CoreMP7

DirectCore

Product Summary

- Personal Audio (MP3, WMA, and AAC Players)
- Personal Digital Assistants
- Wireless Handset
- Pagers
- Digital Still Camera
- Inkjet/Bubble-Jet Printer
- Monitors

Key Features

- FPGA Optimized ARM7™ Family Processor
- Compatible with ARM7TDMI-S™
- 32/16-Bit RISC Architecture (ARMv4T)
- 32-Bit ARM® Instruction Set
- 16-Bit Thumb® Instruction Set
- 32-Bit Unified Bus Interface
- 3-Stage Pipeline
- 32-Bit ALU
- 32-Bit Memory Addressing Range
- Static Operation
- EmbeddedICE-RT™ Real-Time Debug Unit
- JTAG Interface Unit

Benefits

- Fully Implemented in FPGA Fabric
- All Microprocessor I/Os Available to User
- Unified Bus Interface Simplifies SoC Design
- ARM and Thumb Instruction Sets Can Be Mixed

ARM Supported Families

- ProASIC3 (M7A3P)
- ProASIC3E (M7A3PE)
- Fusion (M7AFS)

Synthesis and Simulation Support

- Directly Supported within the Actel Libero® Integrated Design Environment (IDE)
- Synthesis: Synplify® and Design Compiler®
- Simulation: Vital-Compliant VHDL Simulators and OVI-Compliant Verilog Simulators

Verification and Compliance

- Compliant with ARMv4T ISA
- Compatible with ARM7TDMI-S

Core Version

- This Datasheet Defines the Functionality for CoreMP7 v1.0.

Contents

Introduction

The CoreMP7 soft IP core is an ARM7 family processor optimized for use in Actel ARM-ready FPGAs and is compatible with the ARM7TDMI-S. Users should refer to the *ARM7TDM-S Technical Reference Manual* (DDI0234A-7TMIS-R4.pdf), published by the ARM Corporation, for detailed information on the ARM7. The ARM7 TRM is available for download from the ARM website at www.arm.com.

The CoreMP7 is supplied with an Advanced Microcontroller Bus Architecture (AMBA) Advanced High-Performance Bus (AHB) compliant wrapper for inclusion in an AMBA-based processor system such as the one generated by the Actel CoreConsole IP Deployment Platform (IDP).

ARM7 Family Processor

The CoreMP7 is a general purpose, 32-bit, ARM7 family microprocessor that offers high performance and low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The simplicity of RISC results in a high instruction throughput and fast real-time interrupt response from a small and cost-effective processor core. Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory. The CoreMP7 processor also implements the Thumb instruction set, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue.

The 16-bit Thumb instruction set approaches twice the density of standard ARM code while retaining most of the ARM performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because Thumb code operates on the same 32-bit register set as ARM code. Thumb code is able to reduce up to 65% of the code size compared to 32-bit ARM instructions, and offers 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system.

Device Utilization

The CoreMP7 has been implemented in M7 ProASIC3/E and M7 Fusion devices. A summary of the implementation data is listed in Table 1.

CoreMP7S

This variant of the CoreMP7 is optimized for maximum speed and minimum size and does not include debug.

CoreMP7Sd

This variant of the CoreMP7 is optimized for minimum size and includes debug.

Table 1 • **CoreMP7 Utilization and Performance**

Device Variant	Performance (MHz)	Tiles	RAM Block	Utilization (%)
M7A3P1000				
CoreMP7S	28.548	6397	4	26.0
CoreMP7Sd	22.714	8522	4	34.7
M7A3PE600				
CoreMP7S	29.699	6324	4	45.7
CoreMP7Sd	23.646	8587	4	62.1
M7A3P250				
CoreMP7S	23.904	6104	4	99.3
M7AFS600				
CoreMP7S				
CoreMP7Sd				
M7AFS1500				
CoreMP7S				
CoreMP7Sd				

General Description

The CoreMP7 processor architecture, core, and functional diagrams are illustrated in the following figures:

- The CoreMP7 block diagram is shown in Figure 1.
- The CoreMP7 core is shown in Figure 2 on page 4.
- The CoreMP7 functional diagram is shown in Figure 3 on page 5.

