Rate This Document

Cortex-M1

Handbook

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200127-2

Release: November 2007

Trademarks

Table of Contents

Introduction

This document describes the architecture and implementation of the 32-bit Cortex™-M1 microprocessor developed by ARM® specifically for use in FPGAs. Cortex-M1 is a functional subset of Cortex-M3, which uses the M3 three-stage pipeline and runs the ARMv6-M instruction set. The streamlined Cortex-M1, developed for use in embedded applications, is designed to balance size and speed when implemented in an FPGA.

Key Features

- 32-bit RISC architecture (ARMv6-M)
- 32-bit AHB-Lite bus interface
- Separate Tightly Coupled Memory Interfaces
- Three-stage pipeline
- 32-bit ALU
- 1–32 interrupts with 4 priority levels per interrupt
- 4-GB memory addressing range (the upper 0.5 GB is reserved)
- Optional real-time debug unit
- JTAG interface

Benefits

- Fully implemented in FPGA fabric
- All microprocessor I/Os available to user
- No license fees or royalties
- Can run all existing Thumb® code
- Upward-compatible with Cortex-M3

Supported Actel FPGA Families

The Cortex-M1 version part number prefix is in parentheses.

- ProASIC®3 (M1A3P)
- ProASIC3E (M1A3PE)
- Fusion™ (M1AFS)
- IGLOO™ (M1AGL)

The Cortex-M1 soft IP core is a member of ARM's Cortex family of processors and has been optimized for use in Actel ARM-enabled FPGAs.

Cortex-M1 is supplied with an AMBA AHB-Lite compliant interface for inclusion in an AMBA-based processor system, such as one generated by the Actel CoreConsole IP deployment platform. The processor also has a separate tightly coupled memory (TCM) interface.

Utilization and Performance

Cortex-M1 can be implemented in several Actel FPGA devices. Table 1 gives typical utilization figures using standard synthesis tools for four different core configurations.

Table 1 · Cortex M1 Utilization Data

Device	Variant	v2.1		
		Frequency (MHz)	Area	Utilization
M1AFS600-2	No-debug	66.168	4410	31.90%
	Debug	60.111	9210	66.62%
M1AGL600V2	No-debug	25.168	4410	31.90%
	Debug	24.796	9210	66.62%
M1AGL600V5	No-debug	40.925	4410	31.90%
	Debug	40.793	9210	66.62%
M1A3P1000-2	No-debug	68.932	4410	17.94%
	Debug	61.645	9210	37.48%
M1A3PE1500-2	No-debug	68.55	4410	11.48%
	Debug	62.263	9210	23.98%

Note: Configuration is 0 kbytes ITCM, 0 kbytes DTCM, small multiplier, 1interrupt, no OS extensions, little-endian, and debug as shown.

1

Cortex-M1 Overview

Cortex-M1 Processor

Cortex-M1 is a general purpose 32-bit microprocessor that offers high performance and small size in FPGAs. The processor is highly configurable with tightly coupled memories, fast or small multiplier options, OS extensions, optional debug, and a 32-input interrupt controller. Cortex-M1 runs a subset of the Thumb-2 instruction set (ARMv6-M) that includes all base 16-bit Thumb instructions and a few Thumb-2 32-bit instructions (BL, MRS, MSR, ISB, DSB, and DMB). This enables writing very tight and efficient processor code, which is ideal for the limited memory typically found in deeply-embedded applications.

The main blocks in Cortex-M1 are shown in Figure 1-1 and include the processor core, the Nested Vectored Interrupt Controller (NVIC), the AHB interface, and the debug unit. The processor core supports 13 general purpose 32-bit registers, including a Link Register (LR), a Program Counter (PC), a Program Status Register (xPSR), and two banked Stack Pointers (SP). In addition to the AHB-Lite interface, there is a dedicated memory interface for access to the Instruction and Data Tightly Coupled Memories (ITCM and DTCM). The ITCM and DTCM are individually configurable from 0 bytes to 1,024 kbytes, providing ample space for optimizing memory transactions.

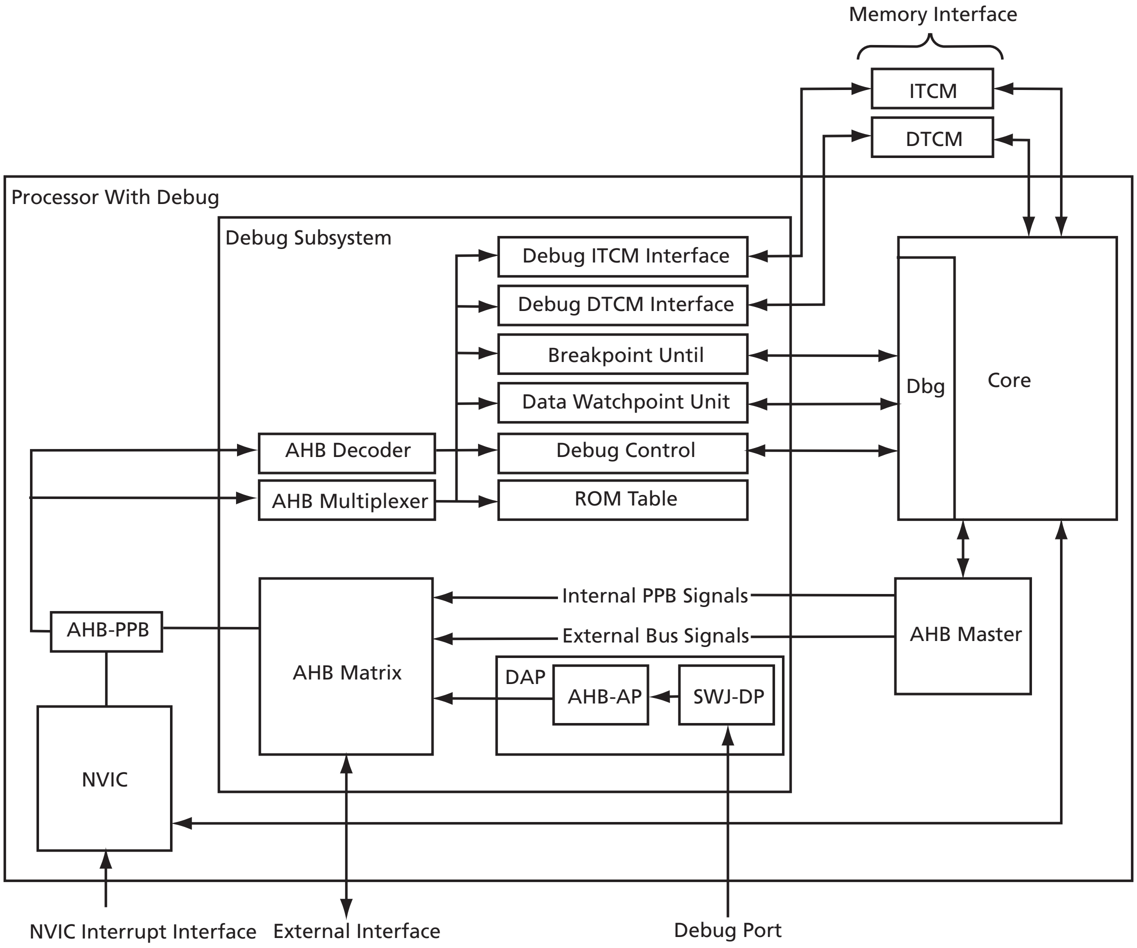


Figure 1-1 · Cortex-M1 Block Diagram with Debug

The NVIC is closely coupled to the Cortex-M1 core to achieve low-latency interrupt processing. The NVIC is configurable and supports 1, 4, 8, 16, or 32 interrupts with 4 levels of priority. The processor state is automatically saved on interrupt entry and restored on interrupt exit, with no instruction overhead to simplify software development.

The 16-bit length of the Cortex-M1 Thumb instruction allows it to approach twice the code density of the standard 32-bit ARM code, while retaining most of the ARM performance advantages over a traditional 16-bit processor that uses 16-bit registers. This is possible because Thumb code operates on the 32-bit register set in the processor. Thumb code is able to provide up to 65% of the code size of ARM, and 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system.

2

Delivery and Deployment

Cortex-M1 is delivered as a CCZ (CoreConsole Component ZIP file), which can be added to the CoreConsole IP database. Using CoreConsole, the processor core can be instantiated, configured, and stitched into an AMBA-based subsystem. The implementation files used in a Libero® Integrated Design Environment (IDE) project for simulation and synthesis can then be generated. These files consist of the BFM files, test wrapper, and the Cortex-M1 secured CDB black box model file, which is the Cortex-M1 core that has gone through the place-and-route process and will be instantiated in the user device. This deployment flow is adopted to ensure that the design is kept completely secure at all times.

Initially, the Cortex-M1 processor is being made available for use in Actel M1 devices with only one user-selectable option: with or without debug. The core is configured with 0 kbytes ITCM, 0 kbytes DTCM, small multiplier, little endian, no OS extensions, and 1 interrupt. Additional versions of the core are planned that will make all of the user-configurable options available.

