CoreMP7 User's Guide

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200057-1/1.06

Release: January 2006

Trademarks

Table of Contents

Introduction

The CoreMP7 is an Actel technology optimized ARM7™ processor and is fully compatible with the ARM7 architecture. For further information on the ARM7 architecture, refer to the *ARM7 Technical Reference Manual* (DDI0234A-7TMIS-R4.pdf), published by ARM® Corporation, available for download from the ARM web site (www.arm.com).

The CoreMP7 is supplied with an MP7Bridge (see Figure 3-1 on page 13) for interfacing to an Advanced Microcontroller Bus Architecture (AMBA) Advanced High-Performance Bus (AHB) based bus system such as the one generated by the Actel CoreConsole IP Deployment Platform (IDP). CoreConsole is used to create a system project which can be directly imported into the Actel Libero® Integrated Design Environment (IDE) for simulation, synthesis, layout, and bitstream generation for Actel FPGAs.

CoreMP7 Processor

The CoreMP7 is a general purpose 32-bit microprocessor offering high-performance and low power consumption. It is fully compatible with the ARM7TDMI-S™ processor from ARM. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The RISC simplicity results in a high instruction throughput and impressive 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.

1 Software Development

ARM has developed a version of their RealView Developer Suite for use with CoreMP7 called the RealView Developer Kit (RVDK). The RVDK is available from Actel and provides a complete development environment for embedded systems applications running on CoreMP7. It consists of a suite of tools, together with supporting documentation and examples. The tools enable you to write, build, and debug the software program for your application, either on target hardware or on a software simulator.

You can use the compilation tools to build C, C++, or ARM assembly language programs. The following tools are included in the RVDK:

- ARM and Thumb® C and C++ compiler, armcc
- ARM and Thumb assembler, armasm
- ARM linker, armlink
- ARM librarian, armar
- ARM image conversion utility, fromelf
- Supporting libraries

There are other tools included in the RVDK as well, including the RealView Debugger.

The interface and project management of RVDK is provided by the RealView Debugger.

A full set of manuals is provided by ARM with the RVDK.

2

CoreMP7 Design Entry Flow

The Libero IDE automatically manages CoreMP7 through the various point tool tools when it is instantiated in a CoreConsole project system design. The project system can consist of only the CoreMP7 itself or it can include a range of subsystem and higher level IP blocks. The overall flow is shown in Figure 2-1.

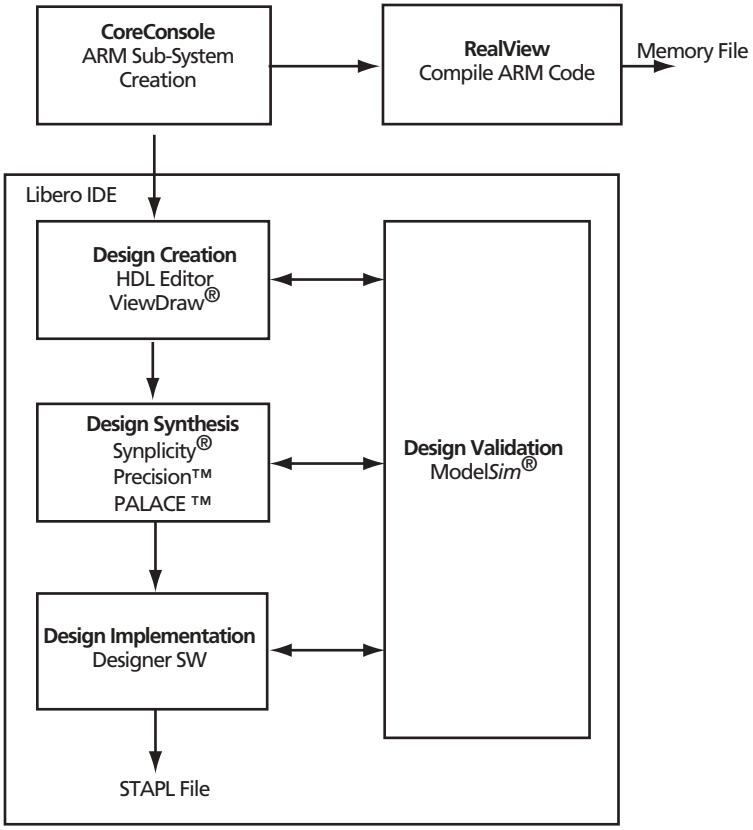

Figure 2-1. MP7 Design Entry Flow

To import a CoreConsole project into the Libero IDE, locate the CCP file for that project in the CoreConsole Libero IDE Export folder hierarchy. The CCP file is an XML file that fully describes

the CoreConsole project. It appears in the CoreConsole Projects section of the Libero IDE as shown in Figure 2-2.

