ARM® Cortex™-M1

DirectCore

Product Summary

Key Features

- Designed Specifically for Implementation in FPGAs
- 32-Bit RISC Architecture (ARMv6-M)
- 32-Bit AHB-Lite Bus Interface
- 3-Stage Pipeline
- 32-Bit ALU
- 32-Bit Memory Addressing Range
- Embedded ICE-RT Real-Time Debug Unit
- JTAG Interface Unit

Intended Use

- MicroTCA
- System Management
- Avionics
- Robotics
- Medical Equipment
- Automotive Infotainment
- Personal Media Players
- Wireless Handsets
- Digital Still Cameras

Benefits

- Fully Implemented in FPGA Fabric
- User Can Access All Core Signals and I/Os
- User Can Program into FPGA
- No License Fees or Royalties
- Can Run All Existing Thumb® Code
- Upward Compatible with Cortex-M3

Supported Families

- IGLOO® (M1AGL)
- IGLOOe (M1AGLE)
- ProASIC®3L (M1A3PXXXXL)
- ProASIC3 (M1A3P)
- ProASIC3E (M1A3PE)
- Fusion (M1AFS)

Synthesis and Simulation support

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

Verification and Compliance

- Compliant with ARMv6-M (ARM Cortex-M1) Instruction Set Architecture (ISA)

Introduction

The Cortex-M1 soft IP core is a member of the ARM Cortex family of processors and has been optimized for use in Actel ARM-enabled FPGAs. Refer to the *ARM Cortex-M1 Handbook* for detailed information on the Cortex-M1.

The ARM Cortex-M1 is supplied with an AMBA AHB-Lite interface for inclusion in an AMBA-based processor system such as the one generated by the Actel CoreConsole IP deployment platform.

Cortex-M1 Processor

ARM Cortex-M1 is a general purpose, 32-bit microprocessor that offers high performance and small size in FPGAs. ARM Cortex-M1 runs a subset of the Thumb-2 instruction set (ARMv6-M), which includes all base 16-bit Thumb instructions and a few Thumb-2 32-bit instructions (BL, MRS, MSR, ISB, DSB, and DMB). This enables very tight and efficient code to be written for the processor that is ideal for the limited memory typically found in embedded applications.

The main blocks in ARM Cortex-M1 are 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 the Link Register (LR), Program Counter (PC), Program Status Register (xPSR), and two banked Stack Pointers (SP).

Figure 1 shows an ARM Cortex-M1 processor with debug block diagram.