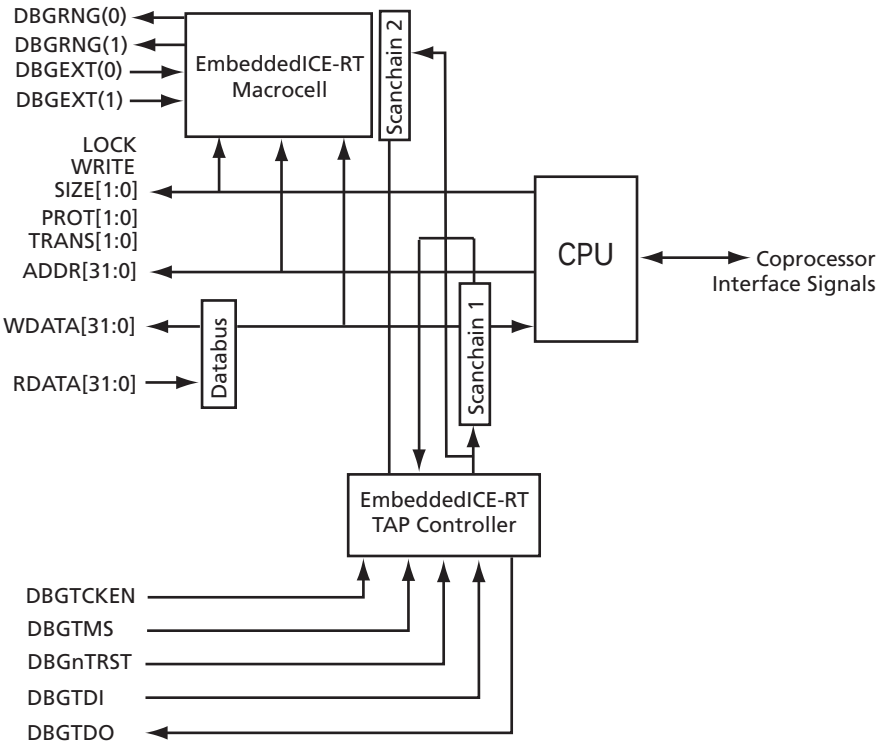

Figure 1 • **CoreMP7 Top-Level Block Diagram**

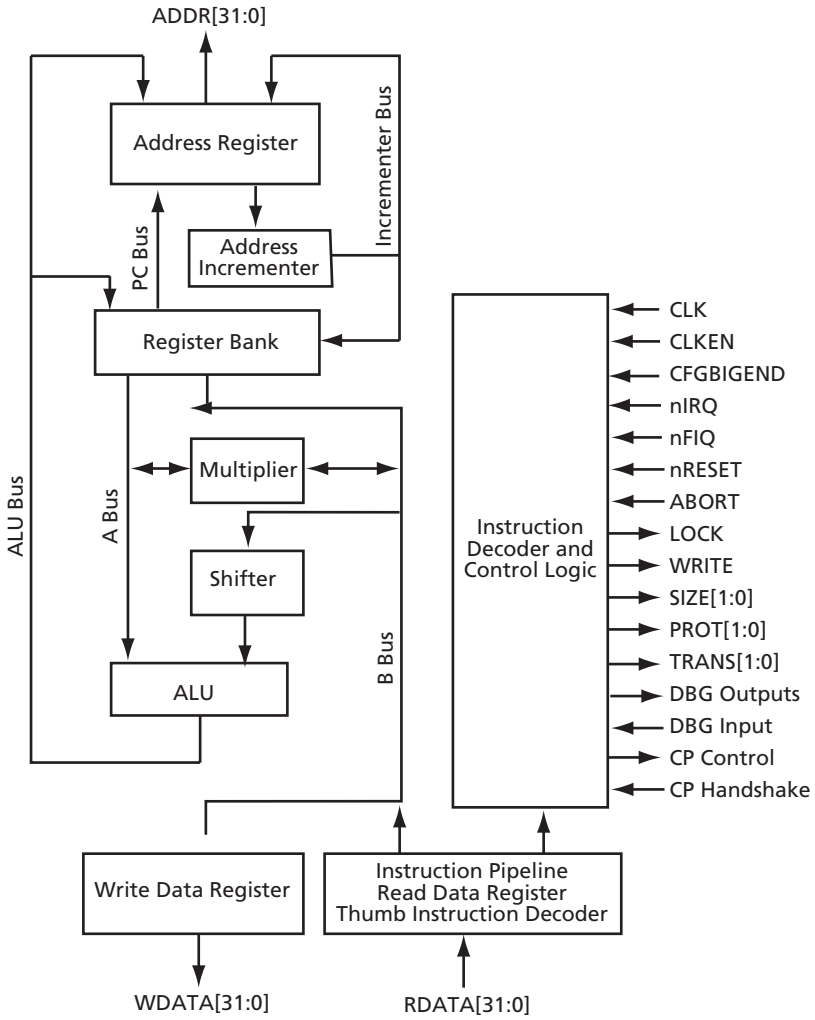

Figure 2 • **CoreMP7 CPU Block Diagram**

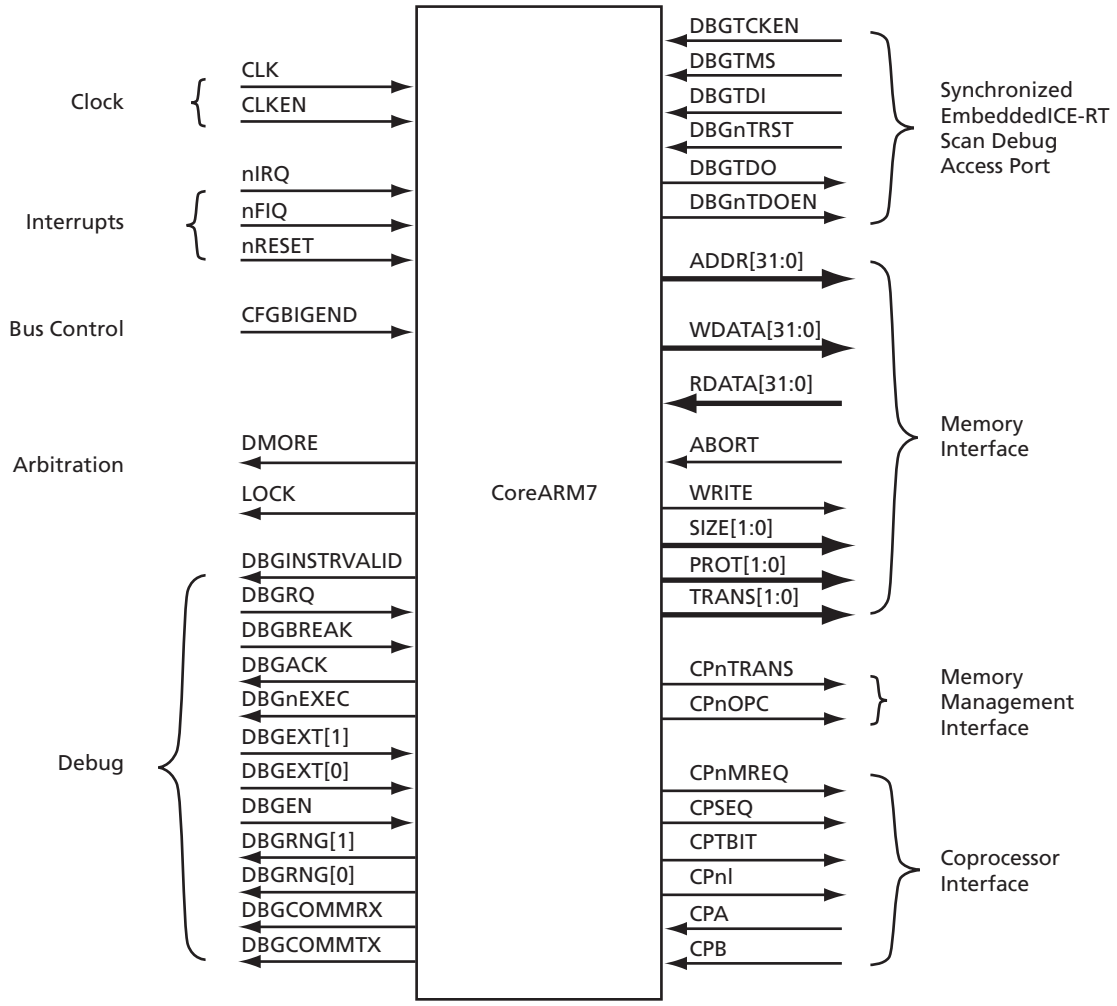

Figure 3 • **CoreMP7 Functional Diagram**

The signals of the CoreMP7 are listed in Table 2.

Table 2 • **Signal Descriptions**

Name	Type	Description
ABORT	Input	Memory abort or bus error
CFGBIGEND	Input	Big/Little Endian configuration
CLK	Input	Clock
CLKEN	Input	Clock enable
CPA	Input	Coprocessor absent
CPB	Input	Coprocessor busy
DBGBREAK	Input	EICE breakpoint/watchband indicator
DBGEN	Input	Debug enable
DBGEXT[1:0]	Input	EICE external input 0

Note: The CoreMP7 is available with either the native ARM7 bus interface or with an AHB wrapper. The use of the AHB wrapper changes or transforms some of the signals in Table 2. This is discussed in detail later in this document.

Table 2 • **Signal Descriptions (Continued)**

Name	Type	Description
DBGnTRST	Input	Test reset
DBGRQ	Input	Debug request
DBGTCKEN	Input	Test clock enable
DBGTDI	Input	EICE data in
DBGTMS	Input	EICE mode select
nFIQ	Input	Interrupt request
nIRQ	Input	Fast interrupt request
nRESET	Input	Reset
RDATA[31:0]	Input	Read data bus
ADDR[31:0]	Output	Address bus
CPnI	Output	Coprocessor instruction (asserted low)
CPnMREQ	Output	Memory request (asserted low)
CPnOPC	Output	Opcode fetch (asserted low)
CPnTRANS	Output	Memory translate (asserted low)
CPSEQ	Output	Sequential address
CPTBIT	Output	Processor in Thumb mode
DBGACK	Output	Debug acknowledge
DBGCOMMRX	Output	EICE communication channel receive
DBGCOMMTX	Output	EICE communication channel transmit
DBGnEXEC	Output	Executed (asserted low)
DBGnTDOEN	Output	TDO enable (asserted low)
DBGRNG[1:0]	Output	EICE rangeout
DBGTDO	Output	EICE data out
DBGINSTRVALID	Output	ETM Instruction valid indicator
DMORE	Output	Set when next data memory access is followed by a sequential data memory access
LOCK	Output	Locked transaction operation
PROT[1:0]	Output	Indicates code, data, or privilege level
SIZE[1:0]	Output	Memory access width
TRANS	Output	Next transaction type (i, n, s)
WDATA[31:0]	Output	Write data bus
WRITE	Output	Indicates write access

Note: The CoreMP7 is available with either the native ARM7 bus interface or with an AHB wrapper. The use of the AHB wrapper changes or transforms some of the signals in Table 2. This is discussed in detail later in this document.

Programmer's Model

This section summarizes the programmer's model of the CoreMP7. Supporting detail is available in the ARM *ARM7TDMI-S Technical Reference Manual* (available for download at www.arm.com) and the *ARM Architecture Reference Manual,* which can be purchased at www.amazon.com.

The CoreMP7 processor implements the ARMv4T architecture and includes both the 32-bit ARM instruction set and the 16-bit Thumb instruction set.

Processor Operating States

The CoreMP7 processor has two operating states:

ARM state: 32-bit, word-aligned ARM instructions are executed in this state.

Thumb state: 16-bit, halfword-aligned Thumb instructions are executed in this state.

In Thumb state, the Program Counter (PC) uses bit 1 to select between alternate halfwords.

Note: Transition between ARM and Thumb states does not affect the processor mode or the register contents.

Switching State

You can switch the operating state of the CoreMP7 between ARM state and Thumb state using the BX instruction. This is described fully in the *ARM Architecture Reference Manual*.

All exception handling is performed in ARM state. If an exception occurs in Thumb state, the processor reverts to ARM state. The transition back to Thumb state occurs automatically on return.

Memory Formats

The CoreMP7 processor views memory as a linear collection of bytes, numbered in ascending order from zero:

- Bytes 0 to 3 hold the first stored word.
- Bytes 4 to 7 hold the second stored word.
- Bytes 8 to 11 hold the third stored word.

Although both Little Endian and Big Endian memory formats are supported, it is recommended that you use Little Endian format.

Data Types

The CoreMP7 processor supports the following data types:

- Word (32-bit)
- Halfword (16-bit)
- Byte (8-bit)

You must align these as follows:

- Word quantities must be aligned to four-byte boundaries.
- Halfword quantities must be aligned to two-byte boundaries.
- Byte quantities can be placed on any byte boundary.

Operating Modes

The CoreMP7 processor has seven operating modes:

- User mode is the usual ARM program execution state, and is used for executing most application programs.
- Fast interrupt (FIQ) mode supports a data transfer or channel process.
- Interrupt (IRQ) mode is used for general-purpose interrupt handling.
- Supervisor mode is a protected mode for the operating system.
- Abort mode is entered after a data or instruction prefetch abort.
- System mode is a privileged user mode for the operating system.
- Undefined mode is entered when an undefined instruction is executed.

Modes other than User mode are collectively known as privileged modes. Privileged modes are used to service interrupts or exceptions, or to access protected resources.

Registers

The CoreMP7 processor has a total of 37 registers:

- 31 general-purpose 32-bit registers
- 6 status registers

These registers are not all accessible at the same time. The processor state and operating mode determine which registers are available to the programmer.

The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are accessible at any one time. In privileged modes, mode-specific banked registers become available. Figure 4 on page 8 shows which registers are available in each mode.

The ARM state register set contains 16 directly accessible registers, r0 to r15. An additional register, the Current Program Status Register (CPSR), contains condition code flags, and the current mode bits. Registers r0 to r13 are general-purpose registers used to hold either data or address values. Registers r14 and r15 have special functions as the Link Register and Program Counter.

Link Register

Register 14 is used as the subroutine Link Register (LR).

Register 14 (r14) receives a copy of r15 when a Branch with Link (BL) instruction is executed.

At all other times, you can treat r14 as a general-purpose register.

The corresponding banked registers—r14_svc, r14_irq, r14_fiq, r14_abt, and r14_und—are similarly used to hold the return values of r15 when interrupts and exceptions arise, or when BL instructions are executed within interrupt or exception routines.

Program Counter

Register 15 holds the Program Counter (PC).

In ARM state, bits [1:0] of r15 are zero. Bits [31:2] contain the PC.

In Thumb state, bit [0] is zero. Bits [31:1] contain the PC.