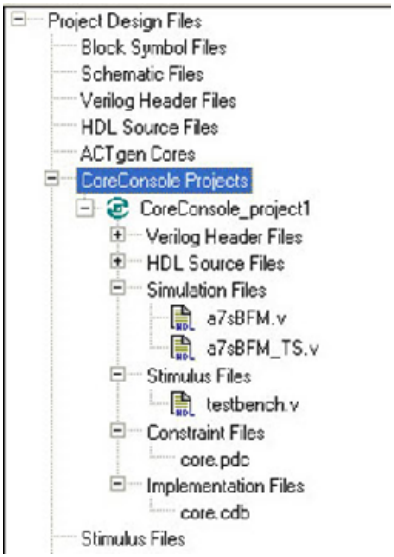

Figure 2-2. CoreConsole Project Section in Libero IDE

The CoreMP7 files associated with this flow are shown in Figure 2-3. Normally the CoreConsole system project imported into Libero IDE consists of a range of RTL IP components alongside the CoreMP7 that is implemented in a secure format that can only be read and used within the tools utilized by the Libero IDE.

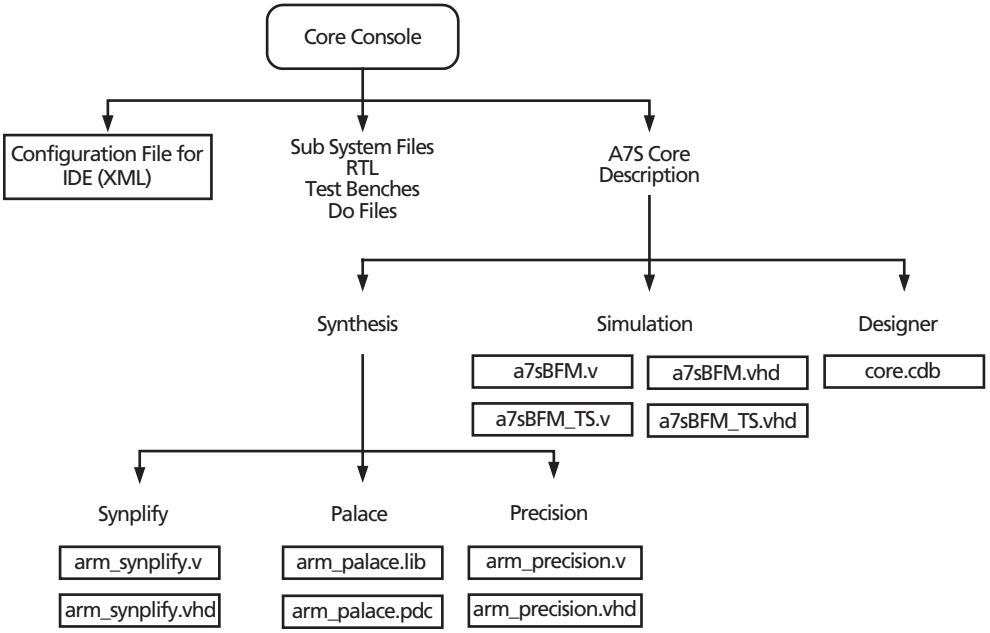

Figure 2-3. CoreMP7 File Flow

The Libero IDE manages all the files imported from CoreConsole, and passes them to the appropriate tools. The secure CoreMP7 implementation is represented inside the Libero IDE as A7S when the RTL wrapping has been removed (See Figure 2-4 and Figure 2-5).

