interface. When the debugger sees that the W bit is set, it can read the Comms Data Write register, and scan the data out. The action of reading this data register clears the Debug Comms Control register W bit. At this point, the communications process can begin again.

9.4.4.2 Receiving a Message from the Debugger

Transferring a message from the debugger to the processor is similar to sending a message to the debugger. In this case, the debugger polls the R bit of the Debug Comms Control register.

- If the R bit is LOW, the Comms Data Read register is free, and data can be placed there for the processor to read.
- If the R bit is set, previously deposited data has not yet been collected, so the debugger must wait.

When the Comms Data Read register is free, data is written there using the JTAG interface. The action of this write sets the R bit in the Debug Comms Control register.

The processor polls the Debug Comms Control register. If the R bit is set, there is data that can be read using an MRC instruction to coprocessor 14. The action of this load clears the R bit in the Debug Comms Control register. When the debugger polls this register and sees that the R bit is clear, the data has been taken, and the process can now be repeated.

9.5 Breakpoints, Watchpoints, and Debug Requests

Breakpoints, watchpoints, and external debug requests can cause the ARM946E-S to enter the debug state. These events are associated with the following debug interface signals:

- DBGIEBKPT, DBGDEWPT, and EDBGRQ are system requests for the ARM946E-S to enter the debug state. The three signals indicate an instruction breakpoint, data watchpoint, and external debug request, respectively. All three originate from hardware external to the ARM946E-S.

- DBGACK acknowledges to the system that the ARM946E-S processor is in the debug state.

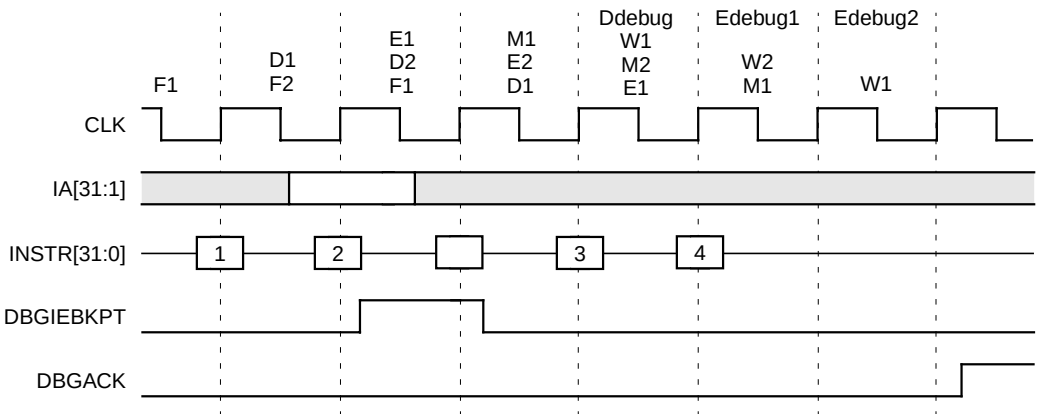
The notations in Figure 9.7 through Figure 9.9 are defined as follows:

1. Fn, Dn, En, Mn, Wn = Fetch, Decode, Execute, Memory, and Writeback, respectively, for instruction n, where n is the instruction in INSTR[31:0]
2. LDR = Load register from memory instruction
3. Dp = Any data processing instruction
4. B = Branch instruction
5. Ddebug = Decode debug entry
6. Edebug = Execute debug
7. T = Branch target

9.5.1 Entry into Debug State on Breakpoint

Any instruction fetched from memory is sampled at the end of a cycle. To apply a breakpoint to that instruction, you must assert the breakpoint signal by the end of the same cycle. Figure 9.7 illustrates breakpoint timing.

Figure 9.7 Breakpoint Timing



To extend the breakpoint functionality of the EmbeddedICE-RT logic, you can add external logic, such as additional breakpoint comparators. The external logic output must be applied to the DBGIEBKPT input. This

signal is ORed with the internally generated Breakpoint signal before being applied to the ARM9E-S core control logic. The timing of the input makes it unlikely that data-dependent external breakpoints can occur.

A breakpointed instruction can enter the execute stage of the pipeline, but state changes that normally occur from executing the instruction are inhibited. All writes from previous instructions complete as usual.

The decode cycle of the debug entry sequence occurs during the execute cycle of the breakpointed instruction. The latched Breakpoint signal forces the processor to start the debug sequence.

9.5.2 Breakpoints and Exceptions

A breakpointed instruction can have a Prefetch Abort associated with it. If so, the Prefetch Abort takes priority and the breakpoint is ignored. It is ignored, because if there is a Prefetch Abort, the instruction data might be invalid and the breakpoint could be data-dependent. Since the data could be incorrect, the breakpoint might have triggered incorrectly.

SWI and undefined instructions are treated the same as any other instruction that might incur a breakpoint. Therefore, the breakpoint takes priority over the SWI or undefined instruction.

On an instruction boundary, if there is a breakpointed instruction and an interrupt (nIRQ or nFIQ), the interrupt is taken and the breakpointed instruction is discarded. After the interrupt is serviced, the execution flow is returned to the original program. The previously breakpointed instruction is fetched again. If the breakpoint is still set, the processor enters the debug state when it reaches the pipeline execute stage.

After the processor enters the halt mode debug state, it is important that additional interrupts not affect the instructions executed. For this reason, interrupts are disabled as soon as the processor enters the halt mode debug state. However, the state of the I and F bits in the Program Status Register (PSR) are not affected.

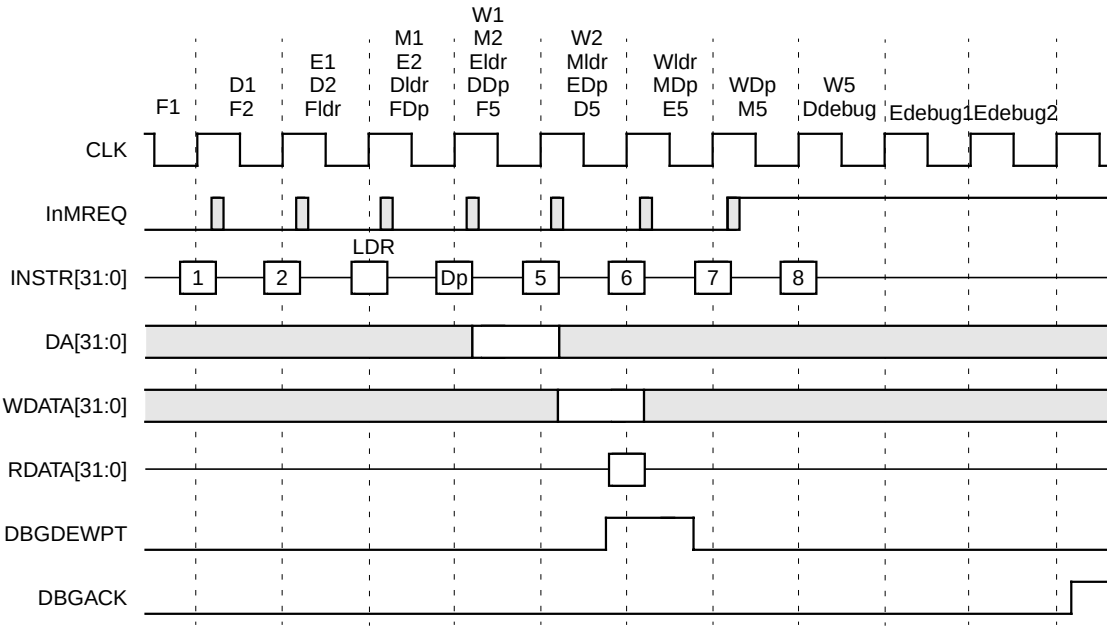
9.5.3 Watchpoints

Entry into the debug state following a watchpointed memory access is imprecise, because of the nature of the pipeline.

To extend functionality of the EmbeddedICE-RT logic, you can build external logic, such as external watchpoint comparators. The external logic output must be applied to the DBGDEWPT input. This signal is ORed with the internally generated Watchpoint signal before it is applied to the ARM9E-S core control logic. The timing of the input makes it unlikely that data-dependent external watchpoints can occur.

After a watchpointed access, the next instruction in the processor pipeline is always allowed to complete execution. When this instruction is a single-cycle data-processing instruction, entry into the debug state is delayed for one cycle while the instruction completes. Figure 9.8 illustrates the timing of debug entry after a watchpointed LDR instruction.

Figure 9.8 Watchpoint Entry with Data Processing Instruction



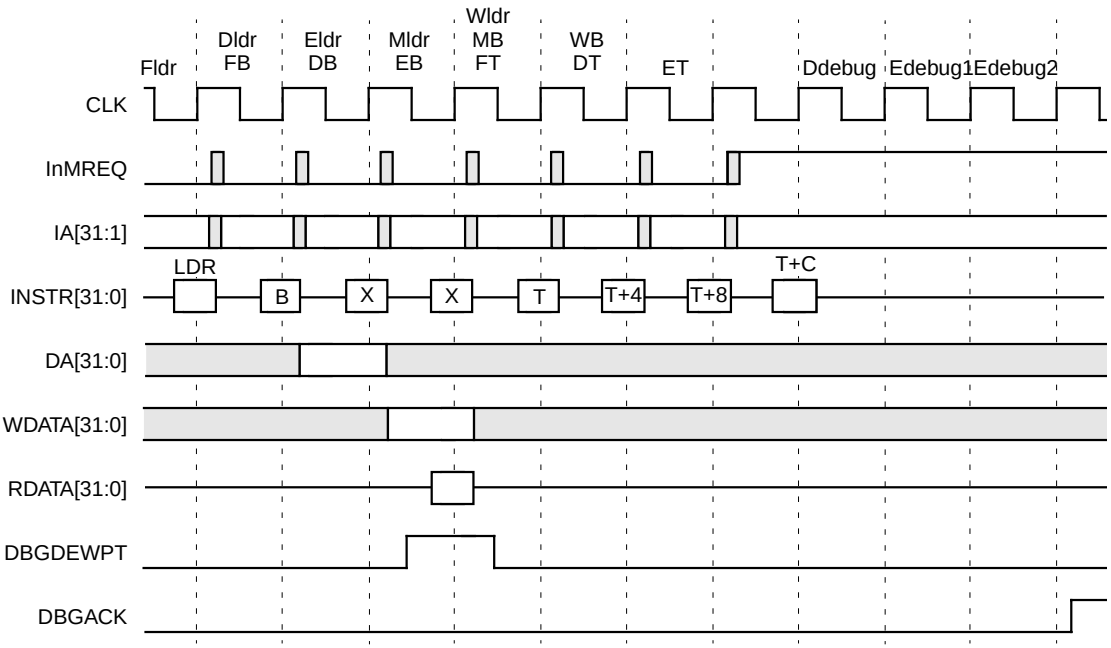
Although instruction 5 enters the execute stage, it is not executed, and there is no state update as a result of this instruction. When the debugging session is complete, normal operation involves returning to instruction 5 because it has not executed yet.

