



**The Abdus Salam
International Centre for Theoretical Physics**



2065-20

**Advanced Training Course on FPGA Design and VHDL for Hardware
Simulation and Synthesis**

26 October - 20 November, 2009

Introduction to SoC Design

Nizar Abdallah
*ACTEL Corp. 2061 Stierlin Court Mountain View
CA 94043-4655
U.S.A.*



Introduction to SoC Design

Agenda

- Overview
- ARM Cortex-M1 Architecture
- Cortex-M1 Instruction Set
- Actel's Cortex-M1
- System-on-Chip and SmartDesign
- Testing
- Cortex-M1 Development Tools
- Reference



Overview

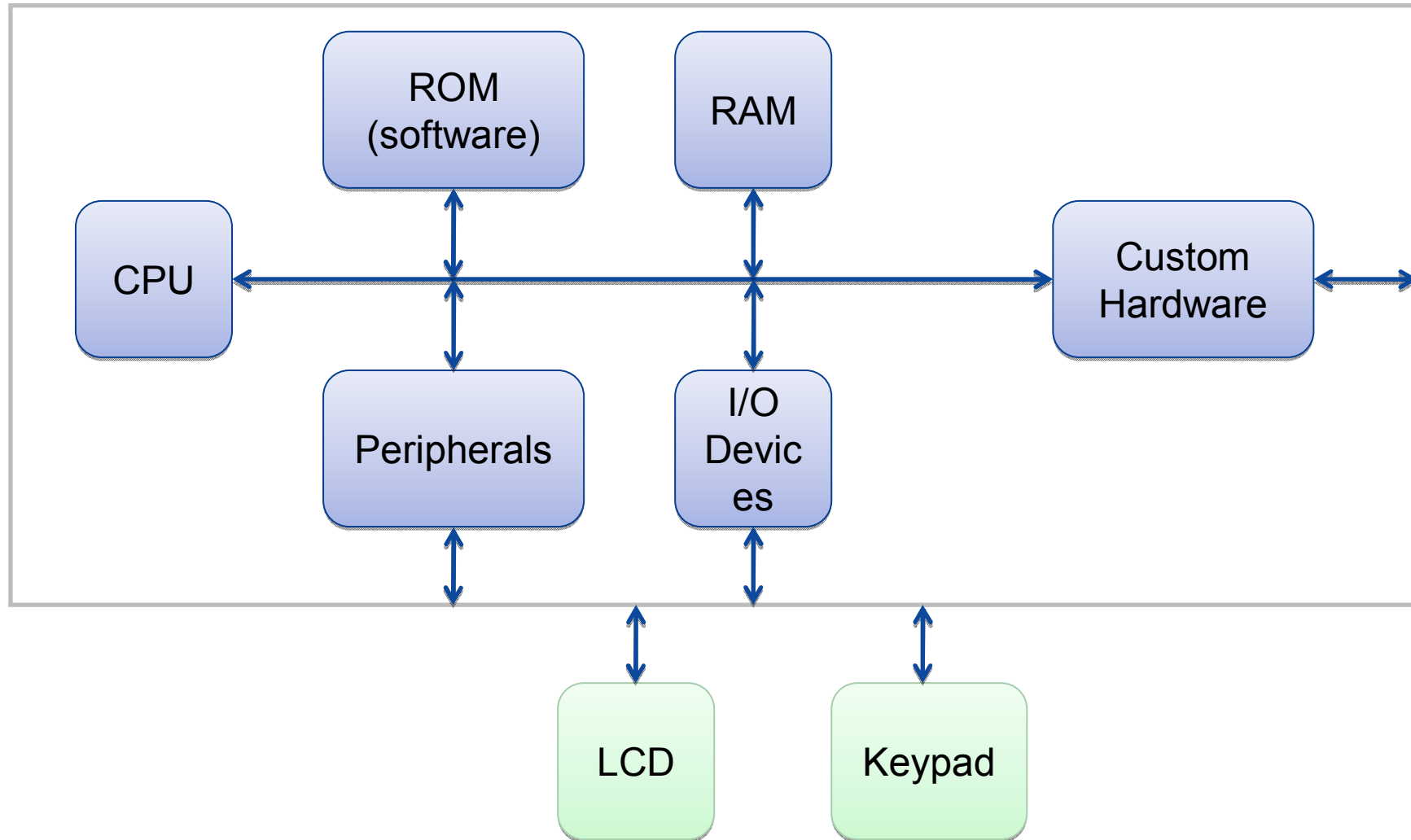
SoC Definition

- What exactly is SoC?
 - Entire system built on a single piece of silicon
 - Includes a processor, memory, DSP Cores, IO devices, interfaces to external circuitry, and custom IPs as Verilog or VHDL modules
 - SoC designs primarily used in embedded applications
- SoC design brings a new level of complexity to IC design
 - Cost to develop is very high
 - Large and complex IP from many diverse sources
 - “Black box” view of IP (cannot change or “see” details)
 - Lack of existing interface standards
 - Fairly complex SW running on an on-chip CPU
 - Integration, performance, and power are crucial
 - Mixed signal (analog & digital) circuitry