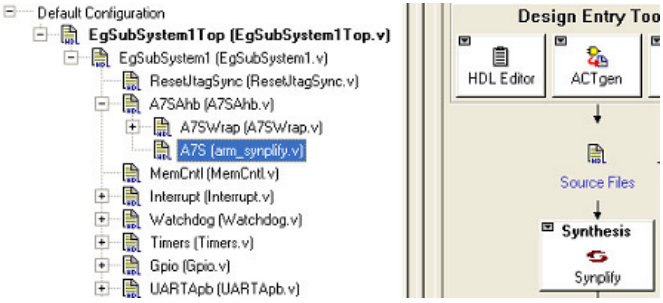

Figure 2-4. A7S Synplify in the Libero IDE

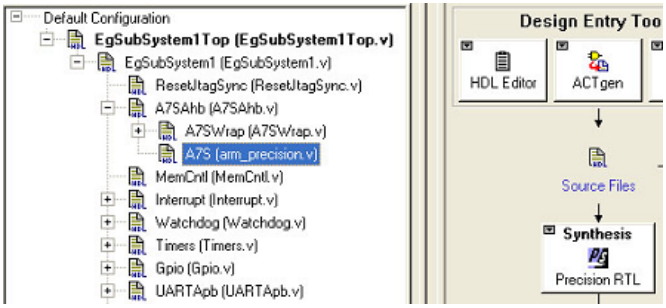

Figure 2-5. A7S Precision in the Libero IDE

CoreMP7 Security

CoreMP7 is imported as a CDB file along with a netlist that instantiates the CoreMP7 core. The CoreMP7 core is delivered in a blackbox that allows you to access the top-level I/O and use the core in Actel M7 ProASIC3/E devices. The contents of the blackbox are kept from view. The CoreMP7 core top-level interface is visible (external ports that need to be routed to the rest of the design). Timing constraints and analysis are done at the interface of the CoreMP7 core. The placement and routing of the CoreMP7 within the FPGA fabric core is fixed. The Libero IDE is aware of the core's placement and will fill in the unused tiles around the core with the IP that you have put together around CoreMP7 in your CoreConsole project system. The CoreMP7 blackbox cannot be unlocked, and can only be programmed in M7 ProASIC3/E devices.

3

MP7Bridge Wrapper

The MP7Bridge is an important component used when building a CoreMP7-based project system. The MP7Bridge component converts the native signals from the CoreMP7 processor into an AMBA AHB master interface suitable for connection to an AHB bus. It also includes circuitry that handles reset signals and the signals that connect to the ARM RealView ICE JTAG port. The incoming hardware reset signal is synchronized to the system clock, and provision is made for handling a watchdog generated reset when a watchdog component is included in the system. Some circuitry to condition the RealView ICE signals is also included.

The details of the MP7Bridge are covered in the *CoreMP7 Subsystem Library Guide*. The interface is shown in Figure 3-1.

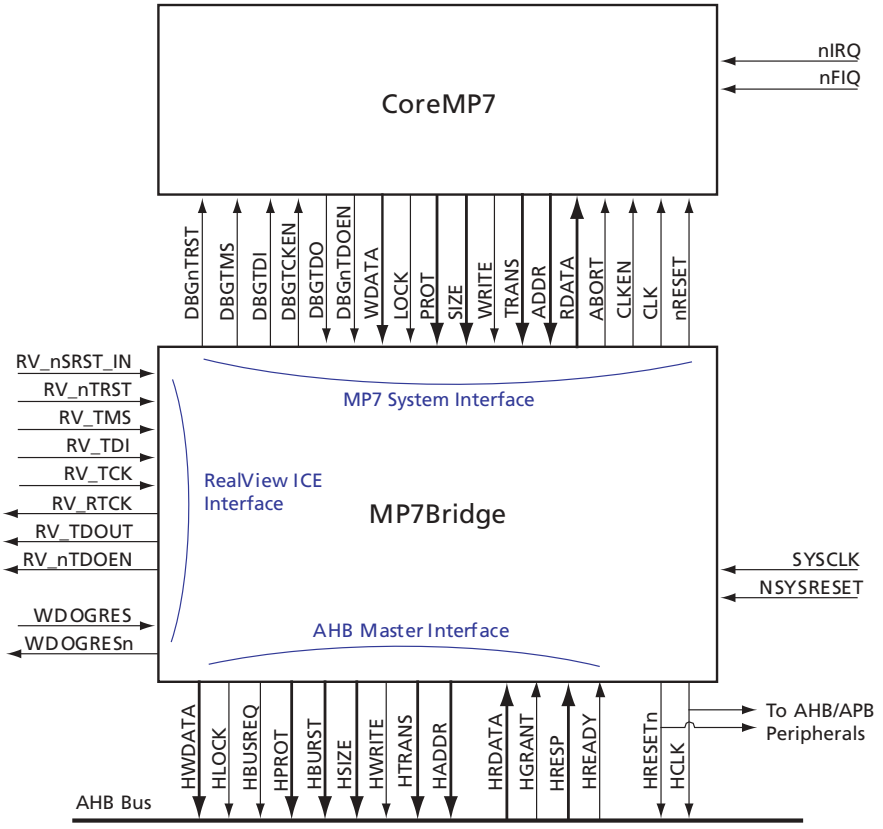

Figure 3-1. Interfacing CoreMP7 with MP7Bridge

4

Bus Functional Model (BFM)