The instruction following the instruction that generated the watchpoint might modify the Program Counter (PC). If this happens, you cannot

determine the instruction that caused the watchpoint. However, you can always restart the processor. The timing diagram in Figure 9.9 shows debug entry after a watchpoint when the next instruction is a branch.

After the processor enters the debug state, you can interrogate the ARM9E-S processor core to determine its state. In the case of a watchpoint, the PC contains a value that is five instructions after the address of the next instruction to be executed. Therefore, upon entry into the debug state, if the instruction `SUB PC, PC, #20` is scanned in and the processor restarts, execution flow returns to the next instruction in the code sequence.

Figure 9.9 Watchpoint Entry with Branch



9.5.4 Watchpoints and Exceptions

If a watchpointed data access is also aborted, the watchpoint condition is registered and the exception entry sequence is performed. Then the processor enters the debug state. If there is an interrupt pending, the ARM9E-S allows the exception entry sequence to occur and then enters the debug state.

9.5.5 Debug Request

A debug request can take place through the EmbeddedICE-RT logic or by asserting the EDBGRQ signal. The request is synchronized and passed to the processor. A debug request takes priority over any pending interrupt. Following synchronization, the processor core enters the debug state after the instruction that is currently in the execute stage finishes both the memory and write stages. While waiting for the instruction to complete, the processor allows no more instructions to enter the execute stage.

Note: If EDBGRQ is asserted while the processor is operating in monitor mode, the processor enters the debug state as if operating in halt mode.

9.5.6 Actions of the ARM9E-S in Debug State

When the ARM9E-S is in the debug state, both memory interfaces indicate internal cycles. This ensures that the tightly coupled SRAM within the ARM946E-S and the AHB interface are both quiescent, allowing the rest of the AHB system to ignore the ARM9E-S and function normally. Because the rest of the system continues operation, the ARM9E-S ignores aborts and interrupts.

The nRESET signal must be held stable during debug. If the system applies reset to the ARM946E-S (nRESET is driven LOW), the state of the ARM9E-S changes without the debugger knowing about it.

9.6 Determining the Core and System State

When the ARM946E-S is in the debug state, you can examine the core and system state by forcing Load and Store Multiple instructions into the instruction pipeline.

Before you examine the core and system state, the debugger must check the EmbeddedICE-RT Debug Status register (bit 4) and determine whether the processor entered debug from the Thumb state or the ARM state. When bit 4 is HIGH, it indicates the processor was in the Thumb state. When bit 4 is LOW, the processor was in the ARM state.

9.7 Real-Time Debug

The ARM9E-S processor contains logic that permits you to debug a system without completely stopping the processor. This allows servicing critical interrupt routines to continue while the debugger interrogates the processor. Setting bit 4 of the Debug Control register enables the ARM9E-S real-time debug features. When set, this bit configures the EmbeddedICE-RT logic so a breakpoint or watchpoint causes the processor to enter abort mode and take the Prefetch Abort or Data Abort vectors, respectively.

The following restrictions apply when the ARM9E-S processor is configured for real-time debugging:

- Breakpoints/watchpoints cannot be data-dependent. No support is provided for the range and chain functionality.
  Breakpoints/watchpoints are based only on:
  - Instruction/data addresses
  - External watchpoint conditioner (DBGEXTERN)
  - User/privileged mode access (DnTRANS/InTRANS)
  - Read/write access (watchpoints)
  - Access size (breakpoints: ITBIT, watchpoints: DMAS[1:0]).
- Single-step hardware is not enabled.
- External breakpoints/watchpoints are not supported.
- Use the vector catching hardware, but you must not configure it to catch the Prefetch or Data Abort exceptions.
- No support is provided for mixing halt mode/monitor mode debug functionality. When the processor is configured for monitor mode, asserting the external EDBGRQ signal or setting the internal EDBGRQ bit causes unpredictable behavior.

If an abort is generated in monitor mode, the abort is recorded in the CP14 Debug Status register (bit 0). For more information about this register, including its format and bit definitions, see Section 3.4.2, “Debug Status Register (C2)” in this manual.

Because the monitor mode debug does not put the processor into the debug state, you must change the contents of the watchpoint registers while external memory accesses are taking place. If the watchpoint registers are updated during a memory access, all matches from the affected watchpoint unit using the register are disabled for that update cycle.

If false matches can occur during changes to the watchpoint registers (due to old data in some registers and new data in others), then you must do the following:

1. Disable the watchpoint unit using the Control register for that watchpoint unit.
2. Change the other registers.
3. Re-enable the watchpoint unit by rewriting the Control register.

9.8 ARM9E-S Clock Domains

The ARM9E-S processor has a single clock, CLK, that is qualified by two clock enables:

- SYSCLKEN controls access to the memory system
- DBGTCKEN controls debug operations

During normal operation, SYSCLKEN conditions CLK to clock the processor. When the ARM946E-S is in the debug state, DBGTCKEN conditions CLK to clock the processor.

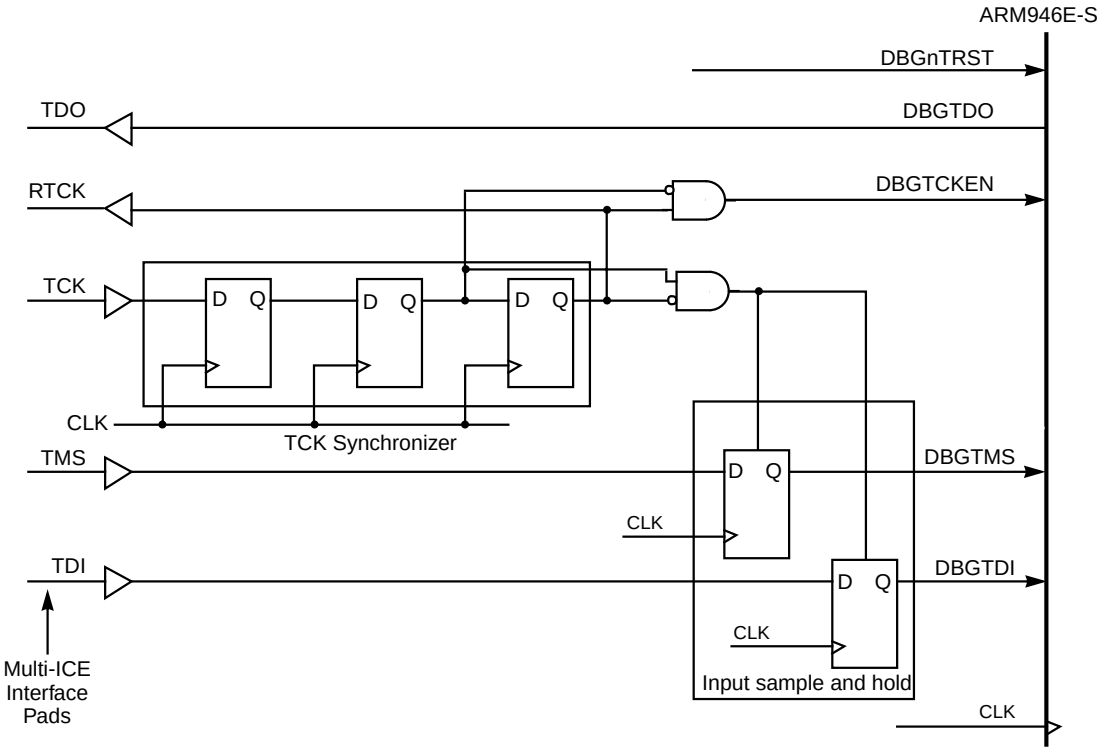
9.9 Synchronizing Debug Clocks

The ARM Multi-ICE debug agent directly supports one or more cores within an ASIC design.

External synchronization is required for the system debug and test clock inputs to the ARM946E-S. To synchronize the ARM946E-S with off-chip debug clocking, you must use a three-stage synchronizer. The off-chip device (for example, Multi-ICE) issues a TCK signal, and waits for the Returned TCK (RTCK) signal to come back. Synchronization is maintained because the off-chip device does not progress to the next TCK until after RTCK is received.

Figure 9.10 shows the clock synchronization logic.

Figure 9.10 Clock Synchronization Logic



Chapter 10
ETM Interface

This chapter describes the ARM946E-S Embedded Trace Macrocell (ETM) interface. It contains the following sections:

10.1 About the ETM

The ARM946E-S supports the connection of an optional external Embedded Trace Macrocell (ETM) to provide real-time tracing of ARM946E-S instructions and data in an embedded system.

The ETM consists of a trace port and triggering facilities.

10.1.1 Trace Port

The ETM compresses the trace information and exports it through the Trace port. An external Trace Port Analyzer (TPA) captures the trace information.

A trace protocol has been developed to provide a real-time trace capability for processor cores that are deeply embedded in much larger ASIC designs. As the ASIC typically includes significant amounts of on-chip memory, you cannot determine how the processor core is operating simply by observing the pins of the ASIC. A trace port is required to confirm the performance of the processor while in operation.

10.1.2 Triggering Facilities

An extensible specification exists that allows you to specify the exact set of trigger resources required for a particular application. Resources include address and data comparators, counters, and sequencers. For more information, see the *Embedded Trace Macrocell (Rev 1) Specification* available from ARM Limited.