CortexM1Top (Cortex-M1 top level)

The user need only select one core, corresponding to Cortex-M1 in the CoreConsole GUI. Cortex-M1 has built-in AHB-Lite Master support, so can connect directly an AHB bus (e.g., CoreAHB[Lite]), unlike CoreMP7, which requires CoreMP7Bridge to bridge from the ARM7TDMI-S native memory interface to an AHB bus. A top-level wrapper around the Cortex-M1 core contains the following functionality:

- VHDL and Verilog versions
- Generics/parameters to allow users to select the following options (to the extent that they are selectable) in CoreConsole:
  - TCM size (0 kbytes, 2 kbytes, ... 1,024 kbytes, in "powers of 2" increments)
  - Multiplier type (small or fast)
  - Debug interface (RealView, FlashPro3, or none)
  - Die
  - OS extensions
  - Number of interrupts

  Note: Not all combinations of these parameters are available to the user.
- Reset generation (combining watchdog reset from CoreWatchdog, if present and the soft reset output from Cortex-M1)
- Buffering of reset signals
- Multiplexing of debug-related JTAG signals (between RealView and FlashPro3 JTAG interfaces)

Cortex-M1 SPIRIT Description

The SPIRIT IP-XACT description of Cortex-M1 is implemented at the top level (CortexM1Top). It provides only valid parameter selection choices in the CoreConsole configuration dialog box. It also takes care of ensuring the correct CDB black box module is exported to the Libero IDE project (as described in the "CortexM1Top (Cortex-M1 top level)" section).

Cortex-M1 I/O Signals

CortexM1Top has fixed I/O signals, regardless of the Cortex-M1 variant chosen. All 32 interrupt signals and all the debug interface signals are always present, regardless of the configuration. The SPIRIT description is written so that unused inputs are tied inactive and unused outputs are left unconnected in the instantiation of CortexM1Top. The I/O signals present in CortexM1Top are described in Table 2-1.

Table 2-1 · Cortex-M1 Signal Descriptions

Name	Width	Type	Description
HCLK	1	Input	Main processor clock
NSYSRESET	1	Input	External push-button/power-up reset
WDOGRES	1	Input	Watchdog reset to Cortex-M1
WDOGRESn	1	Output	Reset of watchdog timer
HRESETn	1	Output	Reset to other components in AHB system
RV_TCK	1	Input	RealView JTAG
RV_nTRST	1	Input	RealView JTAG
RV_TMS	1	Input	RealView JTAG
RV_TDI	1	Input	RealView JTAG
RV_nSRST_IN	1	Input	RealView JTAG
RV_TRCK	1	Input	RealView JTAG
RV_TDOUT	1	Output	RealView JTAG
RV_nTDOEN	1	Output	RealView JTAG
UJTAG_TCK	1	Input	FlashPro3 JTAG
UJTAG_TDI	1	Input	FlashPro3 JTAG
UJTAG_TMS	1	Input	FlashPro3 JTAG
UJTAG_TRSTB	1	Input	FlashPro3 JTAG
UJTAG_TDO	1	Output	FlashPro3 JTAG
IRQ[31:0]	32	Input	External Interrupts
NMI	1	Input	Non-maskable Interrupt
EDBGRQ	1	Input	External debug request
nTRST	1	Input	JTAG reset
JTAGTOP	1	Output	State Controller Indicator
nTDOEN	1	Output	JTAG data out enable
LOCKUP	1	Output	Core is locked up
HALTED	1	Output	Core is in Halt debug state
HREADY	1	Input	Slave ready signal

Table 2-1 · Cortex-M1 Signal Descriptions (continued)

Name	Width	Type	Description
HRESP	1	Input	AHB response signal
HRDATA[31:0]	32	Input	Data from Slave to Master
HTRANS[1:0]	2	Output	AHB transfer type signal
HBURST[2:0]	3	Output	AHB burst signal
HPROT[3:0]	4	Output	Transfer protection bits
HSIZE[2:0]	3	Output	Transfer size
HWRITE	1	Output	Transfer direction
HMASTLOCK	1	Output	Transfer is part of a locked sequence
HADDR[31:0]	32	Output	Transfer address
HWDATA[31:0]	32	Output	Data from Master to Slave

3

Supported Interfaces

The processor contains two bus interfaces:

- External interface
- Tightly Coupled Memory (TCM) interface

The processor also contains an internal Processor Peripheral Bus (PPB) for accesses to the Nested Vectored Interrupt Controller (NVIC), Data Watchpoint (DW) unit, and Breakpoint Unit (BPU).

External Interface

The external interface is an AMBA AHB-Lite bus interface. Descriptions of the bus signals are shown in Table 2-1 on page 10. Processor accesses and debug accesses to external AHB peripherals can occur over this bus interface. Because AHB fetches take two cycles longer than TCM fetches, instructions and data should be contained in TCM where possible. If on-chip FPGA memory is used for the processor, use TCM memory rather than SRAM mapped onto the AHB interface. This enables the highest possible performance.

Figure 3-1 shows the timing of a read without wait states on the external interface. The Address A0 is presented on **HADDR** one cycle later than the internal address is generated, and the returned data D0 is registered again before use in the processor. This enables the AHB peripherals sufficient time to use the address generated.

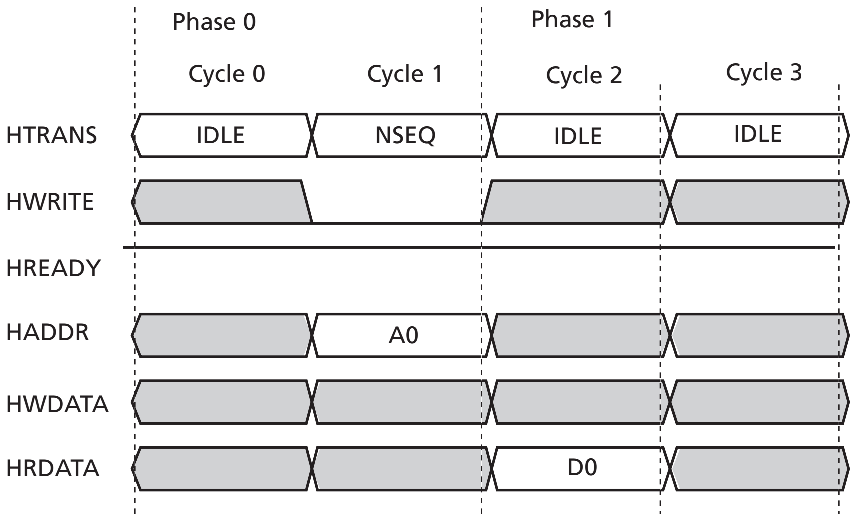


Figure 3-1 · AHB Read without Wait States

Processor accesses and debug accesses share the external interface. Debug accesses take priority over processor accesses. Giving the highest priority to debug means that debug cannot be locked out by a continuously executing stream of core instructions. Timing of processor accesses might be changed by the presence of debug accesses. Debug accesses tend to be infrequent, so debug accesses generally do not have a major impact on processor accesses.

Any vendor-specific components with an AHB interface can populate this bus.

Unaligned accesses to this bus are not supported.

Write Buffer

To prevent bus wait cycles from stalling the processor during data stores, buffered stores to the external interfaces go through a one-entry write buffer. If the write buffer is full, subsequent accesses to the bus stall until the write buffer has drained. The write buffer is only used if the bus waits for the data phase of the buffered store; otherwise the transaction is immediately completed on the bus.

The DMB and DSB instructions wait for the write buffer to drain before completing. If an interrupt arrives while DMB/DSB is waiting for the write buffer to drain, the processor returns to the opcode after the DMB/DSB, on completion of the interrupt. This is because interrupt processing is a so-called memory barrier operation. In other words, all reads and writes occurring before the interrupt appear to happen before the interrupt, so the DMB and DSB instructions must appear to have completed before the interrupt.

Tightly Coupled Memory (TCM) Interface

Core access to Tightly Coupled Memories (TCMs) is made exclusively through a dedicated core memory interface.

The core memory interface is comprised of the following:

- Instruction Tightly Coupled Memory interface to access ITCM
- Data Tightly Coupled Memory interface to access DTCM

Because reads are speculatively fetched from TCMs, Device and Strongly-Ordered memory types such as FIFOs in TCM space are not supported. The processor does not support wait states for the memory interfaces. Figure 3-2 shows the signal timings for ITCM. The DTCM signal timings are the same as ITCM signal timings.

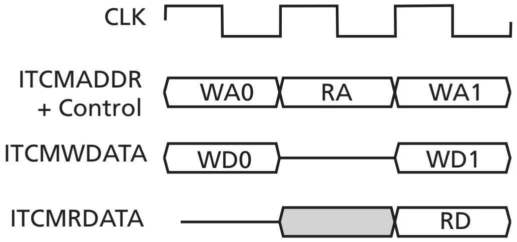


Figure 3-2 · ITCM Signal Timings

The write address, WA0, and write data, WD0, are presented in the same cycle. The Read Address (RA) is presented in one cycle, and the memory generates the Read Data (RD) in the next cycle. The sequence that Figure 3-2 shows is only possible on ITCM, where the RA read is an instruction fetch, and WA1 write is the product of a store instruction. The WA0-RA sequence is possible on DTCM.

In versions of the Cortex-M1 with non-zero sized ITCM, a TCM initialization state machine in the CortexM1Top wrapper module goes out onto the AHB bus and reads the ITCM initialization data from AHB location 0x00000000, and loads it into ITCM while holding the Cortex-M1 core itself in reset. Typically the NVM containing the program would have to be located at location 0x00000000.

In situations where there is only one large RAM available, but the user wants to run from RAM (for example, the user is debugging code on an Actel Cortex-M1 Development Board), the RAM should be mapped to location 0x6000000. The user should download the program (via debugger) to this location and start execution from this point. The user’s software should be written/linked assuming this location during the debug stage. This is the only SRAM area in the memory map that supports both instruction fetches and data accesses. When the debugging has finished, the software needs to be rewritten and linked to assume the starting point of location 0x00000000.

4

Cortex-M1 Features

Programmer's Model

The Cortex-M1 processor implements a subset of the Thumb-2 (ARMv7) architecture called ARMv6-M. This includes all of the 16-bit Thumb-2 instructions and some of the 32-bit instructions. The processor does not support ARM instructions.