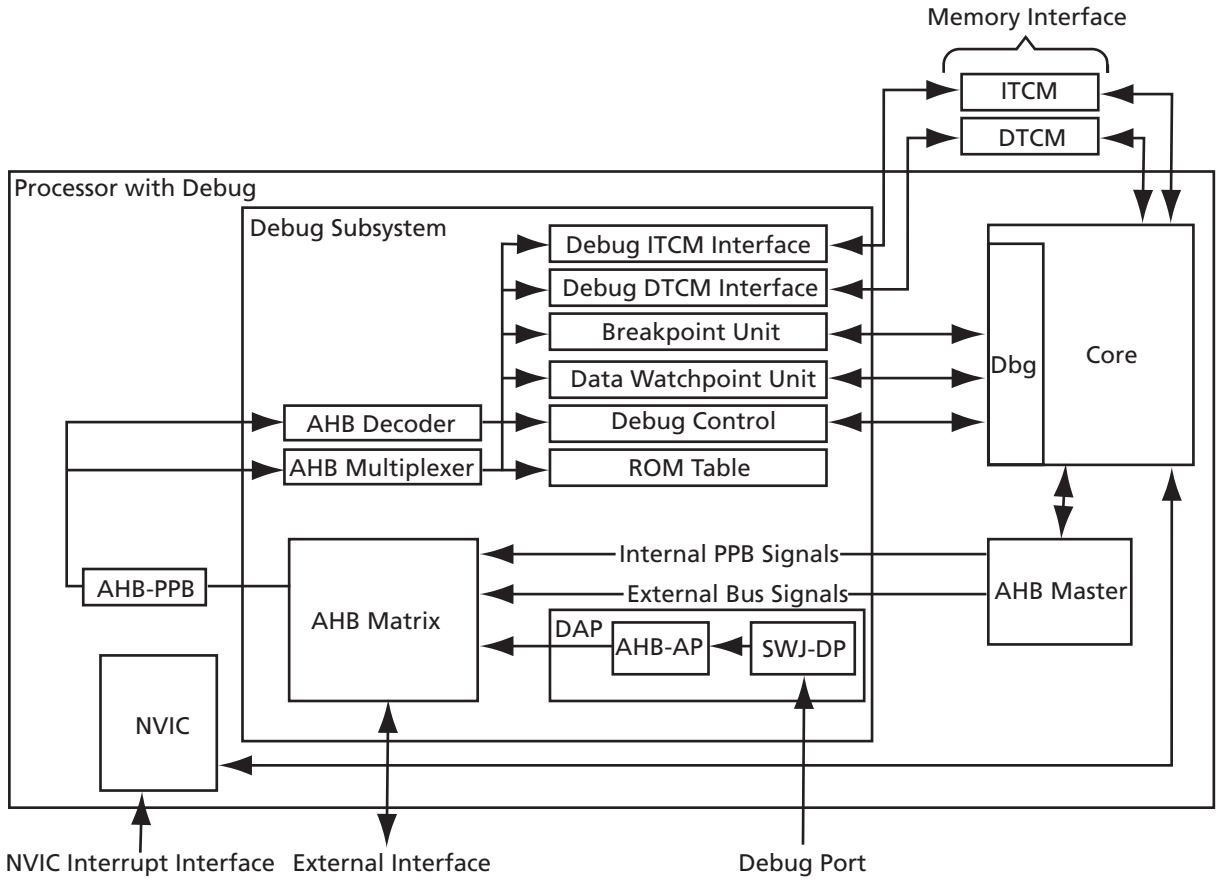

Figure 1 • **Processor with Debug Block Diagram**

The NVIC is closely coupled to the ARM Cortex-M1 core to achieve low-latency interrupt processing. The versions currently available for use in M1 devices support 1 interrupt with 4 levels of priority. Future versions will support up to 32 interrupts. To simplify software development, the processor state is automatically saved on interrupt entry, and restored on interrupt exist, with no instruction overhead.

The ARM Cortex-M1 Thumb instruction set's 16-bit instruction length allows it to approach twice the density in memory of standard 32-bit ARM code while retaining most of the ARM performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because Thumb code operates on the 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.

ARM Cortex-M1–Enabled FPGAs

ARM Cortex-M1 is available for use in a growing number of Actel M1 devices. These include M1AFS600, M1AGL600, M1A3P1000, and a number of additional M1 devices. The devices listed have the features shown in the following sections:

M1AFS600 Fusion

Actel Fusion™ Programmable System Chips (PSCs) are the world's first mixed-signal FPGAs. Fusion integrates a 12-bit analog-to-digital converter, as many as 40 analog I/Os, up to 8 Mbits of flash memory, and FPGA fabric all in a single device. When used in conjunction with a soft processor such as ARM Cortex-M1, Actel Fusion devices represent the definitive soft MCU platform.

M1AFS600 Features

- Industry's first mixed-signal FPGA
- 600,000 system gates – 13,824 logic tiles
- ARM Cortex-M1 uses less than 30% of FPGA logic
- 4 Mbits flash, 108 kbits SRAM
- 30 analog inputs, 10 analog outputs
- 172 digital I/Os
- 2 PLLs, 1% RC oscillator, Xtal oscillator, RTC

M1AGL600 IGLOO FPGAs

The M1 IGLOO devices are reprogrammable, full-featured flash FPGAs designed to meet the demanding power and area requirements of today's portable electronics. Featuring Flash*Freeze™ technology and with operating voltages of 1.2 V / 1.5 V, these devices offer the industry's lowest power consumption. M1 IGLOO devices give designers a flexible system construction platform for building portable products that offer maximum battery life.

M1AGL600 Features

- Ultra-low power flash-based FPGA
- 600,000 system gates – 13,824 logic tiles
- ARM Cortex-M1 uses less than 33% of FPGA logic
- 144 kbits SRAM, 235 digital I/Os

M1A3P1000 ProASIC3/E FPGAs

ProASIC3/E devices, the third generation of Actel Flash FPGAs, offer industry-leading unit cost and lowest total system cost—up to 504 kbits of SRAM, 350 MHz operation, and best-in-class logic utilization. These single-chip FPGAs require no boot ROMs or other support chips and are highly secure, with 128-bit AES encryption and FlashLock® technology.