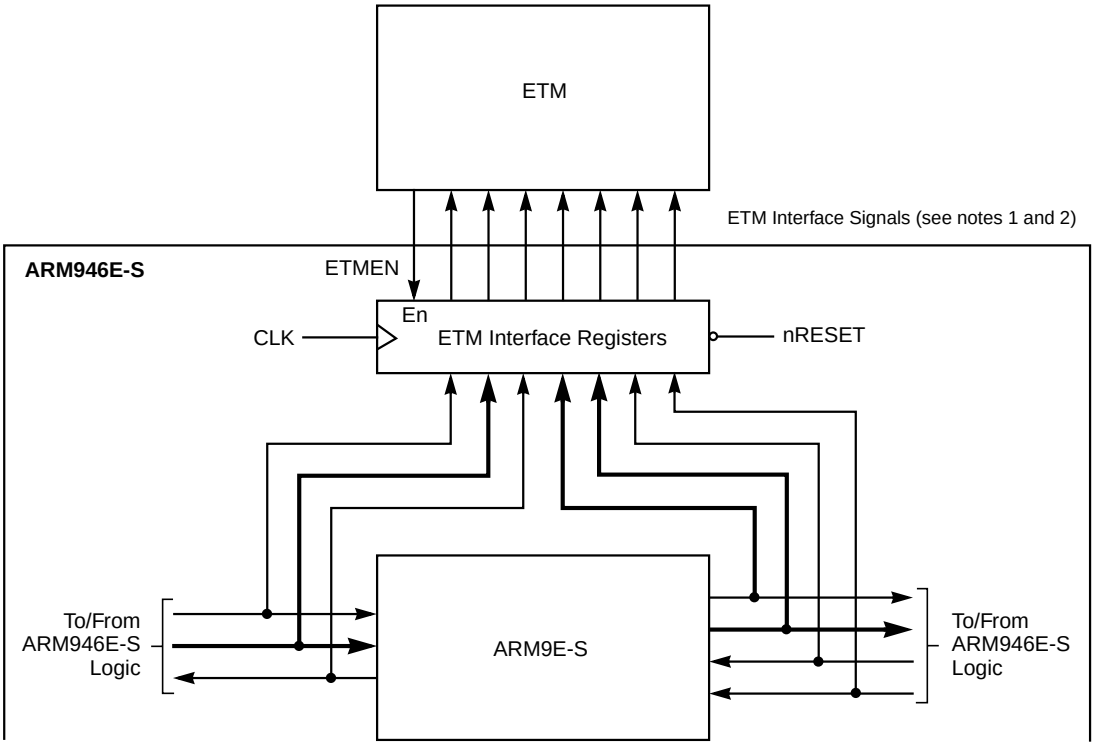
10.2 ETM Interface

The ETM interface is primarily one way. To provide code tracing, the ETM must monitor various processor inputs and outputs. The required inputs and outputs are collected and driven out from the ARM946E-S as the ETM interface.

ETM interface outputs are pipelined by a single clock cycle to provide early output timing and to isolate any ETM input load from the critical ARM946E-S signals. The latency of the pipelined outputs does not affect ETM trace behavior, because all outputs are delayed by the same amount.

Figure 10.1 shows the ARM946E-S ETM interface.

Figure 10.1 ARM946E-S ETM Interface



1. For a list of ETM signals and definitions, see Section 2.12, “ETM Interface Signals.”
2. For timing information, see Section Figure A.10, “ETM Interface Timing,”

10.3 Enabling the ETM Interface

The only input to the ETM interface is an enable signal that allows the required processor I/O to be driven to/from the ARM946E-S. ETMEN is the ETM interface enable signal. When ETMEN is HIGH, the ETM interface is enabled and the outputs are driven so that an external ETM can begin code tracing.

When the ETMEN input is driven LOW, the ETM interface outputs are held at their last value before the interface is disabled. At reset, all ETM interface outputs are reset LOW.

The ETM normally drives the ETMEN input. It is driven HIGH when you have programmed the ETM using its TAP controller.

Note: If you do not use an ETM in an embedded ARM946E-S design, tie the ETMEN input LOW to save power.

Chapter 11
Test Support

This chapter describes the test methodology used for the ARM946E-S synthesized logic and tightly coupled SRAM. It contains the following sections:

11.1 About the ARM946E-S Test Methodology

To achieve a high level of fault coverage, you can use scan insertion and Automatic Test Pattern Generation (ATPG) techniques on the ARM9E-S processor core and ARM946E-S control logic as part of the synthesis flow. You can use BIST to provide high fault coverage of the compiled SRAM.

11.2 Scan Insertion and ATPG

Scan insertion requires that all register elements be replaced by scannable versions that are then connected into a number of large scan chains. These scan chains are used to set up data patterns on the combinatorial logic between the registers and to capture the logic outputs. The logic outputs are then scanned out while the next data pattern is scanned in.

After scan insertion, you can use ATPG tools to create the necessary scan patterns to test the logic. Using this technique, you can achieve very high fault coverage for the standard cell combinatorial logic, typically, in the 95–99% range.

Scan insertion does affect the area and the performance of the synthesized design. This affect is due to the larger scan register elements and the serial routing between them. However, to minimize the impact, scan insertion is performed early in the synthesis cycle and the design is reoptimized after the scan elements are in place.

11.3 BIST of Memory Arrays

BIST is performed at manufacturing test. There is no software interface that will run the memory BIST. The BIST controller is accessible through the JTAG interface.

Appendix A
AC Parameters

This appendix lists the AC timing parameters for the ARM946E-S. It contains the following sections:

- Section A.1, “Timing Diagrams”
- Section A.2, “AC Timing Parameter Definitions”

A.1 Timing Diagrams

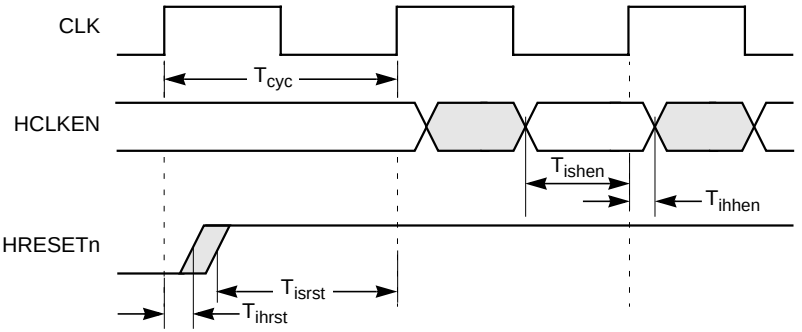
The timing diagrams in this section are:

- Clock, Reset, and AHB Enable Timing
- AHB Bus Request and Grant Related Timing
- AHB Bus Master Timing
- Coprocessor Interface Timing
- Debug Interface Timing
- JTAG Interface Timing
- DBGSDOUT to DBGTDO Timing
- Exception and Configuration Timing
- INTEST Wrapper Timing
- ETM Interface Timing

Refer to Table A.1 on page A-10 for definitions of the parameters shown in the timing diagrams.

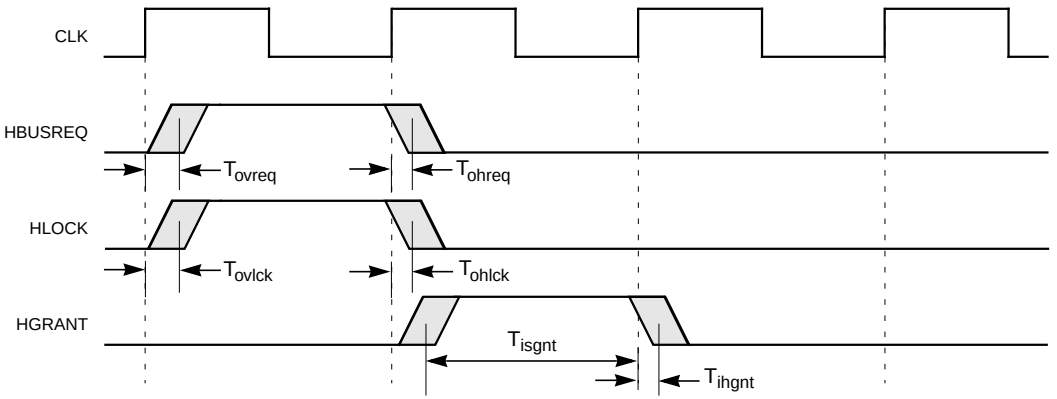
Clock, reset and AHB enable timing parameters are shown in Figure A.1.

Figure A.1 Clock, Reset, and AHB Enable Timing



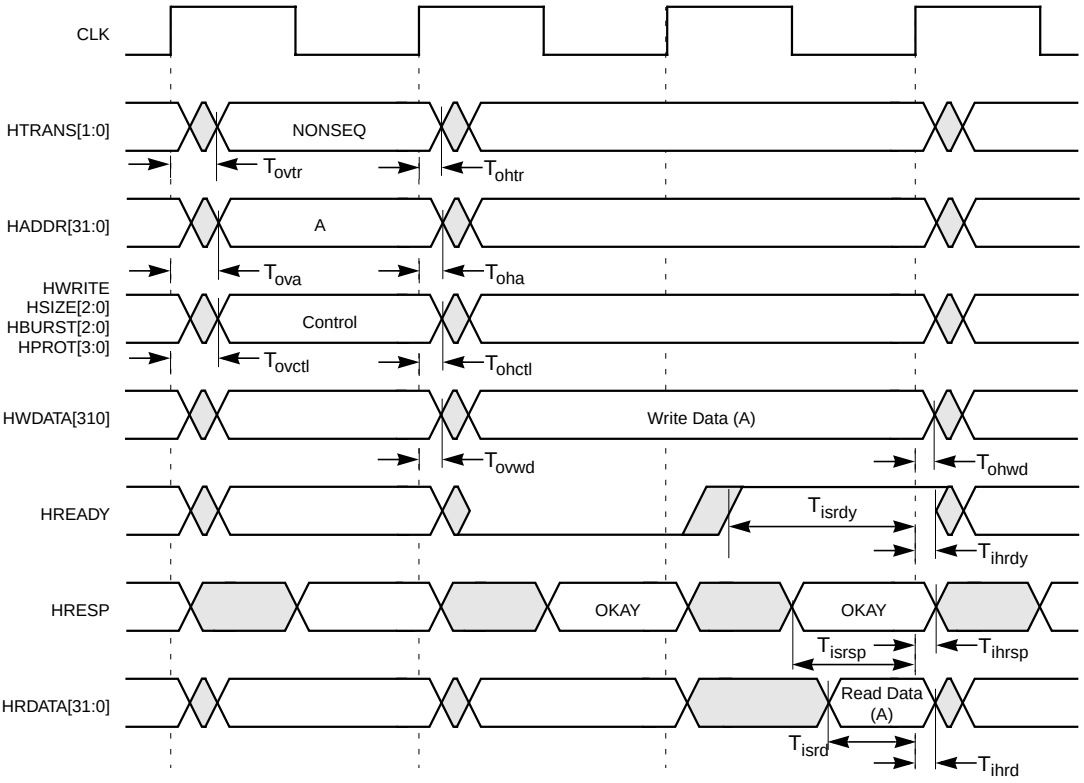
AHB bus request and grant related timing parameters are shown in Figure A.2.