In privileged modes, another register, the Saved Program Status Register (SPSR), is accessible. This contains the condition code flags, and the mode bits saved as a result of the exception that caused entry to the current mode.

Figure 4 shows the ARM state registers.

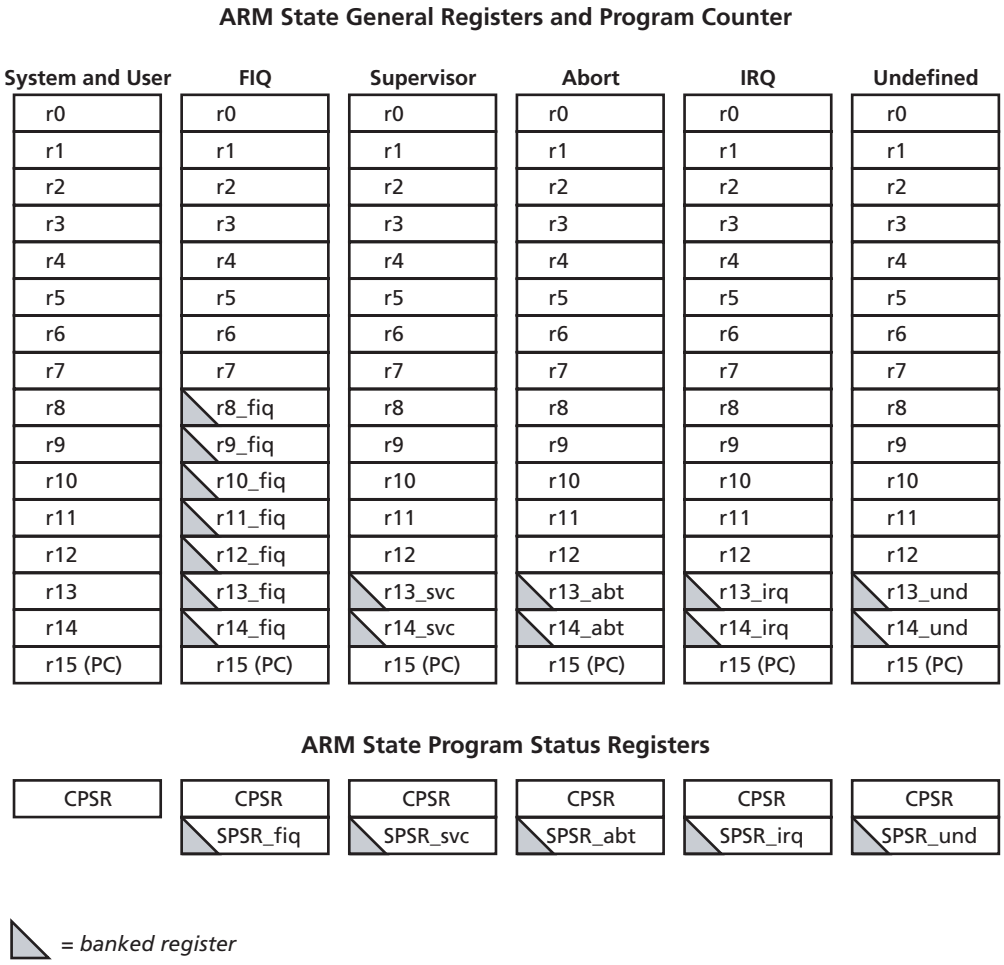

Figure 4 • **CoreMP7 Register Organization in the ARM State**

The Thumb State Register Set

The Thumb state register set is a subset of the ARM state set. The programmer has direct access to the following:

- Eight general registers, r0–r7
- The PC
- A Stack Pointer (SP)
- A Link Register (LR)
- The CPSR

There are banked SPs, LRs, and SPSRs for each privileged mode. The Thumb state register set is shown in Figure 5.

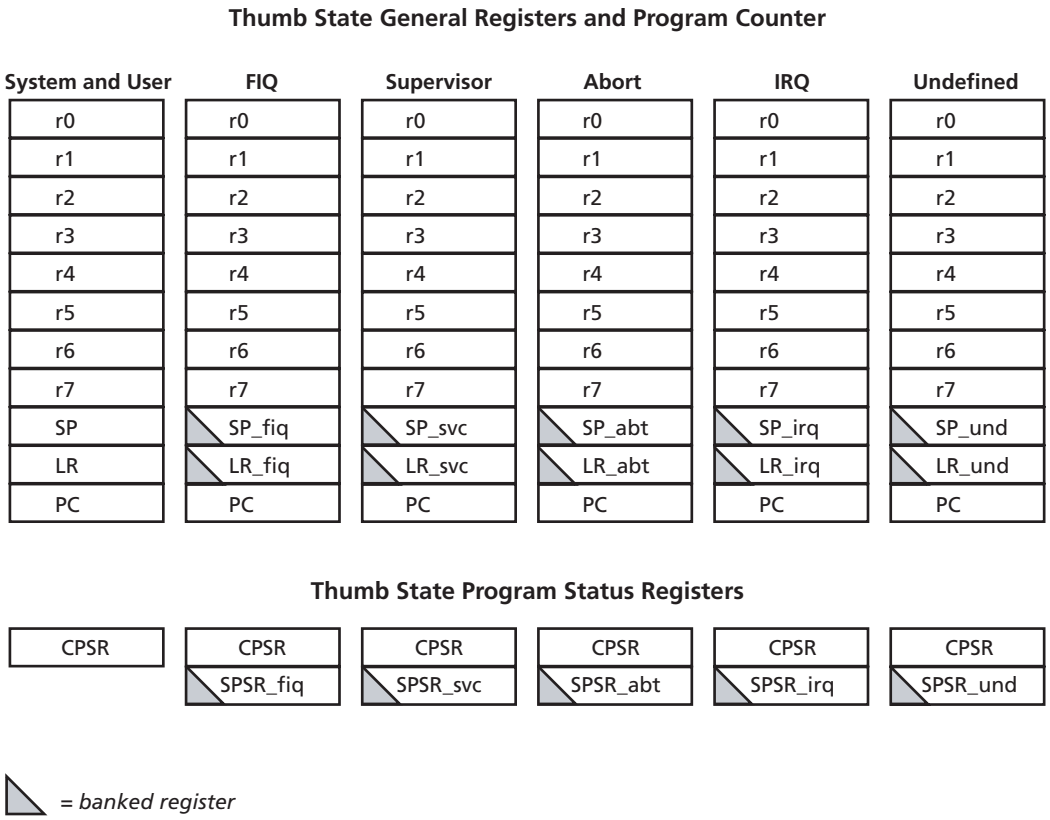

Figure 5 • **CoreMP7 Thumb State Registers**

The Relationship Between ARM State and Thumb State Registers

The Thumb state registers relate to the ARM state registers in the following way:

- Thumb state r0–r7 and ARM state r0–r7 are identical.
- Thumb state CPSR and SPSR, and ARM state CPSR and SPSR are identical.
- Thumb state SP maps onto ARM state r13.
- Thumb state LR maps onto ARM state r14.
- The Thumb state PC maps onto the ARM state PC (r15).

These relationships are shown in Figure 6.

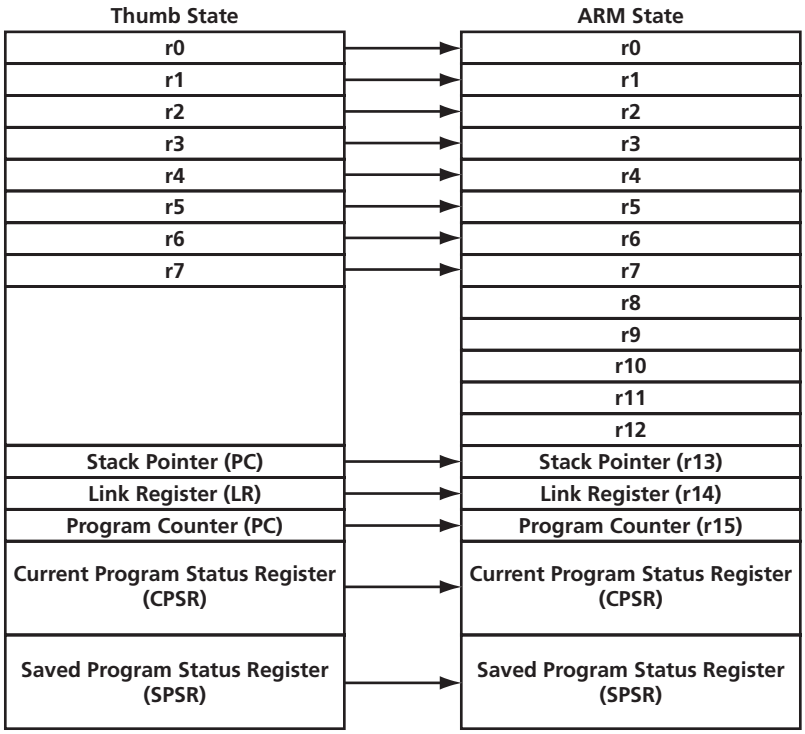

Figure 6 • **Mapping of Thumb State Registers to ARM State Registers**

Note: Registers r0–r7 are known as the low registers. Registers r8–r15 are known as the high registers.

The Program Status Registers

The CoreMP7 core contains a CPSR and five SPSRs for exception handlers to use. The program status registers the following:

- Hold the condition code flags
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 7.