Thumb-2 (ARMv7) is ARM's latest instruction set architecture (ISA) developed for their new Cortex family of processors. Offering increased efficiency and performance, the Thumb-2 ISA differs from previous ARM architectures in that it includes both 16- and 32-bit instructions. The previous 16-bit Thumb instruction set and 32-bit ARM instruction set were separate and had to be executed from different modes within the processor. The new Thumb-2 ISA gives users all the advantages of the reduced code size of the 16-bit Thumb instructions and the higher performance of the 32-bit ARM instructions. This is achieved in a single ISA that can be executed from a single mode within the processor, increasing the efficiency of the code as it executes and improving the performance and throughput of the Cortex family of processors.

Processor Operating States

The Cortex-M1 processor has two operating states:

- **Thumb state** – This is normal execution, running the set of 16-bit, halfword-aligned Thumb and Thumb-2 instructions; as well as the 32-bit BL, MRS, MSR, ISB, DSB, and DMB instructions.
- **Debug state** – This is the state when halting debug

Processor Operating Modes

The Cortex-M1 processor supports two modes of operation:

- **Thread mode** – Entered on Reset, and can be re-entered as the result of an exception return
- **Handler mode** – Entered as the result of an exception

Main Stack and Process Stack Access

Out of reset, all code uses the main stack with the processor in Thread mode. An exception handler such as SVC can change the stack from the main stack to the process stack by changing the EXC_RETURN value it uses on exit. All exceptions continue to use the main stack. The stack pointer (R13) is a banked register that switches between the main stack and the process stack. At any given time only one stack—the process stack or the main stack—is visible using R13.

It is also possible to switch from the main stack to the process stack while in Thread mode by writing to the special purpose Control Register using an MSR instruction.

Data Types

The processor supports the following data types:

- 32-bit words
- 16-bit halfwords
- 8-bit bytes

Note: Unless otherwise stated, the core can access all regions of the memory map, including the code region, with all data types. To support this, the system must support sub-word writes without corrupting neighboring bytes in that word (i.e., individual byte enables for writes).

Registers

The processor has the following 32-bit registers (shown in Figure 4-1):

- 13 general purpose registers, R0–R12
- Stack Pointer (SP), R13
- Link Register (LR), R14
- Program Counter (PC), R15
- Program status registers, xPSR

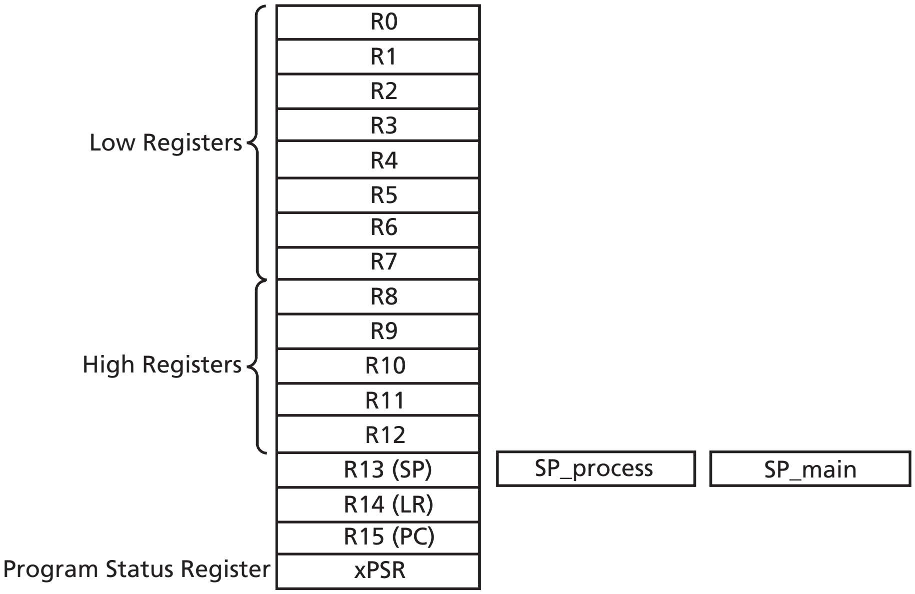


Figure 4-1 · Cortex-M1 Register Set

General Purpose Registers

The general purpose registers, R0–R12, have no architecture-specific uses.

- **Low registers** – Registers R0–R7 are accessible by all instructions that specify a general purpose register.
- **High registers** – Registers R8–R12 are accessible by some, but not all 16-bit instructions.

The R13, R14, and R15 registers have the following special functions:

- **Stack pointer** – Register R13 is used as the Stack Pointer (SP). Because the SP ignores writes to bits [1:0], it is auto-aligned to a word (four-byte) boundary. It has banked register aliases, SP_process and SP_main. Handler mode always uses SP_main, but Thread mode can be configured to use either SP_main or SP_process.
- **Link register** – Register R14 is the subroutine Link Register (LR). The LR receives the return address from the Program Counter (PC) when a Branch and Link (BL) instruction is executed. The LR is also used for exception returns. At all other times, R14 can be treated as a general purpose register.
- **Program counter** – Register R15 is the Program Counter (PC). Bit [0] is always 0, so instructions are always aligned to 16-bit halfword (two-byte) boundaries.

Special Purpose Program Status Registers (xPSR)

Processor status at the system level is broken into three categories and can be accessed as individual registers, a combination of any two of the three, or a combination of all three using the MRS and MSR instructions.

- **Application PSR (APSR)** – Contains the condition code flags. Before entering an exception, the processor saves the condition code flags on the stack. The APSR can be accessed with the MSR and MRS instructions.
- **Interrupt PSR (IPSR)** – Contains the *Interrupt Service Routine* (ISR) number of the current exception
- **Execution PSR (EPSR)** – Contains the *Thumb state bit* (T-bit). Unless the processor is in Debug state, the EPSR is not directly accessible and all fields read as zero using an MRS instruction. MSR instruction writes are ignored.

On entering an exception, the processor saves the combined information from the three status registers on the stack.

Special Purpose Priority Mask Register

Use the special purpose Priority Mask Register to boost execution priority. The special purpose Priority Mask Register can be accessed by using the MSR and MRS instructions. The CPS instruction to set or clear PRIMASK can also be accessed.

Special Purpose Control Register

The special purpose Control Register identifies the stack pointers used.

Stacks

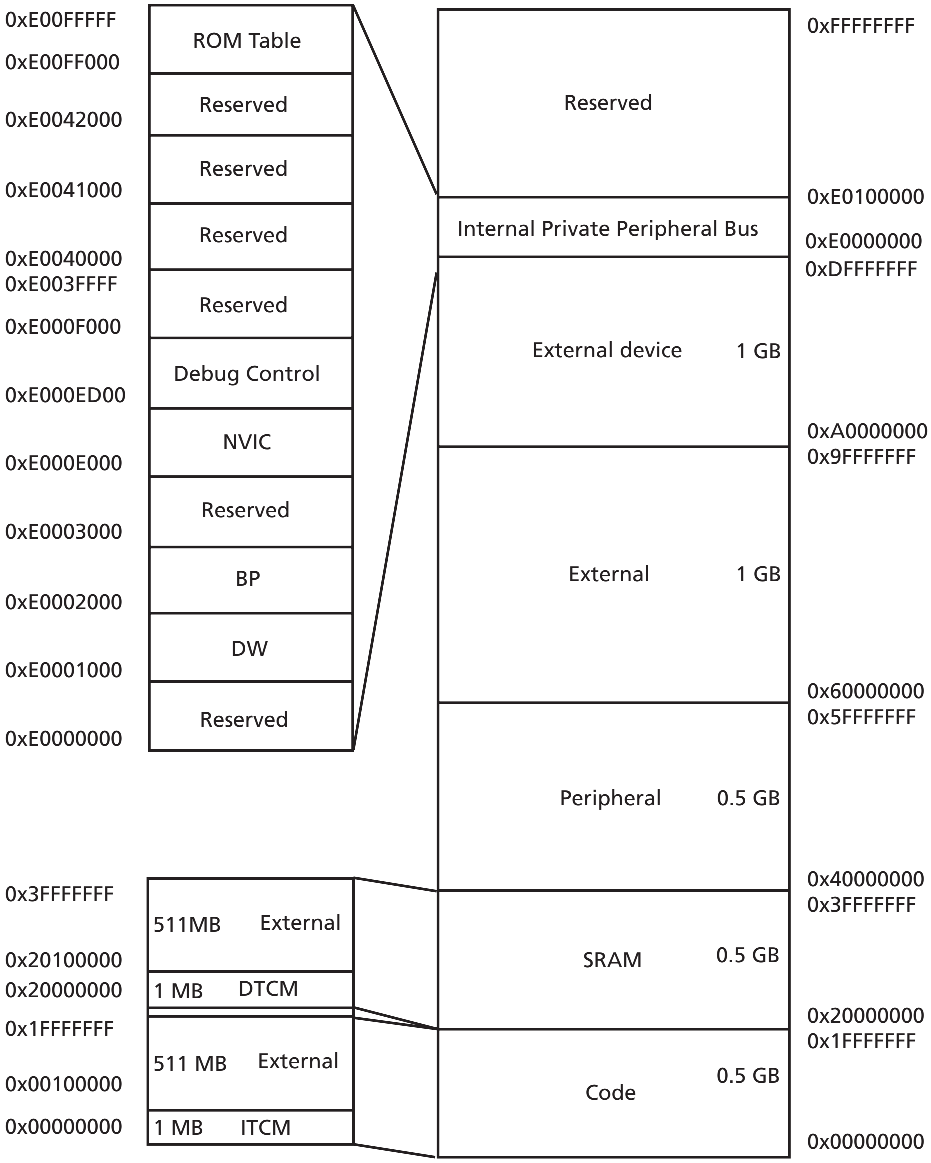
The processor supports two separate stacks:

- **Process stack** – Thread mode can be configured to use the process stack. Thread mode uses the main stack out of reset. SP_process is the *Stack Pointer* (SP) register for the process stack.
- **Main stack** – Handler mode uses the main stack. SP_main is the SP register for the main stack.