Figure A.2 AHB Bus Request and Grant Related Timing



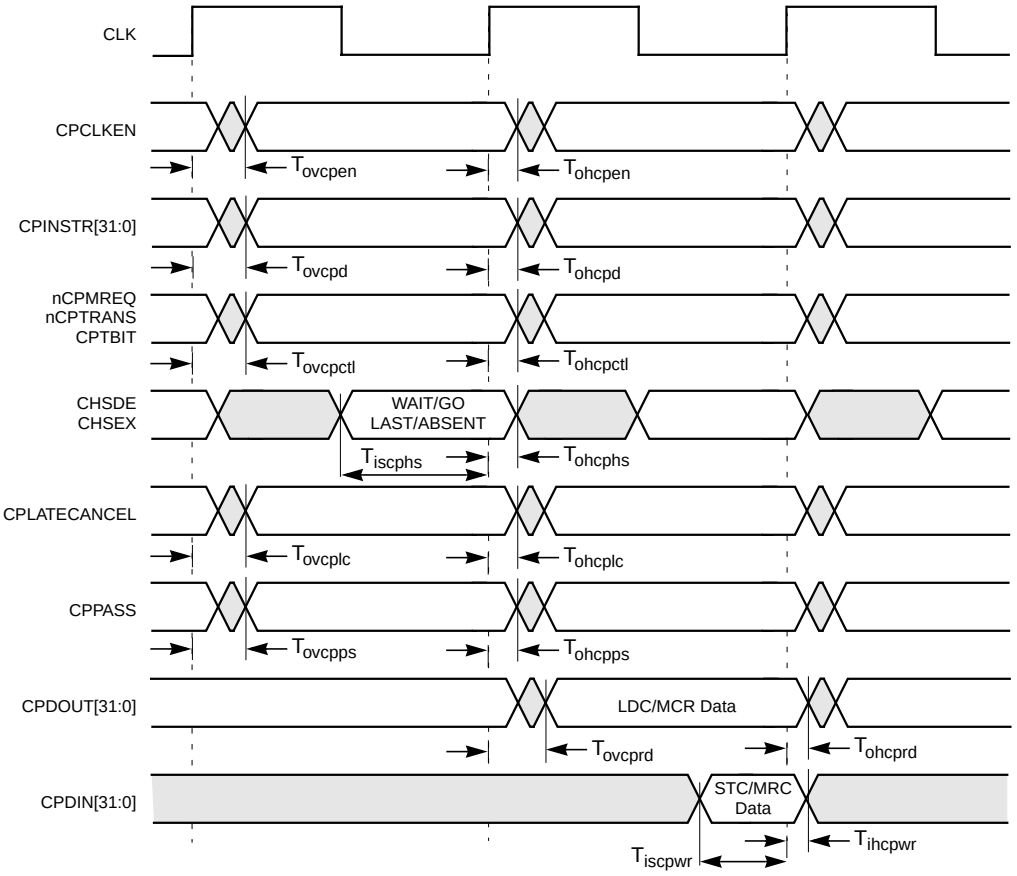
AHB bus master timing parameters are shown in Figure A.3.

Figure A.3 AHB Bus Master Timing



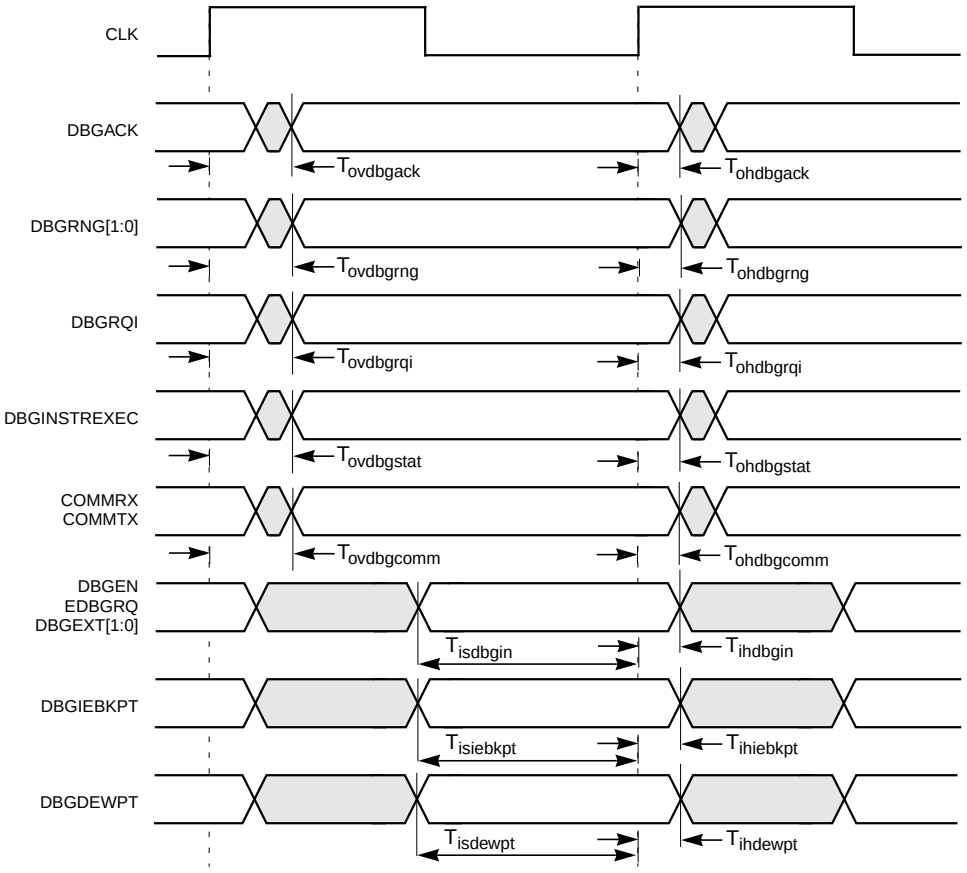
Coprocessor interface timing parameters are shown in Figure A.4.

Figure A.4 Coprocessor Interface Timing



Debug interface timing parameters are shown in Figure A.5.

Figure A.5 Debug Interface Timing



JTAG interface timing parameters are shown in Figure A.6.

Figure A.6 JTAG Interface Timing

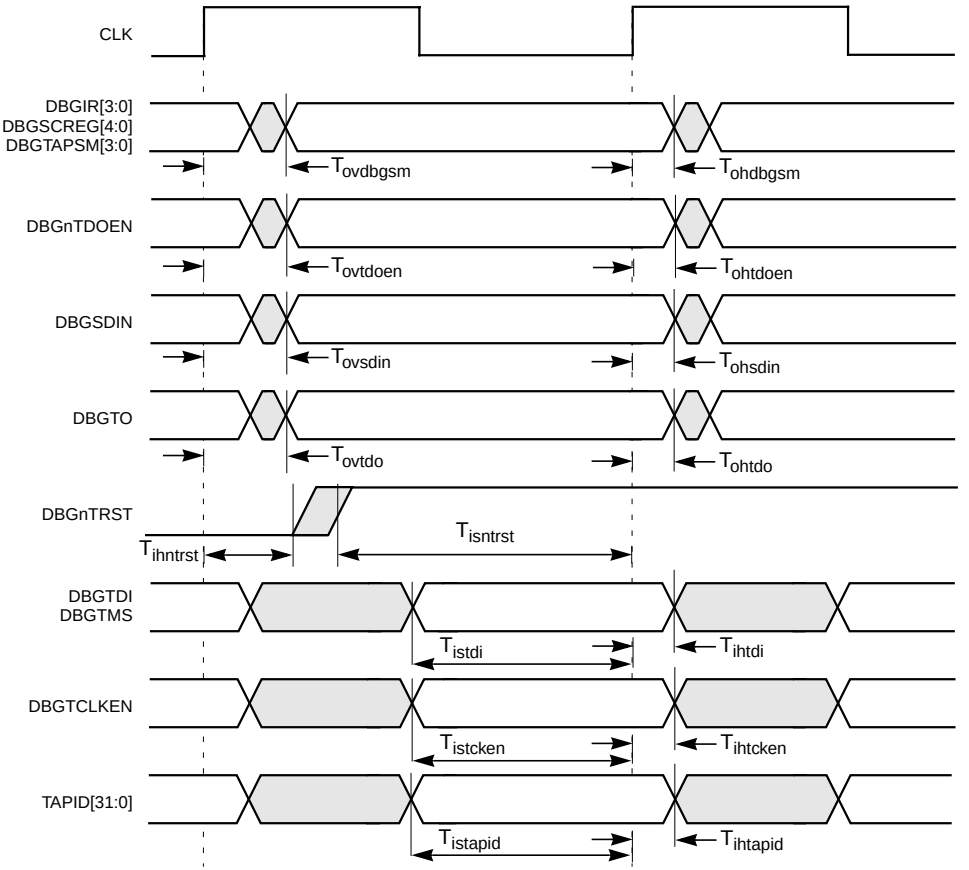
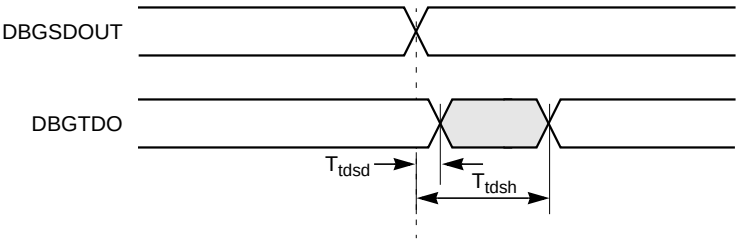


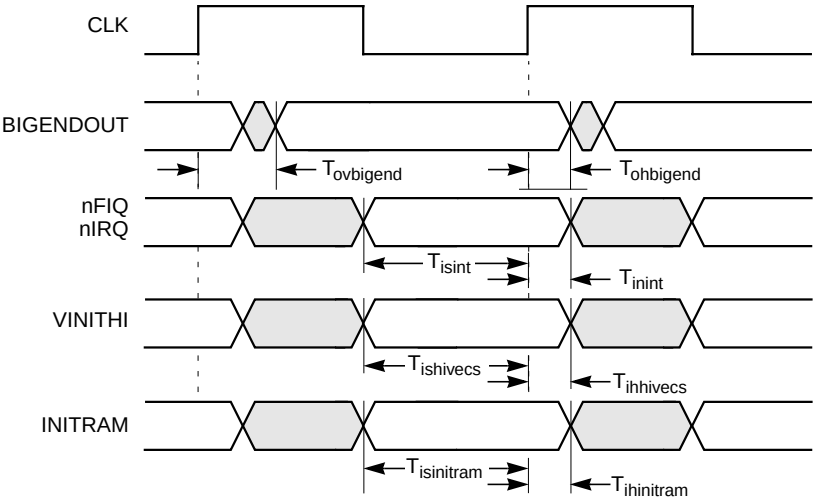
Figure A.7 shows a combinational path timing parameter that exists from the DBGSDOUT input to the DBGTDO output.