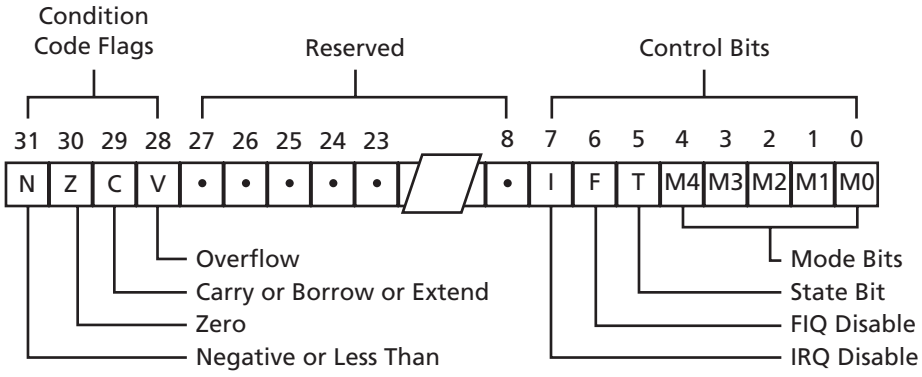

Figure 7 • **Program Status Register Format**

The Condition Code Flags

The N, Z, C, and V bits are the condition code flags. You can set these bits by arithmetic and logical operations. The flags can also be set by MSR and LDM instructions. The CoreMP7 processor tests these flags to determine whether to execute an instruction.

All instructions can be executed conditionally in ARM state. In Thumb state, only the Branch instruction can be executed conditionally. For more information about conditional execution, see the *ARM Architecture Reference Manual*.

AHB Wrapper

The AHB wrapper interfaces between the CoreMP7 native ARM7 interface and the AHB bus. The module translates access from the core to AHB accesses when the core is the current master. The external interface signals from the wrapper are described in Table 3.

Table 3 • **AHB Wrapper External Interface**

Signal External Master I/F	**Direction**	**Description**
HCLK	Input	Bus clock. This clock times all bus transfers. All signal timings are related to the rising edge of **HCLK**.
HRESETn	Input	Reset. The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW AHB signal.
HREADY	Input	Transfer done. When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer.
HRESP[1:0]	Input	Transfer response. Indicates an OKAY, ERROR, RETRY, or SPLIT response.
HGRANT	Input	Bus grant. Indicates that the CoreMP7 is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when **HREADY** is HIGH, so a master gains access to the bus when both **HREADY** and **HGRANT** are HIGH.
HADDR[31:0]	Output	This is the 32-bit system address bus.
HTRANS[1:0]	Output	Transfer type. Indicates the type of the current transfer.
HWRITE	Output	Transfer direction. When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0]	Output	Transfer size. Indicates the size of the transfer, which can be byte (8-bit), halfword (16-bit), or word (32-bit).
HBURST[2:0]	Output	Burst type. Indicates if the transfer forms part of a burst. The CoreMP7 performs incrementing bursts of type INCR.
HPROT[3:0]	Output	Protection control. These signals indicate if the transfer is an opcode fetch or data access, and if the transfer is a Supervisor mode access or User mode access.
HWDATA[31:0]	Output	32-bit data from the MASTER.
HRDATA[31:0]	Input	32-bit data written back to the MASTER.

CoreMP7 Variants

There are two implementations of CoreMP7 for all M7 devices except for the M7A3P250 which only supports the CoreMP7S variant. The utilization and performance of the variants are shown in Table 1 on page 2.

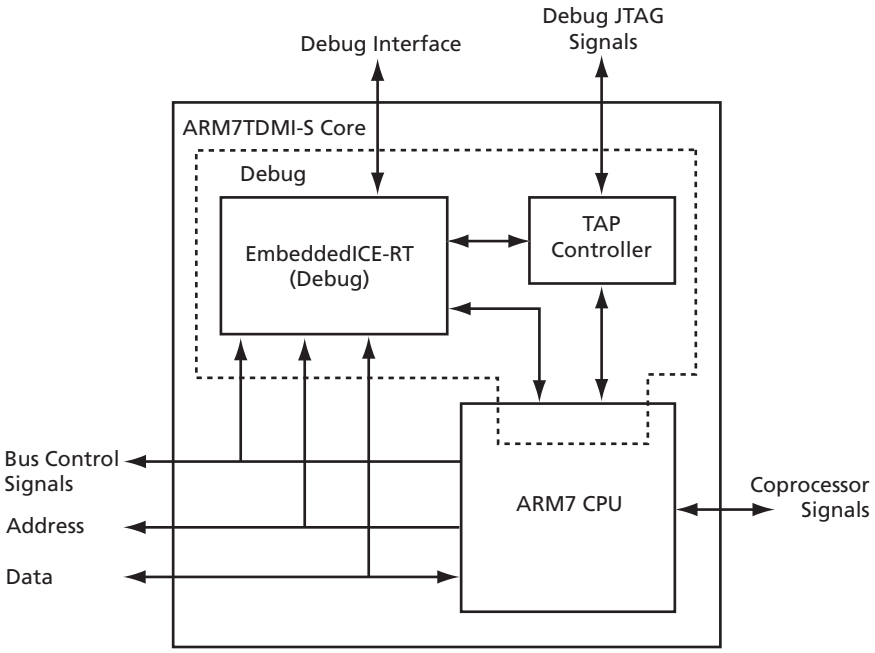

Figure 8 • **ARM7 TDMI-S Core**

CoreMP7Sd

The CoreMP7Sd is configured with all features of the ARM7TDMI-S. The variant incorporates the full debug functionality of the ARM7TDMI-S and is fully compliant with RealView RVDS, RVDK, and other ARM software debug tools.

CoreMP7S

The CoreMP7S is optimized for maximum speed and minimum area, and has the same features as CoreMP7Sd except that it does not include the ICE-RT debug block, the TAP controller, or the coprocessor interface and is Little Endian only, to reduce the size of the core. This means that the standard debug tools cannot be used with this variant of CoreMP7.

No Debug

This is the most obvious characteristic of the CoreMP7S variant. To reduce area, the Debug EmbeddedICE-RT macrocell, the EmbeddedICE-RT TAP controller, and the scan logic have been optimized out. This means that standard software debug tools cannot be used when these variants of the CoreMP7 are instantiated. Users of these variants can use the comparable variant that includes debug for development, and when the application has been fully tested and debugged, one of these variants can be instantiated to reduce area in the final shipping product.

No Coprocessor Interface

The coprocessor interface is a rarely used feature in ARM7 family microprocessors and has been removed from the CoreMP7S variant to minimize area.

Little Endian Only

Most microprocessor-based systems use Little Endian byte ordering. The option of selecting Big Endian has been removed from the CoreMP7S variant to minimize area.

On-Chip RAM Consumed by Register Block

To minimize the area, the CoreMP7 variants map the processor register block into on-chip RAM. RAM blocks used to implement CoreMP7 registers are no longer available for use in user designs.

Delivery and Deployment

The CoreMP7 is delivered as a series of files from the Actel CoreConsole IP Deployment Platform (IDP) development tool, and these files are directly imported into the Design and Simulation folders for use in the Actel Libero IDE FPGA tool suite. The CoreMP7 files consist of the BFM files and test wrapper, AHB wrapper, and the A7S secured CDB file, which is instantiated on the user device at programming.

Bus Functional Model (BFM)

During the development of an FPGA-based SoC, you can use a number of stages of testing. This may involve some, or all, of the following approaches:

- Hardware simulation, using Verilog or VHDL
- Software simulation, using a host-based instruction set simulator (ISS) of the SoC's processor
- Hardware and software co-verification, using a fully functional model of the processor in Verilog, VHDL, or SWIFT form, or using a tool such as Seamless

Generally, due to the rapid prototyping capability of FPGAs, integration of hardware and software often occurs earlier in the SoC development cycle for FPGA targets than it would for ASIC targets. Therefore, hardware and software co-verification, which can be very slow, is usually not a critical issue for all but the most complex FPGA-based SoCs.

A software simulation solution is available to users as part of the CoreMP7 RealView Developer Kit (RVDK). The RealView Instruction Set Simulator, included in the RVDK, provides CoreMP7 instruction accurate simulation, as well as powerful features, such as integration with the RealView debugger (also in the RVDK).

The CoreConsole IPD SoC configuration utility provides a means for the developer to stitch together IP blocks using the AHB bus fabric. It generates a system testbench, controlled by a script-driven, bus functional model (BFM) of the CoreMP7 processor. The BFM allows the developer to model low-level bus transactions, which allow verification of connectivity of the various IP blocks, as well as of the system memory map presented to the CoreMP7 by the rest of the hardware.

This section describes the following aspects of the CoreMP7 BFM:

- Functionality
- BFM usage flow
- BFM script language
- Platforms
- Supported simulation tools
- Example BFM use case

BFM Usage Flow

The BFM is part of an overall system test strategy, so it is helpful to look at the context in which it is used.

Figure 9 shows the various components within an example system-level testbench.