Only one stack, the process stack or the main stack, is visible at any time using R13. After pushing the content, the ISR uses the main stack and all subsequent interrupt preemptions also use the main stack.

Memory Map

Cortex-M1 has a defined memory map with the various processor interfaces addressed by different memory map regions, as shown in Figure 4-2 on page 18. The processor can access all regions within the memory map with the exception of the reserved regions. The reserved regions are Execute Never (XN) and instruction accesses are prevented by the processor hardware. The SRAM, Peripheral, External Device, and Private Peripheral Bus regions in the memory map are also XN. Instructions can be executed from the Code and External (not External Device) regions of the memory map.



Note: TCMs shown are maximum size (1,024 kbytes).

Figure 4-2 · Cortex-M1 Memory Map

The processor views memory as a linear collection of bytes numbered in ascending order from 0. For example:

- Bytes 0–3 hold the first stored word
- Bytes 4–7 hold the second stored word

The processor can access data words in memory in little-endian format or big-endian format.

In little-endian format, the byte with the lowest address in a word is the least significant byte of the word. The byte with the highest address in a word is the most significant. The byte at address 0 of the memory system connects to data lines 7–0.

In big-endian format, the byte with the lowest address in a word is the most significant byte of the word. The byte with the highest address in a word is the least significant. The byte at address 0 of the memory system connects to data lines 7–0.

Cortex-M1 always accesses code in little-endian format. Little-endian is the default memory format for ARM processors. The processor contains a configuration option that enables the user to select either the little-endian or big-endian format during implementation.

Subsystem Restrictions

The fixed memory map of the Cortex-M1 places certain restrictions on the processor subsystem.

The Code region encompasses two CoreAHB/CoreAHBLite 256 MB slots (0 and 1). If the Data region of the memory map is being mapped to a device (RAM, for example), this must be mapped to AHB slot 2 or 3.

1. If the user has internal or external RAM mapped to the Data space and is using non-zero DTCM, there will be a wasted area of external RAM. If the user wishes to use non-zero DTCM, the external SRAM should be mapped up to the second SRAM space (0x60000000 and above).
2. Similarly, if the user wishes to run from internal or external SRAM, with non-zero ITCM, this SRAM should be mapped to 0x60000000 and above. The RAM would be wasted if SRAM were remapped to location zero after boot.
3. If the user wishes to run from NVM, NVM can be left at location zero. The bottom part of NVM is copied over to the ITCM automatically by the Cortex-M1 wrapper after reset. From then on instruction fetches in the ITCM range would go to the ITCM, and instruction fetches above this space (but still within Code space) would go to NVM.

Exceptions

The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. All exceptions are handled in Handler mode. The processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the exception handler Interrupt Service Routine (ISR). The following features enable efficient, low-latency exception handling:

- Automatic state saving and restoring. The processor pushes state registers on the stack before entering the ISR, and pops them after exiting the ISR with no instruction overhead.
- Automatic reading of the vector table entry that contains the ISR address in code memory or data SRAM
- Closely-coupled interface between the processor and the NVIC to enable early processing of interrupts and processing of late-arriving interrupts with higher priority
- Configurable number of interrupts from 1 to 32
- Four levels of interrupt priority that can be software-assigned
- Separate stacks for Handler and Thread modes if OS extensions are implemented
- ISR control transfer using the calling conventions of the C/C++ standard, *Procedure Call Standard for the ARM Architecture* (PCSAA)
- Priority masking to support critical regions

Exception Types

The types of exceptions supported in Cortex-M1 are listed in Table 4-1. A fault is an exception that results from an error condition. Faults can be reported synchronously or asynchronously to the instruction that caused them. In general, faults are reported synchronously. Faults caused by writes over the bus are asynchronous faults. A synchronous fault is always reported with the instruction that caused the fault. An asynchronous fault may vary in how it is reported with respect to the instruction that caused the fault.

Table 4-1 · Cortex-M1 Exceptions

Position	Exception type	Priority	Description	Activated
–	–	–	Stack top is loaded from first entry of vector table on reset.	–
1	Reset	–3 (highest)	Invoked on power-up and warm reset. On first instruction, drops to lowest priority. Thread mode.	Asynchronous
2	Non-maskable Interrupt	–2	Cannot be marked, prevented by activation, by any other exception. Cannot be preempted by any other exception other than Reset.	Asynchronous
3	Hard Fault	–1	All classes of Fault	Synchronous or asynchronous
4–10	–	–	Reserved.	–
11	SVCall	Configurable	System service call with SVC instruction.	Synchronous
12–13	–	–	Reserved	–
14	PendSV	Configurable	Pendable request for system service. This is only pended by software.	Asynchronous
15	SysTick	Configurable	System tick timer has fired.	Asynchronous
16–48	External Interrupt	Configurable	Asserted from outside the processor, IRQ[2^{n-1}:0], and fed through the NVIC (prioritized).	Asynchronous

Exception Priority

In the processor exception model, priority determines when and how the processor handles exceptions. Software priority levels can be assigned to interrupts.

The NVIC supports software-assigned priority levels. A priority level from 0 (highest) to 3 (lowest) can be assigned to an interrupt by writing to the two-bit IP_N field in an Interrupt Priority Register. Hardware priority ranges from –3 (highest), to 3 (lowest). By default, external interrupts have a hardware priority of 3, but hardware priority decreases with increasing interrupt number. The programmable priority level overrides the hardware priority. For example, IRQ(4) would have a default priority lower than IRQ(2), but if IRQ(4) is assigned a software priority of 1 and IRQ(2) is assigned 0, then IRQ(2) has priority over IRQ(4).

Servicing an Exception

When the processor invokes an exception, it automatically pushes the following eight registers in two stages to the stack in the following order:

1. Processor Status Register (*x*PSR)
2. ReturnAddress ()
3. Link Register (LR)
4. R12
5. R3
6. R2
7. R1
8. R0

The SP is decremented by eight words on the completion of the stack push. Figure 4-3 shows the contents of the stack after an exception preempts the current program flow.

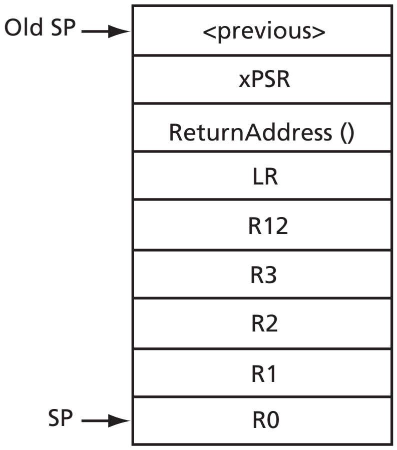


Figure 4-3 · Stack Contents from an Exception

When the processor services an exception, it takes the following steps before it enters the exception service routine.

1. It pushes 8 registers: *x*PSR, ReturnAddress (), R0, R1, R2, R3, R12, and LR on the selected stack
2. It reads the vector from the appropriate vector table entry, for example: (0x0) + (exception_number *4). This vector table read is done after all eight registers in the previous step are pushed onto the stack.
3. On Reset only, SP_main is updated from the first entry in the vector table. Other exceptions do not modify SP_main at this time and in this manner.
4. Updates PC with vector table read location. No other late-arriving exceptions can be processed until the first instruction of the exception starts to execute.
5. LR is set to EXC_RETURN to exit from the exception.

Figure 4-4 shows a timing example of an exception entry without wait states.

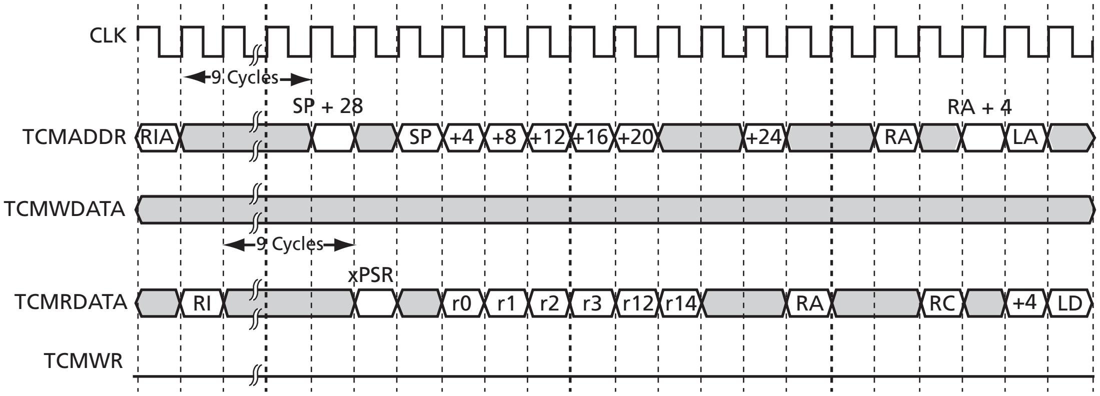


Figure 4-4 · Exception Entry Without Wait States

After returning from the exception, the processor automatically pops the eight registers from the stack. The interrupt return value, EXC_RETURN, passes as a data field in the LR, so exception functions can be normal C/C++ functions and do not require a veneer.