Figure A.7 DBGSDOUT to DBGTDO Timing



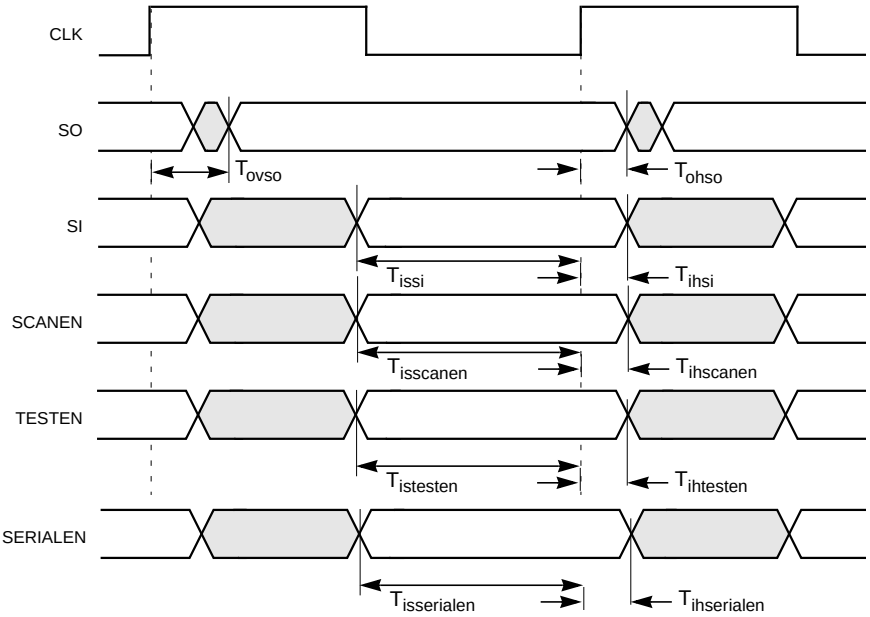
Exception and configuration timing parameters are shown in Figure A.8.

Figure A.8 Exception and Configuration Timing



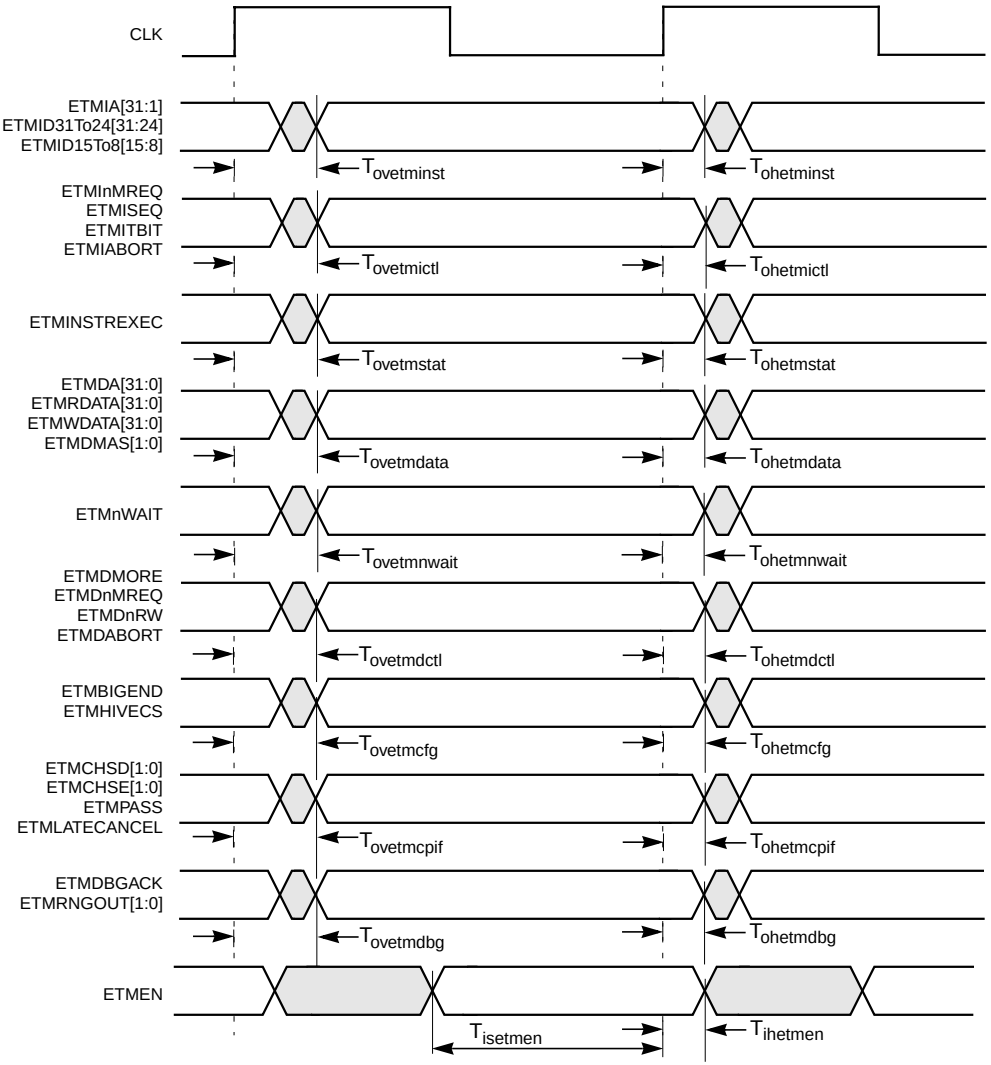
The INTEST wrapper timing parameters are shown in Figure A.9.

Figure A.9 INTEST Wrapper Timing



The ETM interface timing parameters are shown in Figure A.10.

Figure A.10 ETM Interface Timing



A.2 AC Timing Parameter Definitions

Table A.1 shows target AC parameters. All values are expressed as percentages of the CLK period at maximum operating frequency.

Note: The values quoted are relative to the rising clock edge after the clock skew for internal buffering has been added. Inputs given a 0% hold value therefore require a positive hold relative to the top-level clock input. The amount of hold required is equivalent to the internal clock skew.

Table A.1 Timing Parameter Definitions

Symbol	Parameter	Min	Max
T_{cyc}	CLK cycle time	100%	–
T_{ishen}	HCLKEN input setup to rising CLK	85%	–
T_{ihhen}	HCLKEN input hold from rising CLK	–	0%
T_{isrst}	HRESETn deassertion input setup to rising CLK	90%	–
T_{ihrst}	HRESETn deassertion input hold from rising CLK	–	0%
T_{ovreq}	Rising CLK to HBUSREQ valid	–	30%
T_{ohreq}	HBUSREQ hold time from rising CLK	>0%	–
T_{ovlck}	Rising CLK to HLOCK valid	–	30%
T_{ohlck}	HLOCK hold time from rising CLK	>0%	–
T_{isgnt}	HGRANT input setup to rising CLK	40%	–
T_{ihgnt}	HGRANT input hold from rising CLK	–	0%
T_{ovtr}	Rising CLK to HTRANS[1:0] valid	–	30%
T_{ohtr}	HTRANS[1:0] hold time from rising CLK	>0%	–
T_{ova}	Rising CLK to HADDR[31:0] valid	–	30%
T_{oha}	HADDR[31:0] hold time from rising CLK	>0%	–
T_{ovctl}	Rising CLK to AHB control signals valid	–	30%
T_{ohctl}	AHB control signals hold time from rising CLK	>0%	–

Table A.1 Timing Parameter Definitions (Cont.)

Symbol	Parameter	Min	Max
T_{ovwd}	Rising CLK to HWDATA[31:0] valid	–	30%
T_{ohwd}	HWDATA[31:0] hold time from rising CLK	>0%	–
T_{isrdy}	HREADY input setup to rising CLK	75%	–
T_{ihrdy}	HREADY input hold from rising CLK	–	0%
T_{isrsp}	HRESP[1:0] input setup to rising CLK	50%	–
T_{ihrsp}	HRESP[1:0] input hold from rising CLK	–	0%
T_{isrd}	HRDATA[31:0] input setup to rising CLK	40%	–
T_{ihrd}	HRDATA[31:0] input hold from rising CLK	–	0%
T_{ovcpen}	Rising CLK to CPCLKEN valid	–	30%
T_{ohcpen}	CPCLKEN hold time from rising CLK	>0%	–
T_{ovcpid}	Rising CLK to CPINSTR[31:0] valid	–	30%
T_{ohcpid}	CPINSTR[31:0] hold time from rising CLK	>0%	–
$T_{ovcpctl}$	Rising CLK to transaction control valid	–	30%
$T_{ohcpctl}$	Transaction control hold time from rising CLK	>0%	–
T_{iscphs}	Coprocessor handshake input setup to rising CLK	50%	–
T_{ihcphs}	Coprocessor handshake input hold from rising CLK	–	0%
T_{ovcplc}	Rising CLK to CPLATECANCEL valid	–	30%
T_{ohcplc}	CPLATECANCEL hold time from rising CLK	>0%	–
T_{ovcpps}	Rising CLK to CPPASS valid	–	30%
T_{ohcpps}	CPPASS hold time from rising CLK	>0%	–
T_{ovcprd}	Rising CLK to CPDOUT[31:0] valid	–	30%
T_{ohcprd}	CPDOUT[31:0] hold time from rising CLK	>0%	–
T_{iscpwr}	CPDIN[31:0] input setup to rising CLK	40%	–
T_{ihcpwr}	CPDIN[31:0] input hold from rising CLK	–	0%
$T_{ovdbgack}$	Rising CLK to DBGACK valid	–	60%

Table A.1 Timing Parameter Definitions (Cont.)