Introduction

During the development of an FPGA-based SoC, several stages of testing can be performed. 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 processor
- Hardware and software co-verification, using a full-functional model of the processor in Verilog, VHDL, or SWIFT form, or using a tool such as Seamless®

Due to the rapid prototyping capability of FPGA integration of hardware and software, this often occurs earlier in the development cycle than it would for ASIC targets. Therefore, hardware and software co-verification, which can be very slow, is not as critical an issue for most FPGA-based system-level designs.

A software simulator is available as part of the CoreMP7 RVDK tools. The RealView Instruction Set Simulator provides ARM7 instruction accurate simulation, as well as powerful features, such as integration with the RealView debugger.

The CoreConsole tool provides a means for you to stitch together IP blocks using a bus fabric of your choice. It generates a system testbench controlled by a script-driven BFM of the ARM7 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 ARM7 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 4-1 shows the various components within a typical system-level testbench:

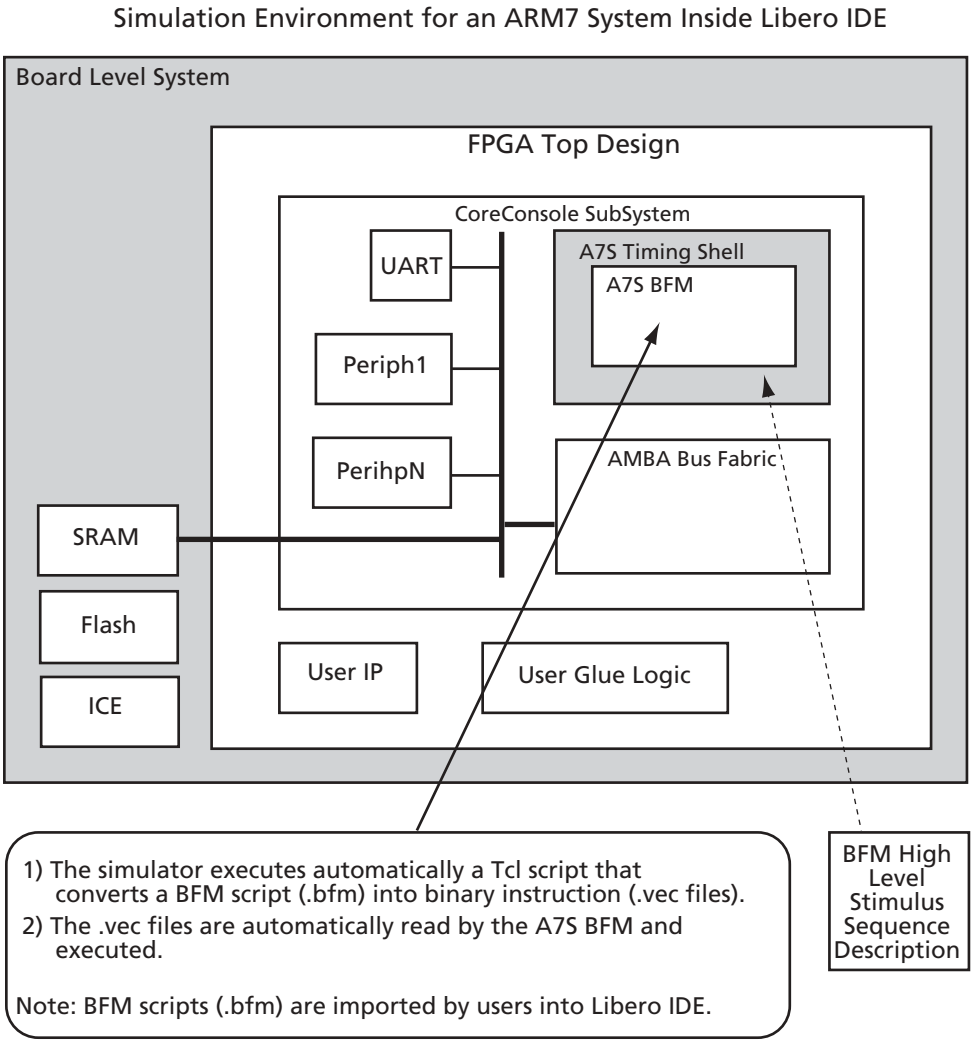

Figure 4-1. BFM Simulation

In Figure 4-1 on page 16 it is assumed that you have specified your processor subsystem by selecting the processor, bus fabric, IP blocks, and the memory system using CoreConsole.