M1A3P1000 Features

- Low-cost flash-based FPGA
- 1,000,000 system gates – 24,576 logic tiles
- ARM Cortex-M1 uses less than 20% of FPGA logic
- 144 kbits SRAM
- 300 digital I/Os

Table 1 • **ARM Cortex-M1 Utilization Data**

Device Variant	Size (tiles)	Size (tiles) RVDS	Size (tiles) FlashPro	RAM Blocks	Device	Utilization (%)
M1 Fusion						
No debug	4,452	–	–	4	M1AFS600	31
With debug	–	9,272	9,462	4	M1AFS600	68
M1 IGLOO 1.5 V						
No debug	4,435	–	–	4	M1AGL600	31
With debug	–	9,250	9,440	4	M1AGL600	68
M1 IGLOO 1.2 V						
No debug	4,435	–	–	4	M1AGL600	31
With debug	–	9,250	9,440	4	M1AGL600	68
M1 ProASIC3/E						
No debug	4,435	–	–	4	M1A3PE1500	12
With debug	–	9,250	9,440	4	M1A3PE1500	25

Note: *Configuration is 0 kbytes ITCM, 0 kbytes DTCM, small multiplier, 1interrupt, no OS extensions, little-endian, and debug as shown. Refer to the Cortex-M1 Handbook for performance information.*

ARM Cortex-M1 Signals

The signals of the core are given in Table 2.

Table 2 • **ARM 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 ARM 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
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

Programmer's Model

The ARM Cortex-M1 processor supports all ARMv6-M Thumb instructions. This includes the entire 16-bit Thumb instruction set architecture and some 32-bit instructions. or information on ARMv6-M Thumb instructions, see the *ARMv6-M Architecture Reference Manual*. The processor does not support ARM instructions.

Processor Operating States

The ARM Cortex-M1 processor has two operating states:

- **Thumb state** – This is the normal execution state, with the processor running the 16-bit and 32-bit halfword-aligned Thumb and Thumb-2 BL, MRS, MSR, ISB, DSB, and DMB instructions.
- **Debug state** – This is the state the processor is in for debugging.

Processor Operating Modes

The ARM Cortex-M1 processor supports two modes of operation:

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

Main Stack and Process Stack Access

Out of reset, all code uses the main stack. An exception handler such as SVC can change the stack used by Thread mode from main stack to 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. Only one stack, either the process stack or the main stack, is visible, using R13, at any time.

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

Registers

The processor has the following 32-bit registers (Figure 2):

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

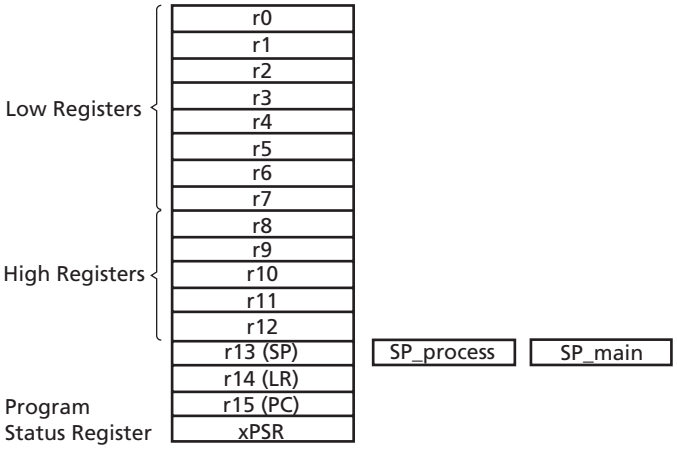

Figure 2 • **Processor Register Set**

General Purpose Registers

The general purpose registers, R0–R12, have no special architecturally-defined uses.

- **Low registers** – Registers R0–R7 are accessible by all instructions that specify a general purpose register.
- **High registers** – Registers R8-R12 are not accessible by 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, and boundary. Handler mode always uses SP_main, but you can configure Thread mode 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 PC when a Branch and Link (BL) instruction is executed. The LR is also used for an exception return. At all other times, you can treat R14 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 halfword boundaries.