Symbol	Parameter	Min	Max
$T_{ohdbgack}$	DBGACK hold time from rising CLK	>0%	–
$T_{ovdbgrng}$	Rising CLK to DBGRNG[1:0] valid	–	60%
$T_{ohdbgrng}$	DBGRNG[1:0] hold time from rising CLK	>0%	–
$T_{ovdbgrqi}$	Rising CLK to DBGRQI valid	–	45%
$T_{ohdbgrqi}$	DBGRQI hold time from rising CLK	>0%	–
$T_{ovdbgstat}$	Rising CLK to DBGINSTREXEC valid	–	30%
$T_{ohdbgstat}$	DBGINSTREXEC hold time from rising CLK	>0%	–
$T_{ovdbgcomm}$	Rising CLK to comms channel outputs valid	–	30%
$T_{ohdbgcomm}$	Comms channel outputs hold time from rising CLK	>0%	–
$T_{isdbgin}$	Debug inputs input setup to rising CLK	30%	–
$T_{ihdbgin}$	Debug inputs input hold from rising CLK	–	0%
$T_{isiebkpt}$	DBGIEBKPT input setup to rising CLK	20%	–
$T_{ihiebkpt}$	DBGIEBKPT input hold from rising CLK	–	0%
$T_{isdewpt}$	DBGDEWPT input setup to rising CLK	20%	–
$T_{ihdewpt}$	DBGDEWPT input hold from rising CLK	–	0%
$T_{ovdbgsm}$	Rising CLK to debug state valid	–	30%
$T_{ohdbgsm}$	Debug state hold time from rising CLK	>0%	–
$T_{ovtdoen}$	Rising CLK to DBGnTDOEN valid	–	40%
$T_{ohtdoen}$	DBGnTDOEN hold time from rising CLK	>0%	–
T_{ovsdin}	Rising CLK to DBGSDIN valid	–	20%
T_{ohsdin}	DBGSDIN hold time from rising CLK	>0%	–
T_{ovtdo}	Rising CLK to DBGTDO valid	–	65%
T_{ohtdo}	DBGTDO hold time from rising CLK	>0%	–
$T_{isntrst}$	DBGnTRST deasserted input setup to rising CLK	35%	–
$T_{ihntrst}$	DBGnTRST input hold from rising CLK	–	0%

Table A.1 Timing Parameter Definitions (Cont.)

Symbol	Parameter	Min	Max
T_{istdi}	Tap state control input setup to rising CLK	25%	–
T_{ihtdi}	Tap state control input hold from rising CLK	–	0%
$T_{istcken}$	DBGTCKEN input setup to rising CLK	50%	–
$T_{ihtcken}$	DBGTCKEN input hold from rising CLK	–	0%
$T_{istapid}$	TAPID[31:0] input setup to rising CLK	20%	–
$T_{ihtapid}$	TAPID[31:0] input hold from rising CLK	–	0%
T_{dsd}	DBGTDO delay from DBGSDOUTBS changing	–	30%
T_{dsh}	DBGTDO hold time from DBGSDOUTBS changing	>0%	–
$T_{ovbigend}$	Rising CLK to BIGENDOUT valid	–	30%
$T_{ohbigend}$	BIGENDOUT hold time from rising CLK	>0%	–
T_{isint}	Interrupt input setup to rising CLK	15%	–
T_{ihint}	Interrupt input hold from rising CLK	–	0%
$T_{ishivecs}$	VINITHI input setup to rising CLK	95%	–
$T_{ihhivecs}$	VINITHI input hold from rising CLK	–	0%
$T_{isinitram}$	INITRAM input setup to rising CLK	95%	–
$T_{ihinitram}$	INITRAM input hold from rising CLK	–	0%
T_{ovso}	Rising CLK to SO valid	–	30%
T_{ohso}	SO hold time from rising CLK	>0%	–
T_{issi}	SI input setup to rising CLK	95%	–
T_{ihsi}	SI input hold from rising CLK	–	0%
$T_{isscanen}$	SCANEN input setup to rising CLK	95%	–
$T_{ihscanen}$	SCANEN input hold from rising CLK	–	0%
$T_{isserialen}$	SERIALEN input setup to rising CLK	95%	–
$T_{ihserialen}$	SERIALEN input hold from rising CLK	–	0%
$T_{ovetminst}$	Rising CLK to ETM instruction interface valid	–	30%

Table A.1 Timing Parameter Definitions (Cont.)

Symbol	Parameter	Min	Max
$T_{ohetminst}$	ETM instruction interface hold time from rising CLK	>0%	–
$T_{ovetmictl}$	Rising CLK to ETM instruction control valid	–	30%
$T_{ohetmictl}$	ETM instruction control hold time from rising CLK	>0%	–
$T_{ovetmstat}$	Rising CLK to ETMINSTREXEC valid	–	30%
$T_{ohetmstat}$	ETMINSTREXEC hold time from rising CLK	>0%	–
$T_{ovetmdata}$	Rising CLK to ETM data interface valid	–	30%
$T_{ohetmdata}$	ETM data interface hold time from rising CLK	>0%	–
$T_{ovetmnwait}$	Rising CLK to ETMnWAIT valid	–	30%
$T_{ohetmnwait}$	ETMnWAIT hold time from rising CLK	>0%	–
$T_{ovetmdctl}$	Rising CLK to ETM data control valid	–	30%
$T_{ohetmdctl}$	ETM data control hold time from rising CLK	>0%	–
$T_{ovetmcfg}$	Rising CLK to ETM configuration valid	–	30%
$T_{ohetmcfg}$	ETM configuration hold time from rising CLK	>0%	–
$T_{ovetmcpif}$	Rising CLK to ETM coprocessor signals valid	–	30%
$T_{ohetmcpif}$	ETM coprocessor signals hold time from rising CLK	>0%	–
$T_{ovetmdbg}$	Rising CLK to ETM debug signals valid	–	30%
$T_{ohetmdbg}$	ETM debug signals hold time from rising CLK	>0%	–
$T_{isetmen}$	ETMEN input setup to rising CLK	50%	–
$T_{ihetmen}$	ETMEN input hold from rising CLK	–	0%

Note: The VINITHI pin is specified as 95% of the cycle because it is for input configuration during reset and can be considered static.

Index

T

U

W

Customer Feedback

We would appreciate your feedback on this document. Please copy the following page, add your comments, and fax it to us at the number shown.

If appropriate, please also fax copies of any marked-up pages from this document.

Important: Please include your name, phone number, fax number, and company address so that we may contact you directly for clarification or additional information.

Thank you for your help in improving the quality of our documents.

Reader's Comments

Fax your comments to: LSI Logic Corporation
Technical Publications
M/S E-198
Fax: 408.433.4333

Please tell us how you rate this document: *ARM946E-S Microprocessor Core with Cache Technical Manual.* Place a check mark in the appropriate blank for each category.

	Excellent	Good	Average	Fair	Poor
Completeness of information	____	____	____	____	____
Clarity of information	____	____	____	____	____
Ease of finding information	____	____	____	____	____
Technical content	____	____	____	____	____
Usefulness of examples and illustrations	____	____	____	____	____
Overall manual	____	____	____	____	____

What could we do to improve this document?

_____

_____

_____

_____

If you found errors in this document, please specify the error and page number. If appropriate, please fax a marked-up copy of the page(s).

_____

_____

_____

Please complete the information below so that we may contact you directly for clarification or additional information.

Name _____ Date _____

Telephone _____ Fax _____

Title _____

Department _____ Mail Stop _____

Company Name _____

Street _____

City, State, Zip _____

Customer Feedback

Customer Feedback

U.S. Distributors by State

A. E. Avnet Electronics
http://www.hh.avnet.com
B. M. Bell Microproducts, Inc. (for HAB's)
http://www.bellmicro.com
I. E. Insight Electronics
http://www.insight-electronics.com
W. E. Wyle Electronics
http://www.wyle.com

Alabama
Daphne
I. E. Tel: 334.626.6190
Huntsville
A. E. Tel: 256.837.8700
B. M. Tel: 256.705.3559
I. E. Tel: 256.830.1222
W. E. Tel: 800.964.9953

Alaska
A. E. Tel: 800.332.8638

Arizona
Phoenix
A. E. Tel: 480.736.7000
B. M. Tel: 602.267.9551
W. E. Tel: 800.528.4040
Tempe
I. E. Tel: 480.829.1800
Tucson
A. E. Tel: 520.742.0515

Arkansas
W. E. Tel: 972.235.9953

California
Agoura Hills
B. M. Tel: 818.865.0266
Granite Bay
B. M. Tel: 916.523.7047
Irvine
A. E. Tel: 949.789.4100
B. M. Tel: 949.470.2900
I. E. Tel: 949.727.3291
W. E. Tel: 800.626.9953
Los Angeles
A. E. Tel: 818.594.0404
W. E. Tel: 800.288.9953
Sacramento
A. E. Tel: 916.632.4500
W. E. Tel: 800.627.9953
San Diego
A. E. Tel: 858.385.7500
B. M. Tel: 858.597.3010
I. E. Tel: 800.677.6011
W. E. Tel: 800.829.9953
San Jose
A. E. Tel: 408.435.3500
B. M. Tel: 408.436.0881
I. E. Tel: 408.952.7000
Santa Clara
W. E. Tel: 800.866.9953
Woodland Hills
A. E. Tel: 818.594.0404
Westlake Village
I. E. Tel: 818.707.2101

Colorado
Denver
A. E. Tel: 303.790.1662
B. M. Tel: 303.846.3065
W. E. Tel: 800.933.9953
Englewood
I. E. Tel: 303.649.1800
Idaho Springs
B. M. Tel: 303.567.0703