Embedded System Definition

- A dedicated computer hardware with software designed to solve a specific problem
- Uses “hidden” microprocessors from 8bit MC to 64-bit MP
- Some RAM or ROM is required; Flash commonly used
- ...and a mix of Timers, Interrupt controller, UARTs, GPIO, DMA controllers, Real time clock, LCD controller...
- Embedded system software divided into operating software and application software

Embedded System Diagram



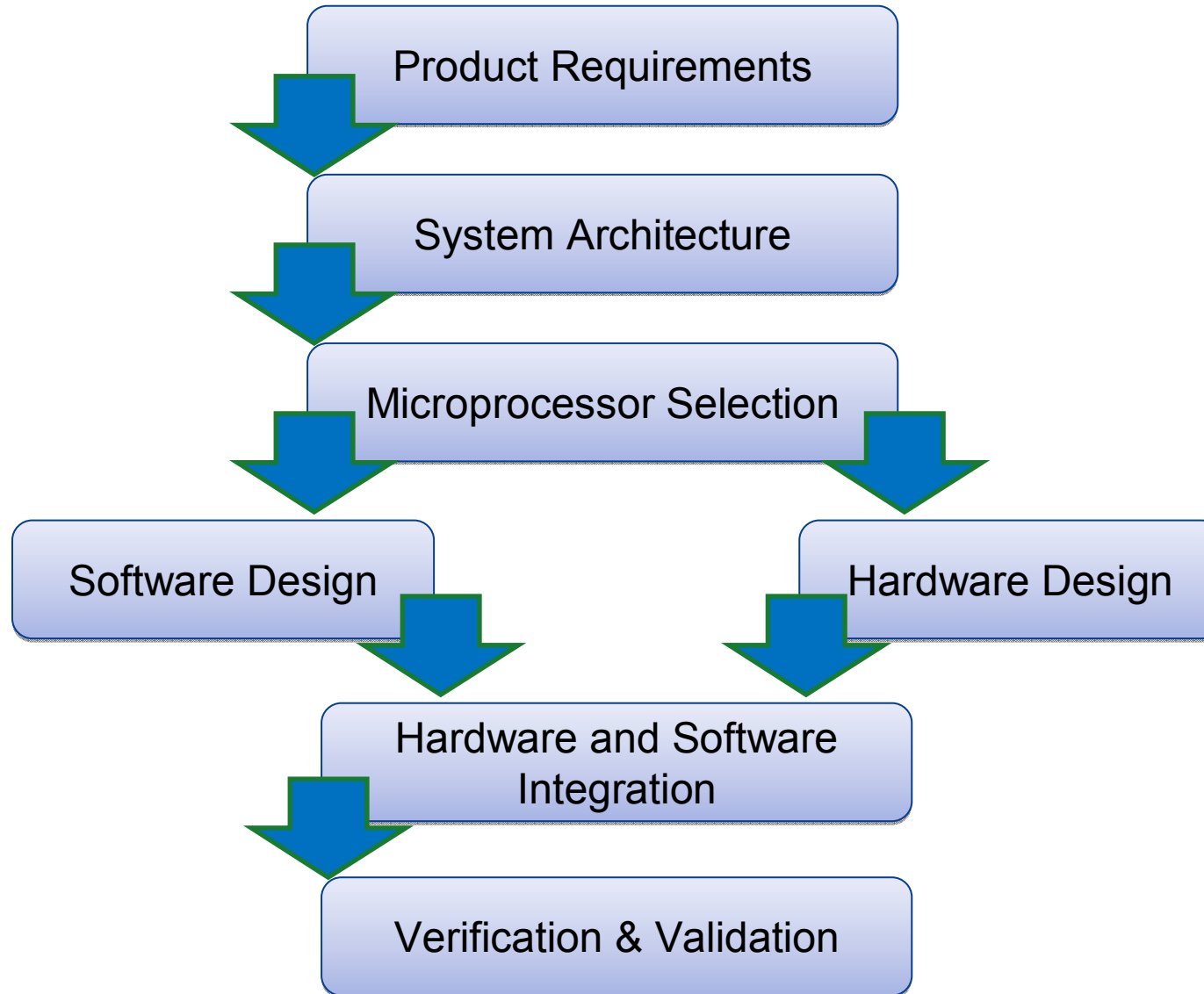
Embedded System Design Constraints

- Cost
- Memory
- Power
- Real-time Response
- Performance
- System Size
- Reliability
- Time-to-Market