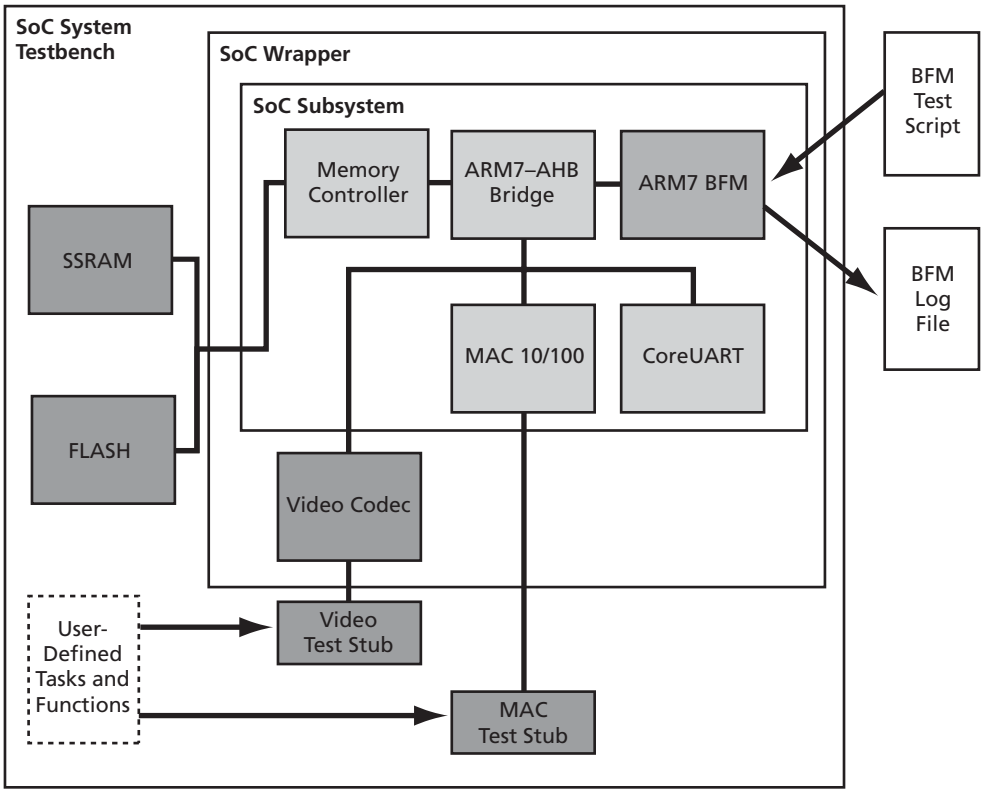

Figure 9 • **SoC System-Level Testbench Example**

In Figure 9, it is assumed that the developer specifies the SoC subsystem, by means of selecting the processor, bus fabric, IP blocks, and the memory subsystem in CoreConsole. In this example, the user selects:

- CoreMP7 processor
- AMBA AHB bus fabric
- MAC 10/100 IP core
- CoreUART IP core
- External SSRAM and Flash memory

The user also specifies the memory map of the system. Based on this information, CoreConsole generates the following outputs, amongst others:

- Verilog/VHDL model of SoC subsystem
- Verilog/VHDL models of IP cores
- CoreMP7 BFM
- BFM test script
- System-level skeleton testbench

The BFM acts as a pin-for-pin replacement of the CoreMP7 in the SoC subsystem. It initiates bus transactions on the native CoreMP7 bus, which are cycle-accurate with real bus cycles that the CoreMP7 would produce. It has no knowledge, however, of real CoreMP7 instructions.

The BFM may be used to run a basic test of the SoC subsystem, using the skeleton system testbench. CoreConsole uses known attributes (from SPIRIT XML descriptions) of any registers or addressable locations within the IP cores, together with the user-defined memory map, to generate a basic BFM script. This script does a write to and/or read from all accessible locations. It has knowledge of whether registers are read-only, read/write, clear-on-read, or write-only. From this it can decide what expected data should be on reads.

You can edit the SoC Verilog/VHDL to add new design blocks, such as the Video Codec in Figure 9. The system-level testbench may also be filled out to include tasks that test any newly added functionality, or add stubs to allow more complex system testing involving the IP cores. The BFM input scripts can be manually enhanced, so that you can test access to register locations in newly added logic. In this way, you can provide stimuli to the system from the inside (via the CoreMP7 BFM), as well as from the outside (via testbench tasks).

The design flow into which the BFM fits is shown in Figure 10.

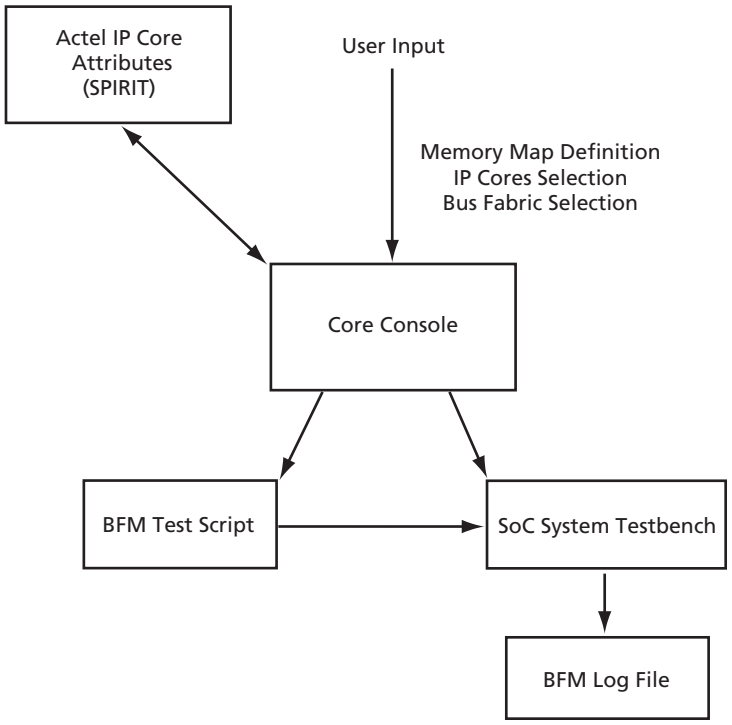

Figure 10 • **BFM Flow Diagram**

Functionality

This section describes the specific functionality of the CoreMP7 BFM. The BFM models the CoreMP7 native bus. This includes the following bus signals:

- ADDR, // address bus
- WDATA, // write data bus
- RDATA, // read data bus
- TRANS, // next transaction type (i, n, s)
- WRITE, // indicates write access
- CLKEN, // clock enable

The BFM also models the following control signals:

- CFGBIGEND, // Big/Little Endian configuration
- CLK, // clock
- nFIQ, // interrupt request
- nIRQ, // fast interrupt request
- SIZE, // memory access width

CoreMP7 Pin Compatibility

The BFM model is pin-for-pin compatible with the CoreMP7. This allows the model to be dropped into the space that would be occupied by the core in the system testbench.

CoreMP7 Bus Cycle Accuracy

The bus cycle timings for the CoreMP7 native bus signals are specified in the *ARM7TDMI-S Technical Reference Manual*. The CoreMP7 BFM models these bus cycles exactly.

Scripting

In order to provide a simple and extensible mechanism for providing stimuli to the BFM, a BFM scripting language is defined in the "BFM Script Language" section on page 17. This has the ability to initiate writes to system resources, reads from system resources (with or without checking of expected data), and to wait for interrupt events.

Self-Checking

The BFM gives a pass/fail indication at the end of a test run. This is based on whether any of the expected data read checks failed or not.

Endianess

The BFM supports both Big and Little Endian memory configurations. For byte and halfword transfers, it reads and writes data from/to the appropriate data lanes.

Interrupt Support

The BFM has the ability to wait for either of the two CoreMP7 interrupt lines to be triggered before proceeding with the remainder of the test script.

Log File Generation

The BFM generates output messages to the console of the simulation tool and also generates a HTML log file. The messages in this file are color-coded, so that any errors may be easily identified.

BFM Script Language

The following script commands are defined for use by the BFM.

write

This command causes the BFM to perform a write to a specified offset, within the memory map range of a specified system resource.

Syntax

```
write width resource_name byte_offset data;
```

width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte respectively.

resource_name

This is a string containing the user-friendly instance name of the resource being accessed, as defined by the user in the memory map (when input to CoreConsole).

byte_offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

data

This is the data to be written. It is specified as a hexadecimal value.

Example

```
write W videoCodec 20 11223344;
```

read

This command causes the BFM to perform a read of a specified offset, within the memory map range of a specified system resource.

Syntax

```
read width resource_name byte_offset;
```

width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte respectively.

resource_name

This is a string containing the user-friendly instance name of the resource being accessed, as defined by the user in the memory map (when input to CoreConsole).

byte_offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

Example

```
read W videoCodec 20;
```

readcheck

This command causes the BFM to perform a read of a specified offset, within the memory map range of a specified system resource, and to compare the read value with the expected value provided.

Syntax

```
readcheck width resource_name byte_offset
data;
```

width

This takes on the enumerated values of W, H, or B, for word, halfword, or byte respectively.

resource_name

This is a string containing the user-friendly instance name of the resource being accessed, as defined by the user in the memory map (when input to CoreConsole).

byte_offset

This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

data

This is the expected read data. It is specified as a hexadecimal value.

Example

```
readcheck W videoCodec 20 11223344;
```

Timing Shell

There is a timing shell provided for each CoreMP7 variant wrapped around the BFM itself. Therefore the BFM is bus cycle accurate, and it performs setup/hold checks to model output propagation delays.

Example BFM Use Case

This section goes through an example use case of the CoreMP7 BFM. The example SoC to be used in this section is the same as that shown in Figure 10 on page 16. In this system, the developer requires two Actel IP cores, namely the MAC 10/100 and the CoreUART.

SPIRIT Attributes

CoreConsole has access to a database of Actel IP cores and a list of attributes for each core. These attributes are organized according to the SPIRIT specification, in XML. For example, in the case of the CoreUART, the attributes would indicate that there are three registers, as shown in Table 4 on page 18.