Clocking and Resets

The processor requires only a single functional clock input, HCLK, and one reset signal, SYSRESETn. If debug is implemented, there is also a SWJ-DP clock and nTRST. The debug reset signal DBGRESETn relates to the debug logic clocked by HCLK.

The SYSRESETn signal resets the entire processor system with the exception of debug logic in two areas:

- Nested Vectored Interrupt Controller (NVIC)
- Debug subsystem

The TCMs and the register file cannot be reset by SYSRESETn or DBGRESETn.

Reset Functionality

The top-level wrapper (CortexM1Top) handles reset generation for the processor subsystem by combining various reset sources. This block also ensures that the various resets assert asynchronously to HCLK but negate synchronously to HCLK. The reset to the Cortex-M1 (SYSRESETn) is generated by the TcmInit block.

HRESETn

HRESETn resets the AHB fabric and peripherals (except CoreWatchdog), and goes to the user top level (the top level of the CoreConsole subsystem). This signal is asserted whenever any of the following signals assert:

- NSYSRESET (external pushbutton/power-up reset)
- SYSRESETREQ (soft reset from Cortex-M1)
- WDOGRES (Watchdog reset from CoreWatchdog, if present in processor subsystem)

DBGRESETn

DBGRESETn resets the debug circuitry within Cortex-M1. This is asserted if any of the following signals assert:

- NSYSRESET (external pushbutton/power-up reset)
- nTRST (Multiplexed version of FlashPro3 TRST or RealView TRST, depending on the selected active debugger interface)

TcmInit

TcmInit controls initialization of ITCM from AHB-mapped memory external to CortexM1Top (either on- or off-chip). This block detects the negation of HRESETn. At this point, it takes ownership of the AHB-Lite bus by means of multiplexing rather than arbitration. It performs a number of 32-bit reads in AHB space, starting at location 0x0, and copies the data into ITCM. The number of reads depends on the size of the ITCM. When the initialization of ITCM is complete, the ownership of the AHB bus is passed back to Cortex-M1. At this time, TcmInit also negates SYSRESETn to Cortex-M1.

Active Debug Interface Selection

Depending on which debug interface the user requires (if any), the Debug Subsystem block multiplexes the appropriate debug signals to Cortex-M1. This block also instantiates the UJTAG macro and the uj_jtag interface block.

Buffering

The internally generated reset signals must be buffered with CLKINT buffers in this block. This is because the synthesis tools have no knowledge of the large fanout of these nets within the Cortex-M1 black box. The CLKINT buffers within the black box are removed at instantiation time by Designer. Note that there is no buffer added on HCLK in this block. The user must add a CLKINT buffer onto HCLK at the top-level.

Nested Vectored Interrupt Controller

The NVIC facilitates low-latency exception and interrupt handling and implements System Control Registers. The NVIC supports reprioritizable interrupts. The NVIC and the processor core interface are closely coupled, which enables low-latency interrupt processing and efficient processing of late arriving interrupts. The NVIC registers are listed in Table 4-2 and can only be accessed using word transfers. Any attempt to write a halfword or byte individually causes corruption of the register bits. All NVIC registers and system debug registers are little-endian regardless of the endianness state of the processor.

Table 4-2 · Cortex-M1 NVIC Registers

Name of Register	Type	Address	Reset Value
Irq 0 to 31 Set Enable Register	R/W	0xE000E100	0x00000000
Irq to 31 Clear Enable Register	R/W	0xE000E180	0x00000000
Irq to 31 Set Pending Register	R/W	0xE000E200	0x00000000
Irq to 31 Clear Pending Register	R/W	0xE000E280	0x00000000
Priority 0 Register	R/W	0xE000E400	0x00000000
Priority 1 Register	R/W	0xE000E404	0x00000000
Priority 2 Register	R/W	0xE000E408	0x00000000
Priority 3 Register	R/W	0xE000E40C	0x00000000
Priority 4 Register	R/W	0xE000E410	0x00000000
Priority 5 Register	R/W	0xE000E414	0x00000000
Priority 6 Register	R/W	0xE000E418	0x00000000
Priority 7 Register	R/W	0xE000E41C	0x00000000

The processor supports both level and pulse interrupts. A level interrupt is held asserted until it is cleared by the ISR accessing the device. A pulse interrupt is a variant of an edge model. The edge must be sampled on the rising edge of the processor clock (HCLK) instead of being asynchronous.

For level interrupts, if the signal is not deasserted before the return from the interrupt routine, the interrupt remains pending and re-activates. This is particularly useful for FIFO and buffer-based devices because it ensures that they drain either by a single ISR or by repeated invocations, with no extra work. This means that the device continues to assert the signal until the device is empty.

A pulse interrupt must be asserted for at least one HCLK cycle to enable the NVIC to latch the pending bit.

A pulse interrupt can be reasserted during the ISR so that the interrupt can be pending and active at the same time. If this occurs, the application design must ensure that a second pulse does not arrive before the first pulse is activated. The second pulse cannot set the pending bit and would have no effect, because the interrupt is already pending. However, if the interrupt is activated for at least one cycle, the NVIC latches the pending bit. After the ISR activates, the pending bit is cleared. After the bit is cleared if the interrupt is asserted again while it is activated, it can latch the pending bit again.

5

Software Development

SoftConsole is the free-of-charge Actel software development environment that allows quick turn-around for C and C++ based projects targeting Cortex-M1 and other processor-based platforms available for use in Actel devices. SoftConsole allows users to create a project and transparently manages all compilation stages in order to generate a binary file ready to be used with the Cortex-M1 processor. SoftConsole includes a fully integrated debugger that offers easy access to memory contents, registers, and single-instruction execution. Programs developed with SoftConsole can be debugged on a target board or in the tool's simulator using the same uniform interface.

SoftConsole provides a flexible and easy-to-use graphical user interface for managing your software development projects. The tool gives you the ability to quickly develop and debug software programs and to implement them in Actel devices. SoftConsole enables users to edit and debug software programs, organize files, and configure settings in a project. This tool provides simultaneous access to multiple tool windows and the ability to quickly switch editing, debug, and synchronization views.

The compilation tools can be used to build C or C++ programs. The following tools are included in SoftConsole:

- SoftConsole Eclipse-based IDE
- GCC Compiler
- GDB Debugger
- Support for program download and debug with FlashPro3

There are other tools available from ARM and third-party companies that can be purchased, including the RealView Development Suite, RealView Microcontroller Development Kit, and embedded workbench tools from IAR. The RealView and IAR tools feature compilers that offer a higher level of efficiency than the GCC compiler included in SoftConsole. If higher code density is required for an application than what can be achieved using GNU, Actel recommends that one of these other tools be purchased and used.

6

Cortex-M1 Design Entry Flow

Libero IDE automatically manages Cortex-M1 through the tool flow when it is instantiated in a CoreConsole v1.3 project system design. The project system can consist of only the Cortex-M1 itself or it can include a range of subsystem and higher-level IP blocks. The overall flow is shown in Figure 6-1.

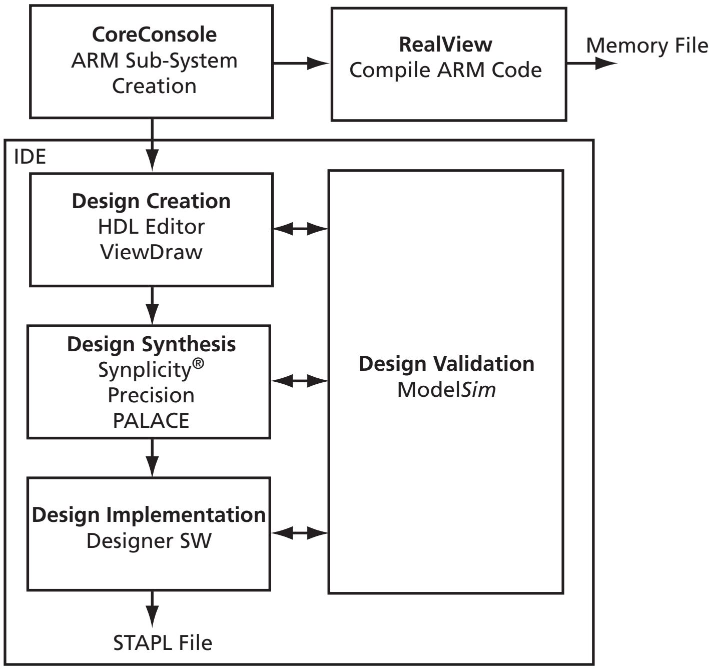


Figure 6-1 · Cortex-M1 Design Entry Flow

Design Flow if CoreConsole is Launched from Inside Libero IDE

Users will typically launch CoreConsole from Libero IDE v8.x, though CoreConsole can also be used as a standalone tool. If CoreConsole is launched from Libero IDE, the CoreConsole design is located under the Libero IDE project folder. Import of the generated files by Libero IDE is transparent to the user.

Design Flow if CoreConsole is Used as Standalone Tool

If CoreConsole is used as a standalone tool, the user must import the CoreConsole v1.3 project into Libero IDE v8.x by locating the CXF file for that project in the CoreConsole Libero IDE Export folder hierarchy. CoreConsole exports the generated files by default into the:

- *CoreConsole\LiberoExport\<my_coreconsole_project>*
- Where *CoreConsole* is the root folder of the CoreConsole installation, and <*my_coreconsole_project*> is the name of the project being generated. These files can be examined by navigating to the folder on your computer.
- When you install CoreConsole, you are given the option of selecting the name and location of the *CoreConsole* designs folder, in which case the files are output in <your_selected_folder>*\LiberoExport\<my_coreconsole_project>*.