Embedded System Classification

- Little or no custom hardware design
 - Time-to-market requirement
 - Use high integration microprocessor or off the shelf boards
- A lot of custom hardware – SoB design
 - High performance requirements
 - High performance microprocessor (PowerPC, MIPS chips)
 - Large custom logic on boards (FPGA, ASIC)
 - Even maybe multiple boards...
- A lot of custom hardware – SoB design
 - Small size and low-power requirements
 - One or more microprocessors on a chip
 - IPs such as ARM or MIPS
 - Cost to develop is very high
 - Consumer electronics: MP3, digital cameras...

Embedded System Design Process



Embedded System Design Process

- Requirements
 - Product specifications
 - Required features and functionalities
- System Architecture
 - Defines major blocs and functions
 - Interfaces, bus structure, HW & SW functionalities
 - Simulation, SW, and Spreadsheet to define best architecture
 - How many packet/sec can this router handle?
- Microprocessor selection
 - One of the most difficult tasks
 - Perf, cost, power, SW dev. Tools, legacy SW, RTOS, Sim models...

Embedded System Design Process

- Hardware Design
 - Important data to the SW team
 - CPU address map, Register definition for all SW programmable registers
- Software Design
 - Boot code
 - Hardware diagnostics, device drivers, Application software
 - RTOS
- Hardware & Software integration
- Verification and Validation

Example 1: Embedded System

- In order to test the hardware, a CORDIC algorithm is to be implemented. This algorithm calculates the sine and cosine functions of a given number. Such a number is to be entered over the UART interface through a Hyper-Terminal or some similar program on a PC.
- The first action of the main program will initialize the UART. Then some text is to be output over the UART, following which the program would wait for input from the user. Such input would first be checked for any errors, before being formatted as a fixed point number, with 8 bits before and 24 bits after the decimal point, and passed on to the function that implemented the CORDIC algorithm.
- The result of the calculation are to be transformed back into strings and output over the UART.



Introduction to Cortex-M1

Agenda

- Overview
- ARM Cortex-M1 Architecture
- Cortex-M1 Instruction Set
- Actel's Cortex-M1
- System-on-Chip and SmartDesign
- Testing
- Cortex-M1 Development Tools
- Reference



Overview

ARM History

- First ARM Processor Developed in Mid-eighties
 - Acorn Computers, Limited in Cambridge, England
 - Originally, ARM Stood for Acorn RISC Machine
 - Later Changed to Advanced RISC Machine
- All Major Chip Manufacturers Have Licenses to One or More ARM Cores, more than 100...
 - Analog Devices, Atmel, Cirrus, Fujitsu, IBM, Infineon, Intel, Mitsubishi, Motorola, National Semiconductor, NEC, Philips, Sharp, ST Microelectronics, Texas Instruments, Toshiba ...
- ARM7TDMI Most Popular ARM Core in Embedded Systems
- Business Model
 - License fees for microprocessors and other IP blocks
 - Per-chip royalties on shipments of chips using ARM IP
 - Tools & boards to support development and debugging

ARM Processors

- Established Processors
 - ARM7 (e.g., Actel's CoreMP7), ARM9, ARM11, and Others
- RISC principles: Simplicity and High Instruction Throughput
- New Cortex Processors from ARM
 - Range of Processors which Target Different Markets/Applications
 - Three Cortex Variants:

Series	Market Segment	Instruction Sets Supported
A	Applications – Complex Operating Systems and User Applications	ARM, Thumb and Thumb-2
R	Real-time Systems	ARM, Thumb and Thumb-2
M	Microcontroller – Deeply-embedded Processors for Cost/Power-sensitive Applications	Thumb-2 Only

RISC Processors

- World Domination

Merchant RISC Microprocessor Shipments (1000s)

	thru 1994	1995	1996	1997	1998	1999	2000	2001
ARM/StrongARM	2,170	2,100	4,200	9,800	50,400	152,000	414,000	402,000
MIPS	3,254	5,500	19,200	48,000	53,200	57,000	62,800	62,000
Hitachi SH	2,800	14,000	18,300	23,800	2,600	33,000	50,000	45,000
PowerPC	2,090	3,300	4,300	3,800	6,800	8,300	18,800	23,000
Total	30,499	33,830	58,480	98,220	149,080	262,820	556,800	538,860