Table 4 • **Attributes of Three Registers**

Offset	Register	Read/Write	Width
0	UART Status Register	R	Byte
1	UART Tx Data	W	Byte
2	UART Rx Data	R	Byte

Based on these attributes, CoreConsole can determine when generating the BFM script that there are three locations corresponding to the UART, which may be accessed. In this case, none of the registers are R/W, so there will not be any self-checking that can be performed for the UART. Nevertheless, the bus transactions do take place and the cycles may be viewed in a waveform of the simulator.

Memory Map

You must feed in the memory map of the SoC to CoreConsole. During this stage, the absolute address ranges of the various system resources in the CoreMP7 memory map are fed in. Also, user-friendly instance names of these resources are fed in.

For example, you can feed the memory map information in Table 5 into CoreConsole.

Table 5 • **Memory Map Information**

Resource	Actel IP Core	Address Range
SSRAM	N	0–3fffff
Flash	N	400000–7fffff
UART	Y	c00000–c0000b
MAC	Y	d00000–d00040
Video Codec	N	e00000–e000ff

Based on the information above, CoreConsole generates the SoC subsystem corresponding to the Actel IP cores present. It also generates a BFM script, which accesses all the registers in the Actel IP cores.

Bus Fabric Selection

You may select one of a number of bus fabrics in CoreConsole. For example, you can select AMBA AHB-Lite. However, this selection is irrelevant for the CoreMP7 BFM, because it only generates native CoreMP7 bus-based transactions.

Automatic BFM Script

At this point, having run CoreConsole to completion, a BFM script is available. This would look something like the following:

```
read B uart 0;
write B uart 4 bb;
read B uart 8;
write B mac 30 11;
readcheck B mac 11;
```

Run BFM

The developer may run the BFM with the automatic script, or instead, edit the script to put in bus transactions to/from any new logic that has been added to the SoC. For example, transactions to/from the registers in the new Video Codec block could be added.

The skeleton system-level testbench, generated by CoreConsole, could also be modified, to add some external resources (e.g., models of SSRAM and Flash) and some high-level tasks.

Upon running the system simulation, messages appear in the console window of the simulation tool. Also, a color-coded HTML log file is generated. This may look something like the following:

```
# read B uart 0;
Reading offset 0 of uart – data = 0x1c
# write B uart 4 bb;
Writing 0xbb to offset 4 of uart
# read B uart 8;
Reading offset 8 of uart – data = 0x28
# write B mac 30 11;
Writing 0x11 to offset 30 of mac
# readcheck B mac 11;
Reading offset 0 of mac
Error: Expected data = 0x11, Actual data =
   0x22

Test Failed, with 1 error
```

AC Parameters

This section gives the AC timing parameters of the CoreMP7 processor.

Timing Diagrams

Timing diagrams are shown in Figure 11, Figure 12 on page 20, Figure 13 on page 20, Figure 14 on page 21, and Figure 15 on page 21.

Data Access Timing

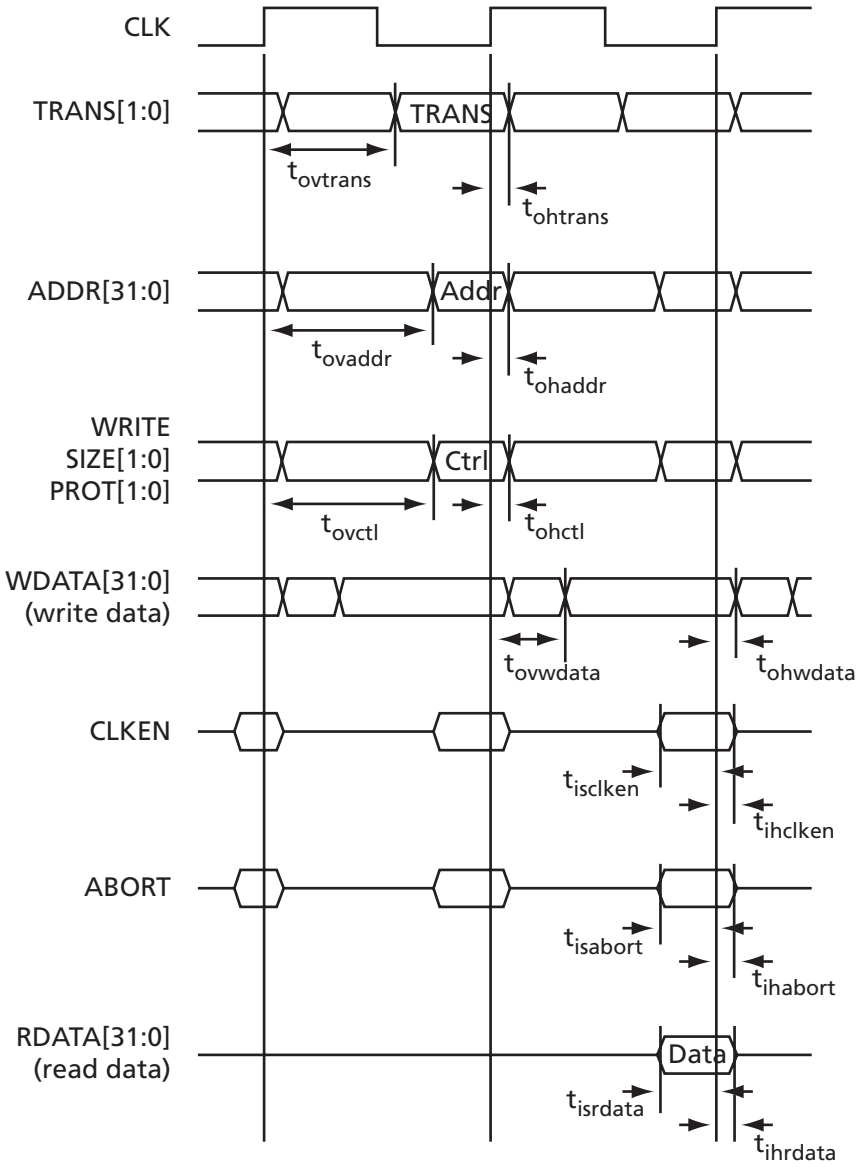

Figure 11 • **Data Access Timing**

Coprocessor Timing

The Coprocessor timing is included for completeness although it is expected that the Coprocessor interface is omitted in most deployments of the CoreMP7.