The *LiberoExport* folder contains the full list of projects. Within each project folder is a common folder and project folder, as shown in Figure 6-2.

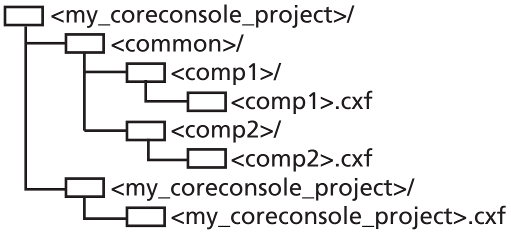


Figure 6-2 · CoreConsole\Libero Export Folder File Structure

The *<my_coreconsole_project>.cxf* file is an XML file that describes the layout and content of the design and is used by Libero IDE to import the CoreConsole project. Within the file set for each component there is a *<comp_n>.cxf* XML file, which describes the sub-component of the design.

The *my_coresonsole_project/my_coreconsole_project/my_coreconsole_project.cxf* file must be imported into Libero IDE. Libero IDE copies the *my_coreconsole_project* folder and locates all the files in the appropriate locations in the Libero IDE workspace. Refer to the *Actel Libero IDE User's Guide* for more information on importing a CoreConsole project.

The Cortex-M1 files associated with this flow are shown in Figure 6-3. Normally the CoreConsole system project in Libero IDE consists of a range of RTL IP components alongside Cortex-M1. These components are implemented in a secure format that can only be read and used within the tools utilized by Libero IDE.

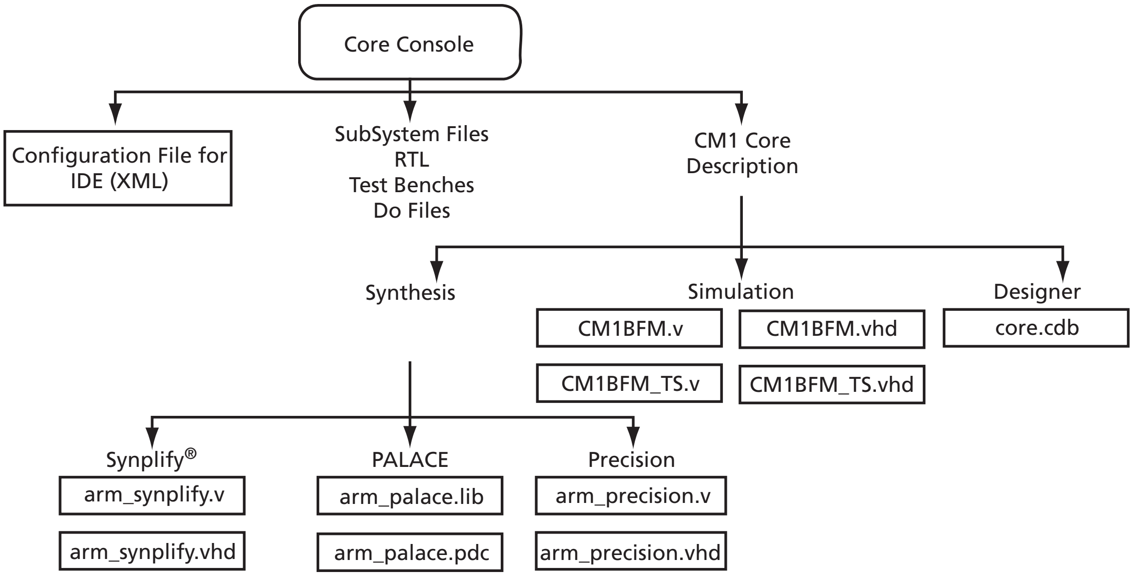


Figure 6-3 · Cortex-M1 File Flow

Libero IDE manages all the files imported from CoreConsole and passes them to the appropriate tools that it utilizes. The secure Cortex-M1 implementation is represented inside the IDE as CM1 when the RTL wrapper has been removed.

Cortex-M1 Security

Cortex-M1 is delivered as a CDB file that the tools in Libero IDE can use to instantiate the processor core in a design. Similar to the earlier CoreMP7 ARM7 processor, the core is delivered in a black box that allows users to access the top-level I/O and use the core in Actel M1 devices, but not view the contents of the black box. The Cortex-M1 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 Cortex-M1 core. The placement and routing of the Cortex-M1 within the FPGA fabric core is fixed. Libero IDE is aware of the core's placement and will fill in the unused tiles around the core with the IP blocks that users put together around Cortex-M1 in their CoreConsole project system. The Cortex-M1 black box cannot be unlocked and can only be programmed into an Actel M1 ProASIC3/E, M1 Fusion, or M1 IGLOO device.

7

Bus Functional Model (BFM)

During the development of an FPGA-based SoC, various 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 or VHDL form, or using a tool such as Seamless

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

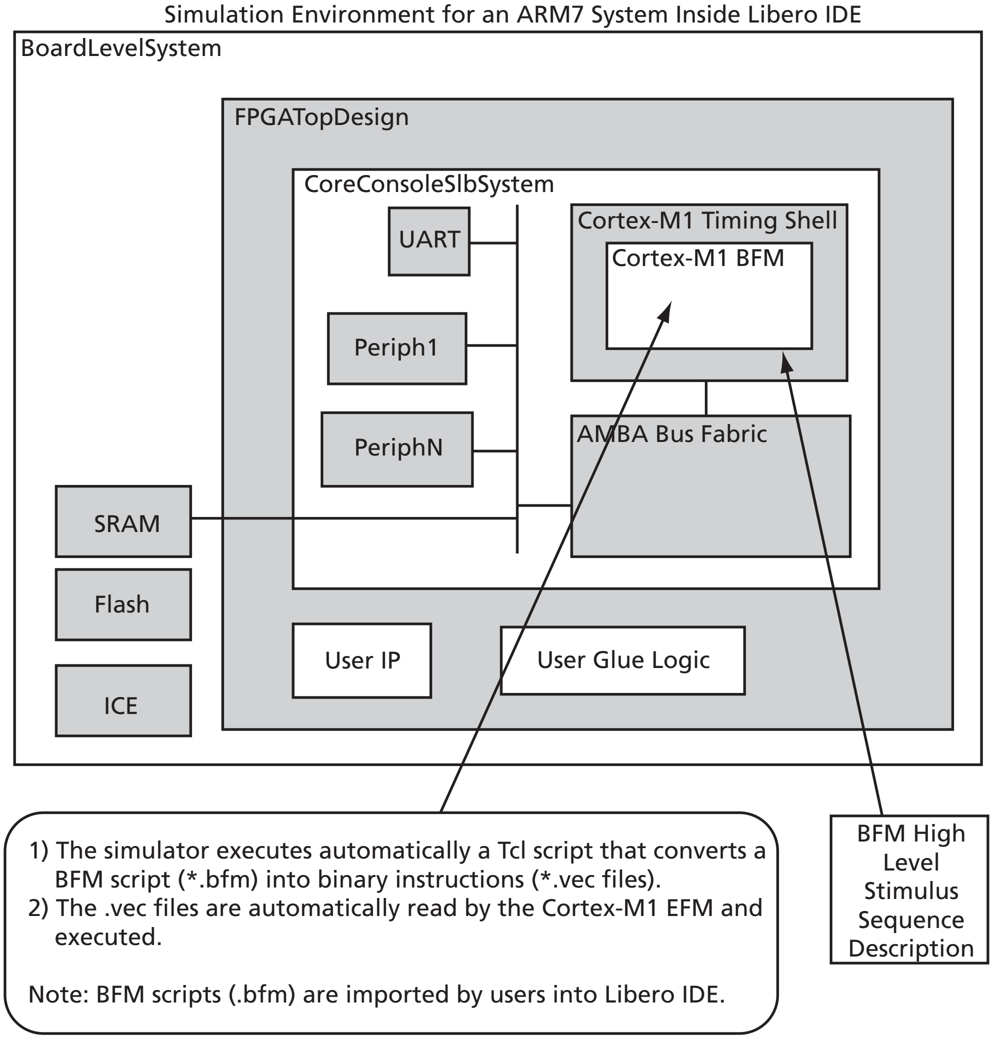
The CoreConsole IP Development Platform (IDP) provides a means for stitching together IP blocks using a bus fabric of choice. It generates a system testbench, controlled by a script-driven BFM of the Cortex-M1 processor. The BFM allows the developer to model bus transactions. This approach allows verification of connectivity of the various IP blocks and the system memory map presented to Cortex-M1 by the rest of the hardware.

This section describes the following aspects of Cortex-M1 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 7-1 shows the various components within a typical system-level testbench.



1) The simulator executes automatically a Tcl script that converts a BFM script (*.bfm) into binary instructions (*.vec files).
2) The .vec files are automatically read by the Cortex-M1 EFM and executed.

Note: BFM scripts (.bfm) are imported by users into Libero IDE.

Figure 7-1 · BFM Simulation Environment

In Figure 7-1 it is assumed that the processor subsystem has been specified by selecting the processor, bus fabric, IP blocks, and the memory system using CoreConsole.

When building the system, the memory map of the system must be specified. Based on this information, CoreConsole generates the following outputs:

- Verilog/VHDL model of SoC subsystem
- Verilog/VHDL models of IP cores
- Cortex-M1 BFM
- BFM test script
- System-level skeleton testbench

The BFM acts as a pin-for-pin replacement for the Cortex-M1 in the project system. It initiates cycle-accurate bus transactions on the native Cortex-M1 bus. It has no knowledge, however, of real Cortex-M1 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 (from SPIRIT IP-XACT 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, and write-only. From this it can decide what the expected data should be on reads.