Connecticut
Cheshire
A. E. Tel: 203.271.5700
I. E. Tel: 203.272.5843
Wallingford
W. E. Tel: 800.605.9953

Delaware
North/South
A. E. Tel: 800.526.4812
Tel: 800.638.5988
B. M. Tel: 302.328.8968
W. E. Tel: 856.439.9110

Florida
Altamonte Springs
B. M. Tel: 407.682.1199
I. E. Tel: 407.834.6310
Boca Raton
I. E. Tel: 561.997.2540
Bonita Springs
B. M. Tel: 941.498.6011
Clearwater
I. E. Tel: 727.524.8850
Fort Lauderdale
A. E. Tel: 954.484.5482
W. E. Tel: 800.568.9953
Miami
B. M. Tel: 305.477.6406
Orlando
A. E. Tel: 407.657.3300
W. E. Tel: 407.740.7450
Tampa
W. E. Tel: 800.395.9953
St. Petersburg
A. E. Tel: 727.507.5000

Georgia
Atlanta
A. E. Tel: 770.623.4400
B. M. Tel: 770.980.4922
W. E. Tel: 800.876.9953
Duluth
I. E. Tel: 678.584.0812

Hawaii
A. E. Tel: 800.851.2282

Idaho
A. E. Tel: 801.365.3800
W. E. Tel: 801.974.9953

Illinois
North/South
A. E. Tel: 847.797.7300
Tel: 314.291.5350
Chicago
B. M. Tel: 847.413.8530
W. E. Tel: 800.853.9953
Schaumburg
I. E. Tel: 847.885.9700

Indiana
Fort Wayne
I. E. Tel: 219.436.4250
W. E. Tel: 888.358.9953
Indianapolis
A. E. Tel: 317.575.3500

Iowa
W. E. Tel: 612.853.2280
Cedar Rapids
A. E. Tel: 319.393.0033

Kansas
W. E. Tel: 303.457.9953
Kansas City
A. E. Tel: 913.663.7900
Lenexa
I. E. Tel: 913.492.0408

Kentucky
W. E. Tel: 937.436.9953
Central/Northern/ Western
A. E. Tel: 800.984.9503
Tel: 800.767.0329
Tel: 800.829.0146

Louisiana
W. E. Tel: 713.854.9953
North/South
A. E. Tel: 800.231.0253
Tel: 800.231.5775

Maine
A. E. Tel: 800.272.9255
W. E. Tel: 781.271.9953

Maryland
Baltimore
A. E. Tel: 410.720.3400
W. E. Tel: 800.863.9953
Columbia
B. M. Tel: 800.673.7461
I. E. Tel: 410.381.3131

Massachusetts
Boston
A. E. Tel: 978.532.9808
W. E. Tel: 800.444.9953
Burlington
I. E. Tel: 781.270.9400
Marlborough
B. M. Tel: 800.673.7459
Woburn
B. M. Tel: 800.552.4305

Michigan
Brighton
I. E. Tel: 810.229.7710
Detroit
A. E. Tel: 734.416.5800
W. E. Tel: 888.318.9953
Clarkston
B. M. Tel: 877.922.9363

Minnesota
Champlin
B. M. Tel: 800.557.2566
Eden Prairie
B. M. Tel: 800.255.1469
Minneapolis
A. E. Tel: 612.346.3000
W. E. Tel: 800.860.9953
St. Louis Park
I. E. Tel: 612.525.9999

Mississippi
A. E. Tel: 800.633.2918
W. E. Tel: 256.830.1119

Missouri
W. E. Tel: 630.620.0969
St. Louis
A. E. Tel: 314.291.5350
I. E. Tel: 314.872.2182

Montana
A. E. Tel: 800.526.1741
W. E. Tel: 801.974.9953

Nebraska
A. E. Tel: 800.332.4375
W. E. Tel: 303.457.9953

Nevada
Las Vegas
A. E. Tel: 800.528.8471
W. E. Tel: 702.765.7117

New Hampshire
A. E. Tel: 800.272.9255
W. E. Tel: 781.271.9953

New Jersey
North/South
A. E. Tel: 201.515.1641
Tel: 609.222.6400
Mt. Laurel
I. E. Tel: 856.222.9566
Pine Brook
B. M. Tel: 973.244.9668
W. E. Tel: 800.862.9953
Parsippany
I. E. Tel: 973.299.4425
Wayne
W. E. Tel: 973.237.9010

New Mexico
W. E. Tel: 480.804.7000
Albuquerque
A. E. Tel: 505.293.5119

U.S. Distributors by State (Continued)

New York
Hauppauge
I. E. Tel: 516.761.0960
Long Island
A. E. Tel: 516.434.7400
W. E. Tel: 800.861.9953
Rochester
A. E. Tel: 716.475.9130
I. E. Tel: 716.242.7790
W. E. Tel: 800.319.9953
Smithtown
B. M. Tel: 800.543.2008
Syracuse
A. E. Tel: 315.449.4927

North Carolina
Raleigh
A. E. Tel: 919.859.9159
I. E. Tel: 919.873.9922
W. E. Tel: 800.560.9953

North Dakota
A. E. Tel: 800.829.0116
W. E. Tel: 612.853.2280

Ohio
Cleveland
A. E. Tel: 216.498.1100
W. E. Tel: 800.763.9953
Dayton
A. E. Tel: 614.888.3313
I. E. Tel: 937.253.7501
W. E. Tel: 800.575.9953
Strongsville
B. M. Tel: 440.238.0404
Valley View
I. E. Tel: 216.520.4333

Oklahoma
W. E. Tel: 972.235.9953
Tulsa
A. E. Tel: 918.459.6000
I. E. Tel: 918.665.4664

Oregon
Beaverton
B. M. Tel: 503.524.1075
I. E. Tel: 503.644.3300
Portland
A. E. Tel: 503.526.6200
W. E. Tel: 800.879.9953

Pennsylvania
Mercer
I. E. Tel: 412.662.2707
Philadelphia
A. E. Tel: 800.526.4812
B. M. Tel: 877.351.2355
W. E. Tel: 800.871.9953
Pittsburgh
A. E. Tel: 412.281.4150
W. E. Tel: 440.248.9996

Rhode Island
A. E. 800.272.9255
W. E. Tel: 781.271.9953

South Carolina
A. E. Tel: 919.872.0712
W. E. Tel: 919.469.1502

South Dakota
A. E. Tel: 800.829.0116
W. E. Tel: 612.853.2280

Tennessee
W. E. Tel: 256.830.1119
East/West
A. E. Tel: 800.241.8182
Tel: 800.633.2918

Texas
Arlington
B. M. Tel: 817.417.5993
Austin
A. E. Tel: 512.219.3700
B. M. Tel: 512.258.0725
I. E. Tel: 512.719.3090
W. E. Tel: 800.365.9953
Dallas
A. E. Tel: 214.553.4300
B. M. Tel: 972.783.4191
W. E. Tel: 800.955.9953
El Paso
A. E. Tel: 800.526.9238
Houston
A. E. Tel: 713.781.6100
B. M. Tel: 713.917.0663
W. E. Tel: 800.888.9953
Richardson
I. E. Tel: 972.783.0800
Rio Grande Valley
A. E. Tel: 210.412.2047
Stafford
I. E. Tel: 281.277.8200

Utah
Centerville
B. M. Tel: 801.295.3900
Murray
I. E. Tel: 801.288.9001
Salt Lake City
A. E. Tel: 801.365.3800
W. E. Tel: 800.477.9953

Vermont
A. E. Tel: 800.272.9255
W. E. Tel: 716.334.5970

Virginia
A. E. Tel: 800.638.5988
W. E. Tel: 301.604.8488
Haymarket
B. M. Tel: 703.754.3399
Springfield
B. M. Tel: 703.644.9045

Washington
Kirkland
I. E. Tel: 425.820.8100
Maple Valley
B. M. Tel: 206.223.0080
Seattle
A. E. Tel: 425.882.7000
W. E. Tel: 800.248.9953

West Virginia
A. E. Tel: 800.638.5988

Wisconsin
Milwaukee
A. E. Tel: 414.513.1500
W. E. Tel: 800.867.9953
Wauwatosa
I. E. Tel: 414.258.5338

Wyoming
A. E. Tel: 800.332.9326
W. E. Tel: 801.974.9953

Sales Offices and Design Resource Centers

LSI Logic Corporation
Corporate Headquarters
1551 McCarthy Blvd
Milpitas CA 95035
Tel: 408.433.8000
Fax: 408.433.8989

NORTH AMERICA

California
Irvine
18301 Von Karman Ave
Suite 900
Irvine, CA 92612
♦ Tel: 949.809.4600
Fax: 949.809.4444

Pleasanton Design Center
5050 Hopyard Road, 3rd Floor
Suite 300
Pleasanton, CA 94588
Tel: 925.730.8800
Fax: 925.730.8700

San Diego
7585 Ronson Road
Suite 100
San Diego, CA 92111
Tel: 858.467.6981
Fax: 858.496.0548

Silicon Valley
1551 McCarthy Blvd
Sales Office
M/S C-500
Milpitas, CA 95035
♦ Tel: 408.433.8000
Fax: 408.954.3353
Design Center
M/S C-410
Tel: 408.433.8000
Fax: 408.433.7695

Wireless Design Center
11452 El Camino Real
Suite 210
San Diego, CA 92130
Tel: 858.350.5560
Fax: 858.350.0171

Colorado
Boulder
4940 Pearl East Circle
Suite 201
Boulder, CO 80301
♦ Tel: 303.447.3800
Fax: 303.541.0641

Colorado Springs
4420 Arrowswest Drive
Colorado Springs, CO 80907
Tel: 719.533.7000
Fax: 719.533.7020

Fort Collins
2001 Danfield Court
Fort Collins, CO 80525
Tel: 970.223.5100
Fax: 970.206.5549

Florida
Boca Raton
2255 Glades Road
Suite 324A
Boca Raton, FL 33431
Tel: 561.989.3236
Fax: 561.989.3237

Georgia
Alpharetta
2475 North Winds Parkway
Suite 200
Alpharetta, GA 30004
Tel: 770.753.6146
Fax: 770.753.6147

Illinois
Oakbrook Terrace
Two Mid American Plaza
Suite 800
Oakbrook Terrace, IL 60181
Tel: 630.954.2234
Fax: 630.954.2235