Source: Andrew Allison

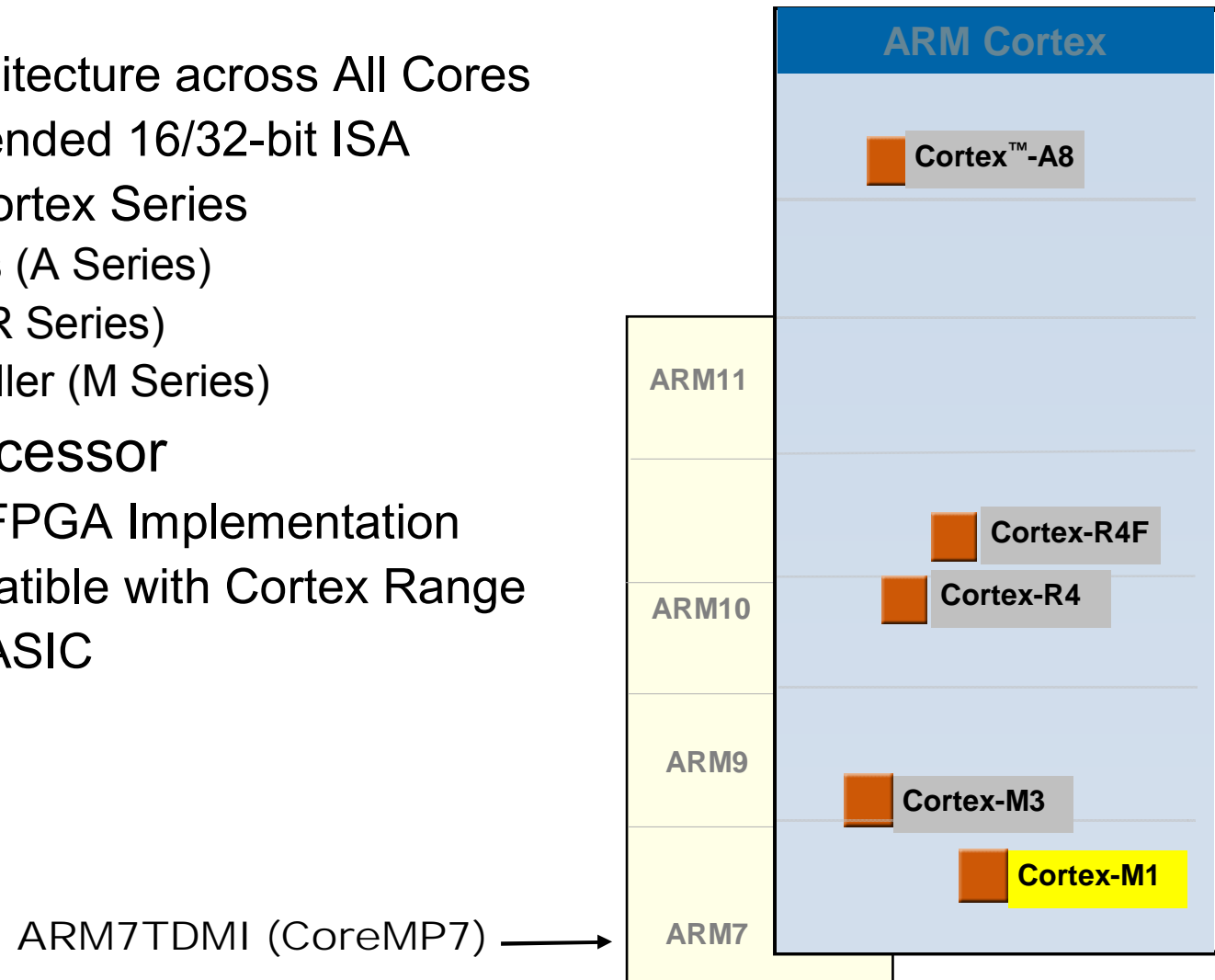
ARM's New Processor Family

■ Cortex

- Common Architecture across All Cores
- Thumb®-2 Blended 16/32-bit ISA
- Three ARM Cortex Series
 - Applications (A Series)
 - Real-time (R Series)
 - Microcontroller (M Series)

■ Cortex-M1 Processor

- Designed for FPGA Implementation
- Upward-compatible with Cortex Range
- Easy Path to ASIC



ARM Cores

- Hard Macro form
 - Provided as a layout object
 - High performance and small die area
 - Limited portability between different silicon processes

- Soft Macro form
 - Verilog RTL format
 - More flexibility
 - Less performance
 - Most common form today!