Special Purpose Program Status Registers (xPSR)

Processor status at the system level breaks down into three categories and can be accessed as individual registers, a combination of any two from 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. You can access the APSR with the MSR and MRS instructions.

- **Interrupt PSR** (IPSR) – Contains the Interrupt Service Routine (ISR) number of the current exception activation.
- **Execution PSR** (EPSR) – Contains the Thumb state bit (T-bit). Unless the processor is in Debug state, the EPSR is not directly accessible. All fields read as zero using an MRS instruction and 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 for priority boosting. You can access the special purpose Priority Mask Register using the MSR and MRS instructions. You can also use the CPS instruction to set or clear PRIMASK.

Special Purpose Control Register

The special purpose Control Register identifies the stack pointers used.

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.

Memory Formats

The processor views memory as a linear collection of bytes numbered in ascending order from 0 (Figure 3).

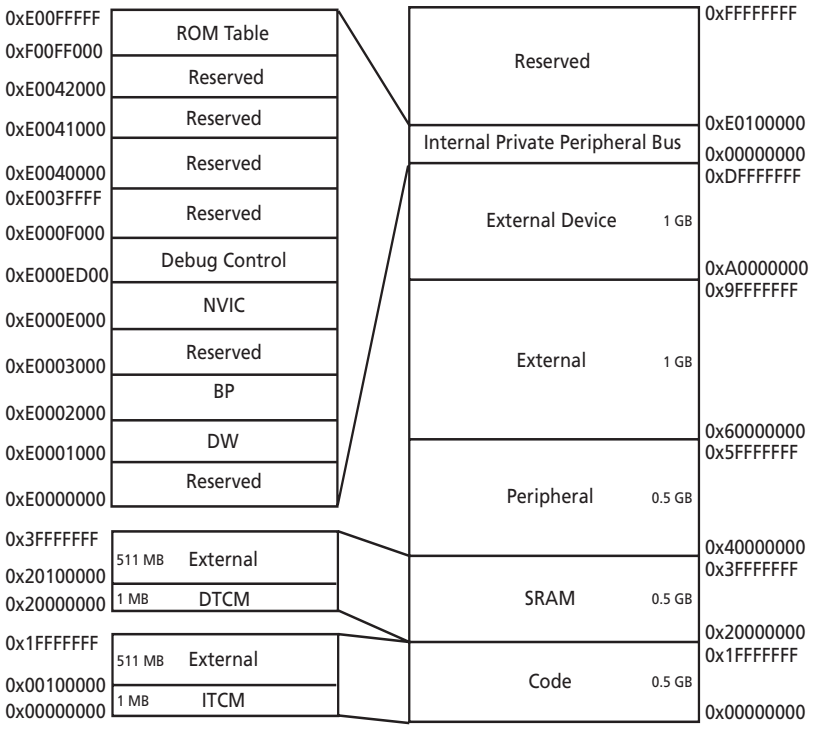

Figure 3 • **Processor Memory Map**

For example:

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

The processor accesses data and code words in little-endian format. Little-endian is the default memory format for ARM processors.

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.

Exceptions

The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. All exceptions are handled in Handler mode. 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
- Fixed number of interrupt priorities, from 2 bits, 4 levels
- 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

Various types of exceptions exist in the processor. 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 does not guarantee how it is reported with respect to the instruction that caused the fault. See Table 3 for a list and description of the exceptions supported by ARM Cortex-M1.

Table 3 • **Exception Types**

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	–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 takes exceptions. You can assign software priority levels to interrupts.

The NVIC supports software-assigned priority levels. You can assign a priority level from 0 to 3 to an interrupt by writing to the 2-bit IP_N field in an Interrupt Priority Register. Hardware priority decreases with increasing interrupt number. Priority level –3 is the highest priority level, and priority level 3 is the lowest. The priority level overrides the hardware priority.

Stacks

The processor supports two separate stacks:

- **Process stack** – You can configure Thread mode 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 use the main stack.

Clocking and Resets

The processor has one functional clock input, HCLK, and one reset signal, SYSRESETn. If debug is implemented, there is also a SWJ-DP clock, SWCLKTCK, and nTRST. SWCLKTCK relates to the DAP logic. 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 the following:

- Nested Vectored Interrupt Controller (NVIC)
- Debug subsystem

The register file cannot be reset by SYSRESETn or DBGRESETn.