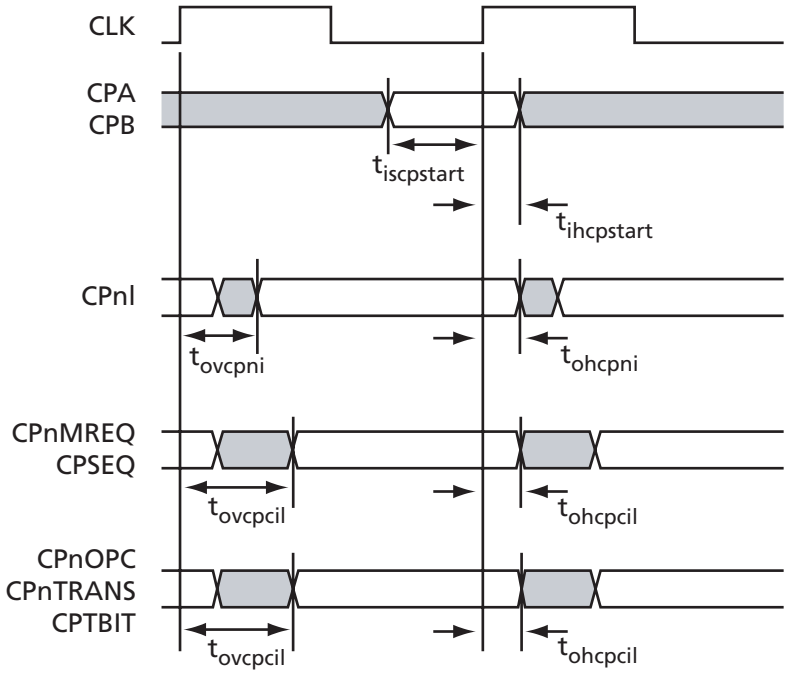

Figure 12 • **Coprocessor Timing**

Exception Timing

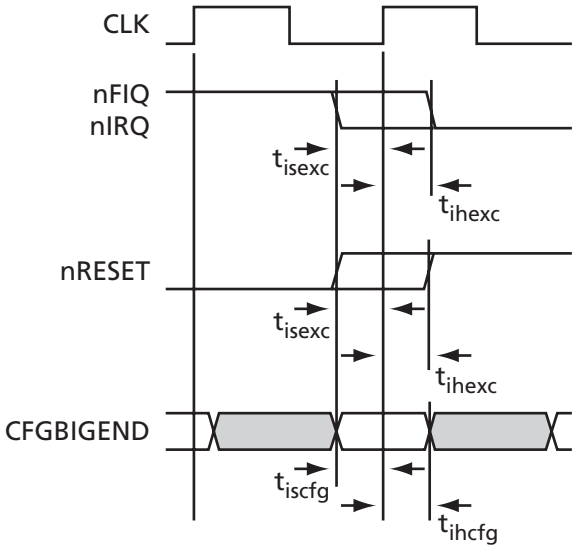

Figure 13 • **Exception Timing**

Debug Timing

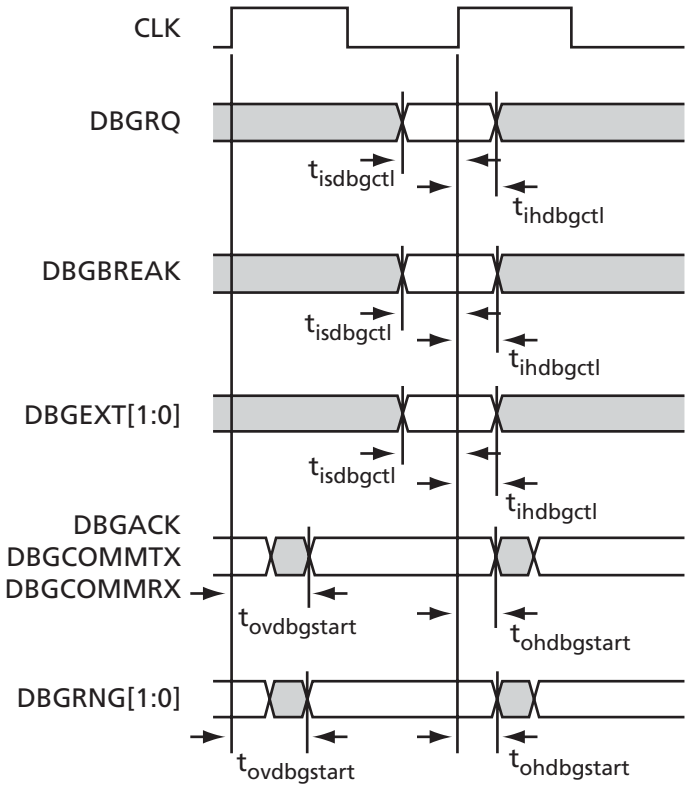

Figure 14 • **Debug Timing**

Scan Timing

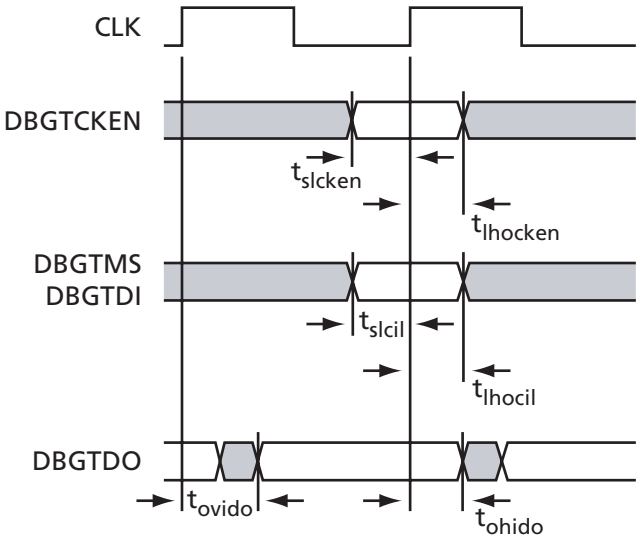

Figure 15 • **Scan Timing**

AC Timing Parameter Definitions

AC Timing Parameters shows target AC parameters. All figures are expressed as percentages of the **CLK** period at maximum operating frequency.

Note: Where 0% is shown, this indicates the hold time to clock edge plus the maximum clock skew for internal clock buffering.

Table 6 • **AC Timing Parameters**

Symbol	Parameter	Min	Max
t_{CYC}	**CLK** cycle time	100%	–
$t_{ISCLKEN}$	**CLKEN** input setup to rising **CLK**	60%	–
$t_{IHCLKEN}$	**CLKEN** input hold from rising **CLK**	–	0%
$t_{ISABORT}$	**ABORT** input setup to rising **CLK**	40%	–
$t_{IHABORT}$	**ABORT** input hold from rising **CLK**	–	0%
$t_{ISRDATA}$	**RDATA** input setup to rising **CLK**	10%	–
t_{ISRST}	**nRESET** input setup to rising **CLK**	90%	–
t_{ISTRST}	**DBGnTRST** input setup to rising **CLK**	25%	
$t_{IHRDATA}$	**RDATA** input hold from rising **CLK**	–	0%
$t_{OCPTBIT}$	Rising **CLK** to **CPTBIT** valid	–	90%
t_{ODBG}	Rising **CLK** to **DBGnEXEC**, **DBGINSTRVALID** valid	–	40%
t_{OLOMO}	Rising **CLK** to **DMORE**, **LOCK** valid	–	90%
t_{OVADDR}	Rising **CLK** to **ADDR** valid	–	90%
t_{OHADDR}	**ADDR** hold time from rising **CLK**	>0%	–
t_{OVCTL}	Rising **CLK** to control valid	–	90%
t_{OHCTL}	Control hold time from rising **CLK**	>0%	–
$t_{OVTRANS}$	Rising **CLK** to transaction type valid	–	50%
$t_{OHTRANS}$	Transaction type hold time from rising **CLK**	>0%	–
$t_{OVWDATA}$	Rising **CLK** to **WDATA** valid	–	40%
$t_{OHWDATA}$	**WDATA** hold time from rising **CLK**	>0%	–
$t_{ISCPSTAT}$	**CPA**, **CPB** input setup to rising **CLK**	20%	–
$t_{IHCPSTAT}$	**CPA**, **CPB** input hold from rising **CLK**	–	0%
$t_{OVCPCTL}$	Rising **CLK** to coprocessor control valid	–	80%
$t_{OHCPCTL}$	Coprocessor control hold time from rising **CLK**	>0%	–
t_{OVCPNI}	Rising **CLK** to coprocessor **CPnI** valid	–	40%
t_{OHCPNI}	Coprocessor **CPnI** hold time from rising **CLK**	>0%	–
t_{ISEXC}	**nFIQ**, **nIRQ**, input setup to rising **CLK**	10%	–
t_{IHEXC}	**nFIQ**, **nIRQ**, **nRESET** hold from rising **CLK**	–	0%
t_{ISCFG}	**CFGBIGEND** setup to rising **CLK**	10%	–
t_{IHCFG}	**CFGBIGEND** hold from rising **CLK**	–	0%
$t_{ISDBGCTL}$	**DBGBREAK**, **DBGEXT**, **DBGRQ** input setup to rising **CLK**	10%	–
$t_{ISDBGSTAT}$	Debug status inputs setup to rising **CLK**	10%	–

Table 6 • **AC Timing Parameters (Continued)**

Symbol	Parameter	Min	Max
$t_{IHDBGSTAT}$	Debug status inputs hold from rising **CLK**	–	0%
$t_{OVDBGCTL}$	Rising **CLK** to debug control valid	–	40%
$t_{OHDBCTL}$	Debug control hold time from rising **CLK**	>0%	–
$t_{ISTCLKEN}$	**DBGTCKEN** input setup to rising **CLK**	60%‘	–
$t_{IHTCKEN}$	**DBGTCKEN** input hold from rising **CLK**	–	0%
t_{ISTCTL}	**DBGTDI**, **DBGTMS** input setup to rising **CLK**	35%	–
t_{IHTCTL}	**DBGTDI**, **DBGTMS** input hold from rising **CLK**	–	0%
t_{OVTDO}	Rising **CLK** to **DBGTDO** valid	–	20%
t_{OHTDO}	**DBGTDO** hold time from rising **CLK**	>0%	–
$t_{OVDBGSTAT}$	Rising **CLK** to debug status valid	40%	–
$t_{OHDBGSTAT}$	Debug status hold time	>0%	–

Debug

The ARM Debug Architecture uses a protocol converter box to allow the debugger to talk via a Joint Test Action Group (JTAG) port directly to the core. In effect, the scan chains in the core that are required for test are re-used for debugging.

The architecture uses the scan chains to insert instructions directly in to the ARM core. The instructions are executed on the core and, depending on the type of instruction that has been inserted, the core or the system state can be examined, saved, or changed. The architecture has the ability to execute instructions at a slow debug speed or to execute instructions at system speed (for example, if access to an external memory was required).

The fact that the debugger is actually using the JTAG scan chains to access the core is of no importance to the user, as the front end debugger remains exactly the same. The user could still use the debugger with a monitor program running on the target system or with an instruction set simulator that runs on the debugger host. In each case the debugging environment is the same.

The advantages of using the JTAG port are:

- Hardware access required by a system for test is re-used for debug.
- Core state and system state can be examined via the JTAG port.
- The target system does not have to be running in order to start debug.

A monitor program for example requires that some target resources are running in order for the monitor program to run.

- Traditional breakpoints and watchpoints are available.
- On-chip resources can be supplemented.
- For example, the ARM Debug Architecture uses an on-chip macro-cell to enhance the debugging facilities available.
- A separate UART to communicate with the monitor program is not required.

The debugging of the target system requires the following:

- A PC host computer running Windows to run the debugger software
- An EmbeddedICE Protocol Converter, a separate box which converts the serial interface to signals compatible with the JTAG interface and a target system with a JTAG interface and an ARM Debug Architecture compliant core.

Once the system is connected, the debugger can start communicating with the target system via the RVI-ME (which is an EmbeddedICE Interface Converter).

The debug extensions consist of several scan chains around the processor core, and some additional signals that are used to control the behavior of the core for debug purposes. The most significant of these additional signals are as follows:

BREAKPT: This core signal enables external hardware to halt processor execution for debug purposes. When HIGH during an instruction fetch, the instruction is tagged as breakpointed, and the core stops if this instruction reaches execute.

DBGRQ: This core signal is a level-sensitive input that causes the CPU core to enter debug state when the current instruction has completed.

DBGACK: This core signal is an output from the CPU core that goes HIGH when the core is in debug state so that external devices can determine the current state of the core.

RealView ICE uses these, and other signals, through the debug interface of the processor core, for example by writing to the control register of the EmbeddedICE logic. For more details, refer to the debug interface section of the ARM datasheet or technical reference manual for your core.