The system Verilog/VHDL can be edited to add new design blocks in the above diagram. The system-level testbench can be edited to include tasks that test any newly added functionality, or for adding 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, stimuli can be provided to the system from the inside (via the Cortex-M1 BFM), as well as from the outside (via testbench tasks).

Functionality

This section describes the specific functionality of the Cortex-M1 BFM. The BFM models the Cortex-M1 external bus. Specifically, it models the following bus signals:

- ADDR – address bus
- WDATA – write data bus
- RDATA – read data bus
- WRITE – indicates write access
- CLKEN – clock enable

The BFM also models the following control signals:

- CLK – clock
- SIZE – memory access width

Cortex-M1 Pin Compatibility

The BFM model is pin-for-pin compatible with the Cortex-M1. This allows the model to be dropped into the space that would be occupied by the processor core in the system testbench.

Cortex-M1 Bus Cycle Accuracy

The bus cycle timings for the Cortex-M1 external bus signals are specified in the *Cortex-M1 Technical Reference Manual*. The Cortex-M1 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 (see "BFM Script Language" on page 34). This allows initiating writes to system resources, reads from system resources (with or without checking of expected data), and waiting for interrupt events.

Self-Checking

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

Endianness

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 the Cortex-M1 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 can be easily identified.

BFM Script Language

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

Write

The write 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

The read 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

The readcheck 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

A timing shell is provided for each Cortex-M1 variant wrapped around the BFM itself. The BFM is bus cycle accurate and performs setup/hold checks to model output propagation delays.

Example BFM Use Case

This section goes through an example use case of the Cortex-M1 BFM. In this system, the developer requires two additional Actel IP cores: the Core10/100 and the CoreUART.

SPIRIT IP-XACT 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 IP-XACT specification in XML. For example, in the case of the CoreUART, the attributes would indicate that there are three registers, as shown in Table 7-1.

Table 7-1 · SPIRIT IP-XACT Attributes

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 will not be any self-checking that can be performed for the UART. Nevertheless, the bus transactions do take place and the cycles can 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 similar to 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 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 a new block could be added.