Kentucky
Bowling Green
1262 Chestnut Street
Bowling Green, KY 42101
Tel: 270.793.0010
Fax: 270.793.0040

Maryland
Bethesda
6903 Rockledge Drive
Suite 230
Bethesda, MD 20817
Tel: 301.897.5800
Fax: 301.897.8389

Massachusetts
Waltham
200 West Street
Waltham, MA 02451
♦ Tel: 781.890.0180
Fax: 781.890.6158

Burlington - Mint Technology
77 South Bedford Street
Burlington, MA 01803
Tel: 781.685.3800
Fax: 781.685.3801

Minnesota
Minneapolis
8300 Norman Center Drive
Suite 730
Minneapolis, MN 55437
♦ Tel: 612.921.8300
Fax: 612.921.8399

New Jersey
Red Bank
125 Half Mile Road
Suite 200
Red Bank, NJ 07701
Tel: 732.933.2656
Fax: 732.933.2643

Cherry Hill - Mint Technology
215 Longstone Drive
Cherry Hill, NJ 08003
Tel: 856.489.5530
Fax: 856.489.5531

New York
Fairport
550 Willowbrook Office Park
Fairport, NY 14450
Tel: 716.218.0020
Fax: 716.218.9010

North Carolina
Raleigh
Phase II
4601 Six Forks Road
Suite 528
Raleigh, NC 27609
Tel: 919.785.4520
Fax: 919.783.8909

Oregon
Beaverton
15455 NW Greenbrier Parkway
Suite 235
Beaverton, OR 97006
Tel: 503.645.0589
Fax: 503.645.6612

Texas
Austin
9020 Capital of TX Highway North
Building 1
Suite 150
Austin, TX 78759
Tel: 512.388.7294
Fax: 512.388.4171

Plano
500 North Central Expressway
Suite 440
Plano, TX 75074
♦ Tel: 972.244.5000
Fax: 972.244.5001

Houston
20405 State Highway 249
Suite 450
Houston, TX 77070
Tel: 281.379.7800
Fax: 281.379.7818

Canada
Ontario
Ottawa
260 Hearst Way
Suite 400
Kanata, ON K2L 3H1
♦ Tel: 613.592.1263
Fax: 613.592.3253

INTERNATIONAL

France
Paris
LSI Logic S.A.
Immeuble Europa
53 bis Avenue de l'Europe
B.P. 139
78148 Velizy-Villacoublay
Cedex, Paris
♦ Tel: 33.1.34.63.13.13
Fax: 33.1.34.63.13.19

Germany
Munich
LSI Logic GmbH
Orleansstrasse 4
81669 Munich
♦ Tel: 49.89.4.58.33.0
Fax: 49.89.4.58.33.108

Stuttgart
Mittlerer Pfad 4
D-70499 Stuttgart
♦ Tel: 49.711.13.96.90
Fax: 49.711.86.61.428

Italy
Milan
LSI Logic S.P.A.
Centro Direzionale Colleoni Palazzo
Orione Ingresso 1
20041 Agrate Brianza, Milano
♦ Tel: 39.039.687371
Fax: 39.039.6057867

Japan
Tokyo
LSI Logic K.K.
Rivage-Shinagawa Bldg. 14F
4-1-8 Kounan
Minato-ku, Tokyo 108-0075
♦ Tel: 81.3.5463.7821
Fax: 81.3.5463.7820

Osaka
Crystal Tower 14F
1-2-27 Shiromi
Chuo-ku, Osaka 540-6014
♦ Tel: 81.6.947.5281
Fax: 81.6.947.5287

Sales Offices and Design Resource Centers (Continued)

Korea
Seoul
LSI Logic Corporation of Korea Ltd
10th Fl., Haesung 1 Bldg.
942, Daechi-dong,
Kangnam-ku, Seoul, 135-283
Tel: 82.2.528.3400
Fax: 82.2.528.2250

The Netherlands
Eindhoven
LSI Logic Europe Ltd
World Trade Center Eindhoven
Building 'Rijder'
Bogert 26
5612 LZ Eindhoven
Tel: 31.40.265.3580
Fax: 31.40.296.2109

Singapore
Singapore
LSI Logic Pte Ltd
7 Temasek Boulevard
#28-02 Suntec Tower One
Singapore 038987
Tel: 65.334.9061
Fax: 65.334.4749

Sweden
Stockholm
LSI Logic AB
Finlandsgatan 14
164 74 Kista
♦ Tel: 46.8.444.15.00
Fax: 46.8.750.66.47

Taiwan
Taipei
LSI Logic Asia, Inc.
Taiwan Branch
10/F 156 Min Sheng E. Road
Section 3
Taipei, Taiwan R.O.C.
Tel: 886.2.2718.7828
Fax: 886.2.2718.8869

United Kingdom
Bracknell
LSI Logic Europe Ltd
Greenwood House
London Road
Bracknell, Berkshire RG12 2UB
♦ Tel: 44.1344.426544
Fax: 44.1344.481039

♦ Sales Offices with Design Resource Centers

International Distributors

Australia
New South Wales
Reptechnic Pty Ltd
3/36 Bydown Street
Neutral Bay, NSW 2089
♦ Tel: 612.9953.9844
Fax: 612.9953.9683

Belgium
Acal nv/sa
Lozenberg 4
1932 Zaventem
Tel: 32.2.7205983
Fax: 32.2.7251014

China
Beijing
LSI Logic International Services Inc.
Beijing Representative Office
Room 708
Canway Building
66 Nan Li Shi Lu
Xicheng District
Beijing 100045, China
Tel: 86.10.6804.2534 to 38
Fax: 86.10.6804.2521

France
Rungis Cedex
Azzurri Technology France
22 Rue Saarinen
Sillic 274
94578 Rungis Cedex
Tel: 33.1.41806310
Fax: 33.1.41730340

Germany
Haar
EBV Elektronik
Hans-Pinsel Str. 4
D-85540 Haar
Tel: 49.89.4600980
Fax: 49.89.46009840

Munich
Avnet Emg GmbH
Stahlgruberring 12
81829 Munich
Tel: 49.89.45110102
Fax: 49.89.42.27.75

Wuennenberg-Haaren
Peacock AG
Graf-Zepplin-Str 14
D-33181 Wuennenberg-Haaren
Tel: 49.2957.79.1692
Fax: 49.2957.79.9341

Hong Kong
Hong Kong
AVT Industrial Ltd
Unit 608 Tower 1
Cheung Sha Wan Plaza
833 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: 852.2428.0008
Fax: 852.2401.2105

Serial System (HK) Ltd
2301 Nanyang Plaza
57 Hung To Road, Kwun Tong
Kowloon, Hong Kong
Tel: 852.2995.7538
Fax: 852.2950.0386

India
Bangalore
Spike Technologies India Private Ltd
951, Vijayalakshmi Complex,
2nd Floor, 24th Main,
J P Nagar II Phase,
Bangalore, India 560078
♦ Tel: 91.80.664.5530
Fax: 91.80.664.9748

Israel
Tel Aviv
Eastronics Ltd
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61392
Tel: 972.3.6458777
Fax: 972.3.6458666

Japan
Tokyo
Daito Electron
Sogo Kojimachi No.3 Bldg
1-6 Kojimachi
Chiyoda-ku, Tokyo 102-8730
Tel: 81.3.3264.0326
Fax: 81.3.3261.3984

Global Electronics Corporation
Nichibei Time24 Bldg. 35 Tansu-cho
Shinjuku-ku, Tokyo 162-0833
Tel: 81.3.3260.1411
Fax: 81.3.3260.7100
Technical Center
Tel: 81.471.43.8200

Marubeni Solutions
1-26-20 Higashi
Shibuya-ku, Tokyo 150-0001
Tel: 81.3.5778.8662
Fax: 81.3.5778.8669

Shinki Electronics
Myuru Daikanyama 3F
3-7-3 Ebisu Minami
Shibuya-ku, Tokyo 150-0022
Tel: 81.3.3760.3110
Fax: 81.3.3760.3101

Yokohama-City
Innotech
2-15-10 Shin Yokohama
Kohoku-ku
Yokohama-City, 222-8580
Tel: 81.45.474.9037
Fax: 81.45.474.9065

Macnica Corporation
Hakusan High-Tech Park
1-22-2 Hadusan, Midori-Ku,
Yokohama-City, 226-8505
Tel: 81.45.939.6140
Fax: 81.45.939.6141

The Netherlands
Eindhoven
Acal Nederland b.v.
Beatrix de Rijkweg 8
5657 EG Eindhoven
Tel: 31.40.2.502602
Fax: 31.40.2.510255

Switzerland
Brugg
LSI Logic Sulzer AG
Mattenstrasse 6a
CH 2555 Brugg
Tel: 41.32.3743232
Fax: 41.32.3743233

Taiwan
Taipei
Avnet-Mercuries Corporation, Ltd
14F, No. 145,
Sec. 2, Chien Kuo N. Road
Taipei, Taiwan, R.O.C.
Tel: 886.2.2516.7303
Fax: 886.2.2505.7391

Lumax International Corporation, Ltd
7th Fl., 52, Sec. 3
Nan-Kang Road
Taipei, Taiwan, R.O.C.
Tel: 886.2.2788.3656
Fax: 886.2.2788.3568

Prospect Technology Corporation, Ltd
4Fl., No. 34, Chu Luen Street
Taipei, Taiwan, R.O.C.
Tel: 886.2.2721.9533
Fax: 886.2.2773.3756

Wintech Microeletronics Co., Ltd
7F., No. 34, Sec. 3, Pateh Road
Taipei, Taiwan, R.O.C.
Tel: 886.2.2579.5858
Fax: 886.2.2570.3123

United Kingdom
Maidenhead
Azzurri Technology Ltd
16 Grove Park Business Estate
Waltham Road
White Waltham
Maidenhead, Berkshire SL6 3LW
Tel: 44.1628.826826
Fax: 44.1628.829730

Milton Keynes
Ingram Micro (UK) Ltd
Garamonde Drive
Wymbush
Milton Keynes
Buckinghamshire MK8 8DF
Tel: 44.1908.260422

Swindon
EBV Elektronik
12 Interface Business Park
Bincknoll Lane
Wootton Bassett,
Swindon, Wiltshire SN4 8SY
Tel: 44.1793.849933
Fax: 44.1793.859555

♦ Sales Offices with Design Resource Centers