When you build the system, you also specify the memory map of the system. Based on this information, CoreConsole generates the following outputs:

- Verilog/VHDL model of SoC subsystem
- Verilog/VHDL models of IP cores
- ARM7 BFM
- BFM test script
- System-level skeleton testbench

The BFM acts as a pin-for-pin replacement of the CoreMP7 in the project system. It initiates bus transactions on the native ARM7 bus. These transactions are cycle-accurate with real bus cycles that the CoreMP7 would produce. It has no knowledge, however, of real ARM7 instructions.

At this point the BFM may be used to run a basic test of the system using the skeleton system testbench. CoreConsole uses known attributes (using SPIRIT XML descriptions) from registers or addressable locations within the IP cores, along 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 the expected data should be on reads.

You can edit the system Verilog/VHDL to add new design blocks, such as a VideoCodec. You can also edit the system-level testbench to include tasks that test any newly added functionality, or to add stubs to allow more complex system testing involving the IP cores. The BFM input scripts may also 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 ARM7 BFM), as well as from the outside (via testbench tasks).

Functionality

This section describes the specific functionality of the ARM7 BFM. The BFM models the ARM7 native bus. Specifically, it models 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
```

ARM7 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 processor core in the system testbench.

ARM7 Bus Cycle Accuracy

The bus cycle timings for the ARM7 native bus signals are specified in the *ARM7TDMI 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 ("BFM Script Language"). 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 ARM7 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 an 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;
```

waitfiq

This command causes the BFM to wait until an interrupt event is seen on the nFIQ pin before proceeding with the execution of the remainder of the script.

Syntax

```
waitfiq;
```

Timing Shell

There is a 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. In this system the developer requires two Actel IP cores: the Core10/100 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 (Table 4-1).

Table 4-1. Registers for CoreUART

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 RW, so there won't 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.

Automatic BFM Script

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

```
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 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 VideoCodec block could be added.

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

When running the system simulation, the following messages appear in the console window of the simulation tool:

```
# 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
```

5

Debug

The ARM debug architecture uses a protocol converter box to allow the RVDK debugger to talk via a Joint Test Action Group (JTAG) port directly to the core. 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 using the JTAG scan chains to access the core is not a problem, because the front-end debugger remains the same. You can 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 either 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 the debug. A monitor program, for example, requires that some target resources are running in order for the monitor program to run.
- The traditional breakpoints and watchpoints are available.
- On-chip resources can be added to.

The debugging of the target system requires the following:

- A host computer to run the debugger software. The host could be a PC running Windows, Linux, or a Sun workstation.
- An EmbeddedICE Protocols Converter. This is a separate box that 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 (one of the types of EmbeddedICE Interface Converters).

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 described below.

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.

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 5-1.

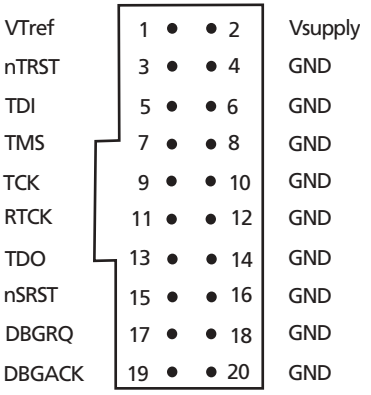

Figure 5-1. JTAG Interface Pinout

The signals on the JTAG interface are shown in Table 5-1.

Table 5-1. 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.

Table 5-1. JTAG signals (Continued)

Signal	I/O	Description
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 TCKs 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 kΩ 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 http://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 ARM7TDMI core 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 when the breakpoints trigger, the information regarding which task has switched out 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 has a built-in debug communication channel function. 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 a co-processor 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. The signals in the debug communication channel are listed in Table 5-2.

Table 5-2. 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 *Multi-ICE System Design Considerations Application Note 72 (ARM DAI 0072A)*.

The EmbeddedICE logic contains 16 registers as shown in Table 5-3. The CoreMP7 debug architecture is described in detail in the *ARM7TDMI-S (rev 4) Technical Reference Manual* (ARM DDI 0234A) published by ARM Limited, and is available via Internet at http://www.arm.com.

Table 5-3. Debug Communication Channel 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

A

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

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 *• EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200057-1/1.06