JTAG Debug Interface

The RVI-ME ICE run control unit is supplied with a short ribbon cable. These both terminate in a 20-way 2.54 mm pitch IDC connector. You can use the cable to mate with a keyed box header on the target. The pinout is shown in Figure 16.

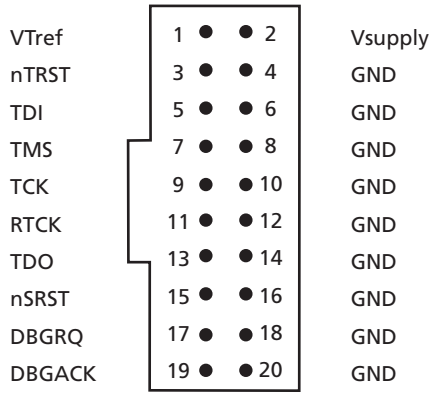

Figure 16 • **JTAG Interface Pinout**

The signals on the JTAG interface are shown in Table 7.

Table 7 • **JTAG Signals**

Signal	I/O	Description
DBGACK	–	This pin is connected in the RealView ICE run control unit, but is not supported in the current release of the software. It is reserved for compatibility with other equipment to be used as a debug acknowledge signal from the target system. It is recommended that this signal is pulled LOW on the target.
DBGRQ	–	This pin is connected in the RealView ICE run control unit, but is not supported in the current release of the software. It is reserved for compatibility with other equipment to be used as a debug request signal to the target system. This signal is tied LOW. When applicable, RealView ICE uses the core's scanchain 2 to put the core in debug state. It is recommended that this signal is pulled LOW on the target.
GND	–	Ground
nSRST	Input/ Output	Open collector output from RealView ICE to the target system reset. This is also an input to RealView ICE so that a reset initiated on the target can be reported to the debugger. This pin must be pulled HIGH on the target to avoid unintentional resets when there is no connection.
nTRST	Output	Open collector output from RealView ICE to the Reset signal on the target JTAG port. This pin must be pulled HIGH on the target to avoid unintentional resets when there is no connection.
RTCK	Input	Return Test Clock signal from the target JTAG port to RealView ICE. Some targets must synchronize the JTAG inputs to internal clocks. To assist in meeting this requirement, you can use a returned, and retimed, **TCK** to dynamically control the **TCK** rate. RealView ICE provides Adaptive Clock Timing that waits for **TCK** changes to be echoed correctly before making further changes. Targets that do not have to process **TCK** can simply ground this pin.
TCK	Output	Test Clock signal from RealView ICE to the target JTAG port. It is recommended that this pin is pulled LOW on the target.
TDI	Output	Test Data In signal from RealView ICE to the target JTAG port. It is recommended that this pin is pulled HIGH on the target.
TDO	Input	Test Data Out from the target JTAG port to RealView ICE. It is recommended that this pin is pulled HIGH on the target.
TMS	Output	Test Mode signal from RealView ICE to the target JTAG port. This pin must be pulled HIGH on the target so that the effect of any spurious **TCK**s when there is no connection is benign.
Vsupply	Input	This pin is not connected in the RealView ICE run control unit. It is reserved for compatibility with other equipment to be used as a power feed from the target system.
VTref	Input	This is the target reference voltage. It indicates that the target has power, and it must be at least 0.628 V. **VTref** is normally fed from **Vdd** on the target hardware and might have a series resistor (though this is not recommended). There is a 10 k pull-down resistor on **VTref** in RealView ICE.

The EmbeddedICE logic which implements the on-chip debug function in the CoreMP7 debug architecture is described in detail in the *ARM7TDMI-S (rev 4) Technical Reference Manual* (ARM DDI0234A), published by ARM Limited, and is available via Internet at www.arm.com.

The CoreMP7 debug architecture uses a JTAG port as a method of accessing the core. The debug architecture uses EmbeddedICE logic which resides on chip with the CoreMP7 core. The EmbeddedICE has its own scan chain that is used to insert watchpoints and breakpoints for the CoreMP7. The EmbeddedICE logic consists of two real-time watchpoint registers, together with a control and status register. One or both of the watchpoint registers can be programmed to halt the CoreMP7 core. Execution is halted when a match occurs between the values programmed into the EmbeddedICE logic and the values currently appearing on the address bus, databus, and some control signals. Any bit can be masked so that its value does not affect the comparison. Either watchpoint register can be configured as a watchpoint (i.e., on a data access) or a break point (i.e., on an instruction fetch). The watchpoints and breakpoints can be combined such that:

- The conditions on both watchpoints must be satisfied before the CoreMP7 is stopped. The CHAIN functionality requires two consecutive conditions to be satisfied before the core is halted. An example of this would be to set the first breakpoint to trigger on an access to a peripheral and the second to trigger on the code segment that performs the task switching. Therefore the breakpoints trigger the information regarding which task has switched out that will be ready for examination.
- The watchpoints can be configured such that a range of addresses are enabled for the watchpoints to be active. The RANGE function allows the breakpoints to be combined such that a breakpoint is to occur if an access occurs in the bottom 256 bytes of memory but not in the bottom 32 bytes.

The CoreMP7 core has a Debug Communication Channel function in-built. The debug communication channel allows a program running on the target to communicate with the host debugger or another separate host without stopping the program flow or even entering the debug state. The debug communication channel is accessed as coprocessor 14 by the program running on the CoreMP7 core. The debug communication channel allows the JTAG port to be used for sending and receiving data without affecting the normal program flow. The debug communication channel data and control registers are mapped in to addresses in the EmbeddedICE logic.

Table 8 • **Debug Communication Channel Signals**

Signal Name	Type	Description
TMS	Input	**Test Mode Select.** The TMS pin selects the next state in the TAP state machine.
TCK	Input	**Test Clock.** This allows shifting of the data in, on the TMS and TDI pins. It is a positive edge triggered clock with the TMS and TCK signals that define the internal state of the device.
TDI	Input	**Test Data In.** This is the serial data input for the shift register.
TDO	Output	**Test Data Output.** This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal.
nTRST	Input	**Test Reset.**The nTRST pin can be used to reset the test logic within the EmbeddedICE logic.
RTCK	Output	**Returned Test Clock.** Extra signal added to the JTAG port. Required for designs based on COREMP7 processor core. Multi-ICE (development system from ARM) uses this signal to maintain synchronization with targets having slow or widely varying clock frequency. For details, refer to the *Multi-ICE System Design Considerations Application Note 72 (ARM DAI 0072A)*.

The EmbeddedICE logic contains 16 registers, as shown in Table 9. The CoreMP7 debug architecture is described in detail in *ARM7TDMI-S (rev 4) Technical Reference Manual* (ARM DDI0234A), published by ARM Limited, and is available via Internet at www.arm.com.

Table 9 • **EmbeddedICE Logic Registers**

Name	Width	Description	Address
Debug Control	6	Force debug state, disable interrupts	00000
Debug Status	5	Status of debug	00001
Debug Comms Control Register	32	Debug communication control register	00100
Debug Comms Data Register	32	Debug communication data register	00101
Watchpoint 0 Address Value	32	Holds watchpoint 0 address value	01000
Watchpoint 0 Address Mask	32	Holds watchpoint 0 address mask	01001
Watchpoint 0 Data Value	32	Holds watchpoint 0 data value	01010
Watchpoint 0 Data Mask	32	Holds watchpoint 0 data mask	01011
Watchpoint 0 Control Value	9	Holds watchpoint 0 control value	01100
Watchpoint 0 Control Mask	8	Holds watchpoint 0 control mask	01101
Watchpoint 1 Address Value	32	Holds watchpoint 1 address value	10000
Watchpoint 1 Address Mask	32	Holds watchpoint 1 address mask	10001
Watchpoint 1 Data Value	32	Holds watchpoint 1 data value	10010
Watchpoint 1 Data Mask	32	Holds watchpoint 1 data mask	10011
Watchpoint 1 Control Value	9	Holds watchpoint 1 control value	10100
Watchpoint 1 Control Mask	8	Holds watchpoint 1 control mask	10101

Ordering Information

All variants of the CoreMP7 soft IP core are included in the CoreConsole IDP. To use CoreMP7, contact your local sales representative and order the CoreConsole IDP development tool, part number, COR-PC-N-1YR.

List of Changes

The following table lists critical changes that were made in the current version of the document.

Previous Version	Changes in Current Version (v2.3)	Page
v2.2	The datasheet was updated to include Fusion devices.	NA
	Table 1 was updated.	2
	The "CoreMP7 Variants" section was updated.	13
v2.1	Table 1 was updated.	2
v2.0	The "No Coprocessor Interface" section was updated.	13
	The "Little Endian Only" section was updated.	13

Datasheet Categories

In order to provide the latest information to designers, some datasheets are published before data has been fully characterized. Datasheets are designated as "Product Brief," "Advanced," and "Production." The definitions of these categories are as follows:

Product Brief

The product brief is a summarized version of an advanced or production datasheet containing general product information. This brief summarizes specific device and family information for unreleased products.

Advanced

This datasheet version contains initial estimated information based on simulation, other products, devices, or speed grades. This information can be used as estimates, but not for production.

Unmarked (production)

This datasheet version contains information that is considered to be final.

www.actel.com

Actel Corporation

2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.

Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Phone +44 (0) 1276 401 450
Fax +44 (0) 1276 401 490

Actel Japan
www.jp.actel.com

EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong
www.actel.com.cn

Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488

51700060-3/2.06