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

When running the system simulation, 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
```

8

Debug

The ARM Debug Architecture uses a protocol converter box to allow the debugger to talk via a JTAG port directly to the core. The scan chains in the core that are required for test are re-used for debugging.

About Debug

Debug facilitates the following:

- Core halt
- Core stepping
- Core register access
- Read/Write to TCMs
- Read/Write to AHB address space
- Breakpoints
- Watchpoints

The main debug components are as follows:

- Debug control registers to access and control debugging of the core
- BreakPoint Unit (BPU) to implement breakpoints
- Data Watchpoint (DW) unit to implement watchpoints and trigger resources
- Debug memory interfaces to access external ITCM and DTCM
- ROM table

All debug components exist on the internal Private Peripheral Bus (PPB), 0xE000ED30 to 0xE000EEFF. Access to the debug components, debug control, and configuration are only available when the debug extension is present. You can never access the debug components from the processor, even when debug is present.

Debug control and data access occur through the Advanced High-Performance Bus-Access Port (AHB-AP).

Access includes the following:

- The AHB-PPB. Through this bus, the debugger can access debug, including the following:
    - debug control
    - DW unit
    - BPU unit
    - The ROM Table
    - TCMs, if configured
- The AHB address space. The AHB slaves in the debug system always expect 32-bit AHB transfers. If a byte or halfword access is created from the DAP, the transfer is extended to a 32-bit access and all 32 bits in the register are accessed. Figure 1-1 on page 7 shows the structure of the debug system, indicating how the AHB-AP can access each of the system components and external buses.

JTAG Debug Interface

The Actel FlashPro3 programmer, which is used to program the FPGA and debug the Cortex-M1 core using SoftConsole, uses a standard 10-pin JTAG interface, shown in Figure 8-1. Other debuggers, including those in the RealView and IAR tools, use a 20-pin, 2.54 mm pitch IDC connector (Figure 8-2 on page 39). The cable can be used to mate with a keyed box header on the target.

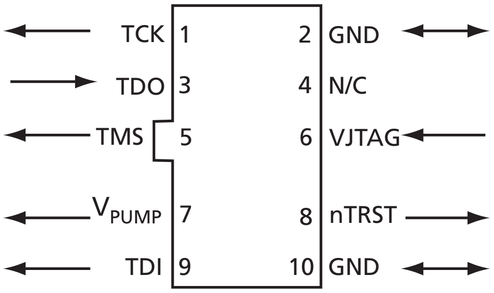


Figure 8-1 · JTAG 10-Pin Connector Pinout

The signals on the 10-pin JTAG interface are shown in Table 8-1.

Table 8-1 · JTAG 10-Pin Connector Signals

Signal	Description
V_{PUMP}	3.3 V Programming voltage
GND	Signal reference
TCK	JTAG clock
TDI	JTAG data input to device
TDO	JTAG data output from device
TMS	JTAG mode select
nTRST	Programmable output pin may be set to Off, Toggle, LOW, or HIGH Level
VJTAG	Reference voltage from the target board
N/C	Programmer does not connect to this pin

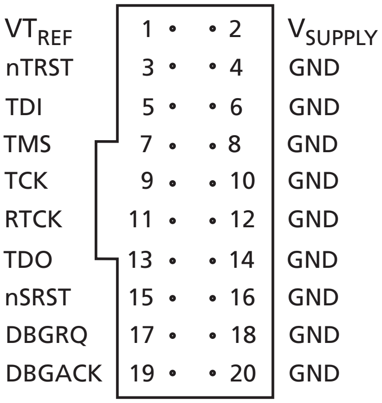


Figure 8-2 · JTAG 20-Pin Connector Pinout

The signals on the 20-pin JTAG interface are shown in Table 8-2.

Table 8-2 · JTAG 20-Pin Connector 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. Actel recommends pulling this signal 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. Actel recommends pulling this signal 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, a returned, and retimed, TCK can be used to dynamically control the TCK rate. RealView ICE provides Adaptive Clock Timing, which 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. Actel recommends pulling this pin LOW on the target.

Table 8-2 · JTAG 20-Pin Connector Signals (continued)

Signal	I/O	Description
TDI	Output	Test Data In signal from RealView ICE to the target JTAG port. Actel recommends pulling this pin HIGH on the target.
TDO	Input	Test Data Out from the target JTAG port to RealView ICE. Actel recommends pulling this pin 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 to avoid adverse effects from any spurious TCKs when there is no connection.
V_{SUPPLY}	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.
VT_{REF}	Input	This is the target reference voltage. It indicates that the target has power, and It must be at least 0.628 V. VT_{REF} is normally fed from V_{DD} on the target hardware and might have a series resistor (though this is not recommended). There is a 10 kB pull-down resistor on VT_{REF} in RealView ICE.

9

AC Parameters

This section gives the AC timing parameters of the Cortex-M1 processor. The AC parameters are valid for v2.1 of Cortex-M1, operating under worst-case commercial conditions. In the following tables, all times are in nanoseconds. The parameter definitions are as follows:

- PERIOD is the minimum supported period of the clock signal specified.
- SETUP is the minimum time for which the specified input signal must be valid before the rising edge of the specified clock.
- HOLD is the minimum time for which the specified input signal must remain valid after the rising edge of the specified clock.
- CLOCK2OUT is the maximum propagation delay from the rising edge of the specified clock to the specified output signal to be guaranteed valid.

The AC parameters for the M1A3PE1500-2 debug and no-debug variants and for the M1AGL600V2 variant are not currently available.

The AC parameters for the M1AFS600-2 no-debug variant are shown in Table 9-1.

Table 9-1 · M1AFS600-2 No-Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	15.1130
SETUP	HCLK	HRDATA	2.0500
SETUP	HCLK	HREADY	5.2020
SETUP	HCLK	HRESP	5.0370
SETUP	HCLK	IRQ	4.3600
SETUP	HCLK	NMI	2.7390
HOLD	HCLK	HRDATA	-0.4100
HOLD	HCLK	HREADY	-1.1900
HOLD	HCLK	HRESP	-1.5940
HOLD	HCLK	IRQ	-1.1470
HOLD	HCLK	NMI	-0.3470
CLOCK2OUT	HCLK	HADDR	3.7550
CLOCK2OUT	HCLK	HPROT	2.2590
CLOCK2OUT	HCLK	HSIZE	1.1840
CLOCK2OUT	HCLK	HTRANS	4.6670
CLOCK2OUT	HCLK	HWDATA	2.9390
CLOCK2OUT	HCLK	HWRITE	2.6680
CLOCK2OUT	HCLK	LOCKUP	2.6980
CLOCK2OUT	HCLK	SYSRESETREQ	2.0440

The AC parameters for the M1AFS600-2 debug variant are shown in Table 9-2.

Table 9-2 · M1AFS600-2 Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	16.6360
PERIOD	SWCLKTCK	SWCLKTCK	16.8000
SETUP	HCLK	EDBGRQ	1.7220
SETUP	HCLK	HRDATA	4.9590
SETUP	HCLK	HREADY	7.4990
SETUP	HCLK	HRESP	6.7370
SETUP	HCLK	IRQ	4.8180
SETUP	HCLK	NMI	3.9330
SETUP	SWCLKTCK	SWDITMS	9.4990
SETUP	SWCLKTCK	TDI	2.2370
HOLD	HCLK	EDBGRQ	-0.8270
HOLD	HCLK	HRDATA	-0.5070
HOLD	HCLK	HREADY	-1.5070
HOLD	HCLK	HRESP	-1.7960
HOLD	HCLK	IRQ	-0.9510
HOLD	HCLK	NMI	-0.5590
HOLD	SWCLKTCK	SWDITMS	1.1310
HOLD	SWCLKTCK	TDI	0.0800
CLOCK2OUT	HCLK	HADDR	6.5710
CLOCK2OUT	HCLK	HALTED	3.3850
CLOCK2OUT	HCLK	HBURST	10.6990
CLOCK2OUT	HCLK	HMASTLOCK	6.3900
CLOCK2OUT	HCLK	HPROT	5.6830
CLOCK2OUT	HCLK	HSIZE	5.8770
CLOCK2OUT	HCLK	HTRANS	8.0970
CLOCK2OUT	HCLK	HWDATA	3.6190
CLOCK2OUT	HCLK	HWRITE	5.8240
CLOCK2OUT	HCLK	LOCKUP	3.0830
CLOCK2OUT	HCLK	SYSRESETREQ	1.8380

Table 9-2 · M1AFS600-2 Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
CLOCK2OUT	SWCLKTCK	JTAGNSW	8.2940
CLOCK2OUT	SWCLKTCK	JTAGTOP	8.3280

The AC parameters for the M1AGL600V2 no-debug variant are shown in Table 9-3.

Table 9-3 · M1AGL600V2 No-Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	39.7330
SETUP	HCLK	HRDATA	4.6420
SETUP	HCLK	HREADY	15.4690
SETUP	HCLK	HRESP	14.2470
SETUP	HCLK	IRQ	11.4390
SETUP	HCLK	NMI	6.9170
HOLD	HCLK	HRDATA	-0.5820
HOLD	HCLK	HREADY	-3.0730
HOLD	HCLK	HRESP	-2.9090
HOLD	HCLK	IRQ	-1.6650
HOLD	HCLK	NMI	-0.4310
CLOCK2OUT	HCLK	HADDR	8.7220
CLOCK2OUT	HCLK	HPROT	6.3690
CLOCK2OUT	HCLK	HSIZE	2.9820
CLOCK2OUT	HCLK	HTRANS	13.5980
CLOCK2OUT	HCLK	HWDATA	7.8890
CLOCK2OUT	HCLK	HWRITE	6.8070
CLOCK2OUT	HCLK	LOCKUP	6.1980
CLOCK2OUT	HCLK	SYSRESETREQ	4.6370

The AC parameters for the M1AGL600V5 no-debug variant are shown in Table 9-4.

Table 9-4 · M1AGL600V5 No-Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	24.4350
SETUP	HCLK	HRDATA	2.9240

Table 9-4 · M1AGL600V5 No-Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
SETUP	HCLK	HREADY	10.0800
SETUP	HCLK	HRESP	9.6380
SETUP	HCLK	IRQ	7.6760
SETUP	HCLK	NMI	5.1000
HOLD	HCLK	HRDATA	-0.3990
HOLD	HCLK	HREADY	-1.6540
HOLD	HCLK	HRESP	-1.6940
HOLD	HCLK	IRQ	-2.0610
HOLD	HCLK	NMI	-0.5600
SETUP	HCLK	SYSRESETn	0.2690
HOLD	HCLK	SYSRESETn	0.1220
CLOCK2OUT	HCLK	HADDR	6.1480
CLOCK2OUT	HCLK	HPROT	3.6660
CLOCK2OUT	HCLK	HSIZE	1.8720
CLOCK2OUT	HCLK	HTRANS	8.7930
CLOCK2OUT	HCLK	HWDATA	4.7220
CLOCK2OUT	HCLK	HWRITE	4.2140
CLOCK2OUT	HCLK	LOCKUP	4.0650
CLOCK2OUT	HCLK	SYSRESETREQ	2.7150

The AC parameters for the M1AGL600V5 debug variant are shown in Table 9-5.

Table 9-5 · M1AGL600V5 Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	24.5140
PERIOD	SWCLKTCK	SWCLKTCK	25.6090
SETUP	HCLK	EDBGRQ	4.9360
SETUP	HCLK	HRDATA	7.2440
SETUP	HCLK	HREADY	13.2560
SETUP	HCLK	HRESP	12.5270
SETUP	HCLK	IRQ	7.8900
SETUP	HCLK	NMI	6.6350

Table 9-5 · M1AGL600V5 Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
SETUP	SWCLKTCK	SWDITMS	13.4690
SETUP	SWCLKTCK	TDI	3.9710
HOLD	HCLK	EDBGRQ	-2.8490
HOLD	HCLK	HRDATA	-0.4890
HOLD	HCLK	HREADY	-1.2810
HOLD	HCLK	HRESP	-2.6590
HOLD	HCLK	IRQ	-1.6540
HOLD	HCLK	NMI	-1.5700
HOLD	SWCLKTCK	SWDITMS	1.6200
HOLD	SWCLKTCK	TDI	-1.3280
CLOCK2OUT	HCLK	HADDR	11.8030
CLOCK2OUT	HCLK	HALTED	4.8150
CLOCK2OUT	HCLK	HBURST	18.7390
CLOCK2OUT	HCLK	HMASTLOCK	8.5370
CLOCK2OUT	HCLK	HPROT	9.0570
CLOCK2OUT	HCLK	HSIZE	11.1280
CLOCK2OUT	HCLK	HTRANS	14.5010
CLOCK2OUT	HCLK	HWDATA	6.1780
CLOCK2OUT	HCLK	HWRITE	9.9420
CLOCK2OUT	HCLK	LOCKUP	6.3600
CLOCK2OUT	HCLK	SYSRESETREQ	2.7230
CLOCK2OUT	SWCLKTCK	JTAGNSW	12.5920
CLOCK2OUT	SWCLKTCK	JTAGTOP	5.8430
CLOCK2OUT	SWCLKTCK	SWDO	5.6440
CLOCK2OUT	SWCLKTCK	SWDOEN	7.5200
CLOCK2OUT	SWCLKTCK	TDO	1.9090
CLOCK2OUT	SWCLKTCK	nTDOEN	1.9170

The AC parameters for the M1A3P1000-2 no-debug variant are shown in Table 9-6.

Table 9-6 · M1A3P1000-2 No-Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	14.5070
SETUP	HCLK	HRDATA	1.8950
SETUP	HCLK	HREADY	5.8290
SETUP	HCLK	HRESP	5.1530
SETUP	HCLK	IRQ	4.3270
SETUP	HCLK	NMI	2.6750
HOLD	HCLK	HRDATA	-0.0980
HOLD	HCLK	HREADY	-0.8520
HOLD	HCLK	HRESP	-0.8590
HOLD	HCLK	IRQ	-0.5100
HOLD	HCLK	NMI	-0.1160
SETUP	HCLK	SYSRESETn	0.2580
HOLD	HCLK	SYSRESETn	0.1990
CLOCK2OUT	HCLK	HADDR	3.3990
CLOCK2OUT	HCLK	HPROT	2.7360
CLOCK2OUT	HCLK	HSIZE	1.2320
CLOCK2OUT	HCLK	HTRANS	4.9130
CLOCK2OUT	HCLK	HWDATA	3.2550
CLOCK2OUT	HCLK	HWRITE	2.3970
CLOCK2OUT	HCLK	LOCKUP	2.6550
CLOCK2OUT	HCLK	SYSRESETREQ	1.9820

The AC parameters for the M1A3P1000-2 debug variant are shown in Table 9-7.

Table 9-7 · M1A3P1000-2 Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
PERIOD	HCLK	HCLK	16.2220
PERIOD	SWCLKTCK	SWCLKTCK	17.5160
SETUP	HCLK	EDBGRQ	2.4600
SETUP	HCLK	HRDATA	4.8700
SETUP	HCLK	HREADY	7.0950

Table 9-7 · M1A3P1000-2 Debug Variant AC Parameters

Parameter	Clock	Signal	Timing
SETUP	HCLK	HRESP	6.8410
SETUP	HCLK	IRQ	5.4760
SETUP	HCLK	NMI	3.3380
SETUP	SWCLKTCK	SWDITMS	12.9760
SETUP	SWCLKTCK	TDI	1.2530
HOLD	HCLK	EDBGRQ	-0.8250
HOLD	HCLK	HRDATA	-0.3580
HOLD	HCLK	HREADY	-0.6230
HOLD	HCLK	HRESP	-1.4070
HOLD	HCLK	IRQ	-0.7920
HOLD	HCLK	NMI	-0.1650
HOLD	SWCLKTCK	SWDITMS	1.0430
HOLD	SWCLKTCK	TDI	0.0620
CLOCK2OUT	HCLK	HADDR	6.0660
CLOCK2OUT	HCLK	HALTED	4.4350
CLOCK2OUT	HCLK	HBURST	11.4210
CLOCK2OUT	HCLK	HMASTLOCK	5.4120
CLOCK2OUT	HCLK	HPROT	5.4390
CLOCK2OUT	HCLK	HSIZE	5.9720
CLOCK2OUT	HCLK	HTRANS	8.1540
CLOCK2OUT	HCLK	HWDATA	4.1110
CLOCK2OUT	HCLK	HWRITE	5.7720
CLOCK2OUT	HCLK	LOCKUP	3.7420
CLOCK2OUT	HCLK	SYSRESETREQ	2.4990
CLOCK2OUT	SWCLKTCK	JTAGNSW	7.9280
CLOCK2OUT	SWCLKTCK	JTAGTOP	6.1220
CLOCK2OUT	SWCLKTCK	SWDO	1.0190
CLOCK2OUT	SWCLKTCK	SWDOEN	7.4190
CLOCK2OUT	SWCLKTCK	TDO	2.4600
CLOCK2OUT	SWCLKTCK	nTDOEN	2.2310

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 the Actel 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. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom

Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

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