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. All NVIC registers are only accessible 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. See Table 4 for a list of the NVIC registers and their addresses.

Table 4 • **NVIC Register**

Name of Register	Type	Address	Reset Value
IRQ 0 to 31 Set Enable Register	R/W	0xE000E100	0x00000000
IRQ 0 to 31 Clear Enable Register	R/W	0xE000E180	0x00000000
IRQ 0 to 31 Set Pending Register	R/W	0xE000E200	0x00000000
IRQ 0 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 repends and reactivates. 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 holds the signal in assert until the device is empty.

A pulse interrupt can be reasserted during the ISR so that the interrupt can be pended and active at the same time. The application design must ensure that a second pulse does not arrive before the first pulse is activated. If it does the second pend has no affect because it is already pended. However, if the interrupt is asserted for at least one cycle, the NVIC latches the pend bit. When the ISR activates, the pend bit is cleared. If the interrupt asserts again while it is activated, it can latch the pend bit again.

Processor Bus Interfaces

As currently available for use in M1 devices, the ARM Cortex-M1 has an AHB-Lite external interface. The processor also contains an internal bus called the Private Peripheral Bus (PPB) for accesses to the Nested Vectored Interrupt Controller (NVIC), Data Watchpoint (DW) unit, and BreakPoint (BPU), but this is not directly accessible to the user.

External Interface

The external interface is an AHB-Lite bus interface. The processor accesses to AHB peripherals and memory are implemented over this bus.

To prevent bus wait cycles from stalling the processor during data stores, buffered stores to the external interface 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 completes on the bus.

Memory Interfaces

The tightly coupled memory interface defined in the ARM Cortex-M1 architecture is not currently supported on M1 devices. Future core releases will have support for this memory interface.

Private Peripheral Bus

The AHB PPB is used to access the Nested Vector Interrupt Controller and the debug components when they are present. The PPB allows communication to flow between the AHB and NVIC units and the debug circuitry when it is implemented in the core.

Delivery and Deployment

ARM Cortex-M1 is delivered as a series of files by CoreConsole that are directly imported into the Design and Simulation folders by Libero IDE. These consist of the BFM files and test wrapper, and the A1S secured CDB file, which is the placed-and-routed ARM Cortex-M1 core, actually instantiated on the user device. This deployment flow is adopted to ensure that the design is kept completely secure at all times.

Initially, the ARM 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 0K ITCM, 0K DTCM, small multiplier, little-endian, no OS extensions, and 1 interrupt.

Bus Functional Model (BFM)

Introduction

During the development of an FPGA-based SoC, there are various stages of testing that can be undertaken. This can involve some, or all, of the following approaches:

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

BFM Usage Flow

The BFM acts as a pin-for-pin replacement of the Cortex-M1 in the simulation of the SoC subsystem. It initiates bus transactions on the native ARM Cortex-M1 bus, which are cycle-accurate with real bus cycles that ARM Cortex-M1 would produce. It does not have the ability, however, to implement real ARM Cortex-M1 instructions. The BFM may be used to run a basic test suite of the SoC subsystem, using the skeleton system testbench.

You can edit the SoC Verilog/VHDL to add new design blocks. You can also fill out the system-level testbench to include tasks that test any newly added functionality, or add stubs to allow more complex system testing involving the IP cores. The BFM input scripts can also be manually enhanced to test out access to register locations in newly added logic. In this way, you can provide stimuli to the system from the inside (via the ARM Cortex-M1 BFM), as well as from the outside (via testbench tasks).

Timing Shell

There is a timing shell provided for each ARM Cortex-M1 variant wrapped around the BFM. Therefore, the BFM is bus cycle accurate, and performs setup/hold checks to model output propagation delays.

Debug

The ARM Debug Architecture uses a protocol converter box to allow the debugger to talk directly to the core via a JTAG port. In effect, the scan chains in the core that are required for test are re-used for debugging. The core uses the scan chains to insert instructions directly into ARM Cortex-M1. 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.

In debug mode, the user can perform the following functions:

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

The main debug components are the following:

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

Datasheet Categories

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

Product Brief

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

Advanced

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

Unmarked (production)

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

www.actel.com

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

Actel Europe Ltd.
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 Building 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
Room 2107, China Resources Building
26 Harbour Road
Wanchai, Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

51700087PB-4/12.08