Cortex-M1

Next Member of the M Series

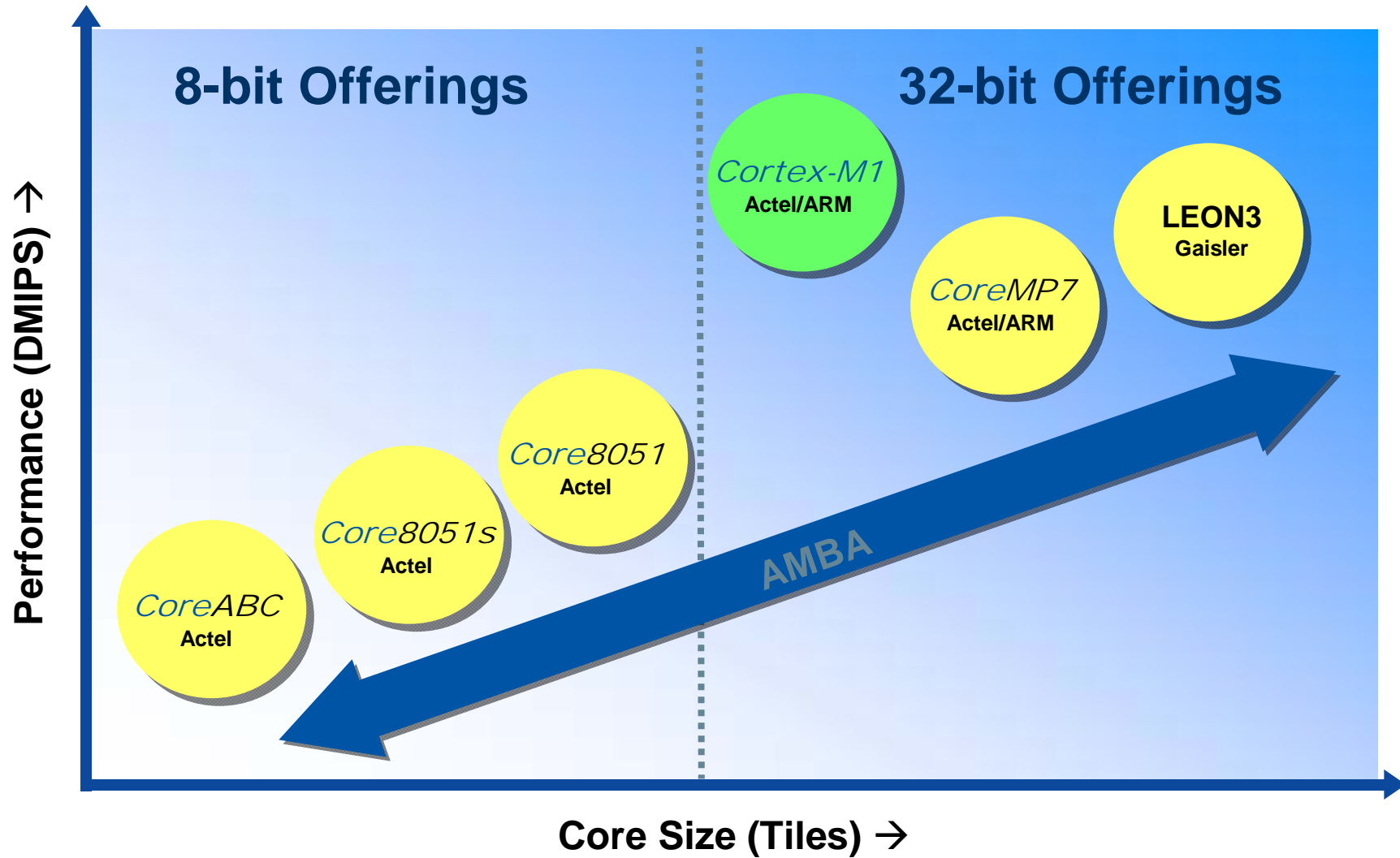
- Intended for FPGA Implementation
 - Soft Processor (Implemented in FPGA Fabric)

- Small, Powerful, Highly-optimized, and Configurable

- ARMv6-M Architecture
 - Subset of Thumb-2: All 16-bit Thumb Instructions and Some 32-bit Thumb-2 Instructions

- Delivered as Black Box via SmartDesign

Actel Processor Offerings



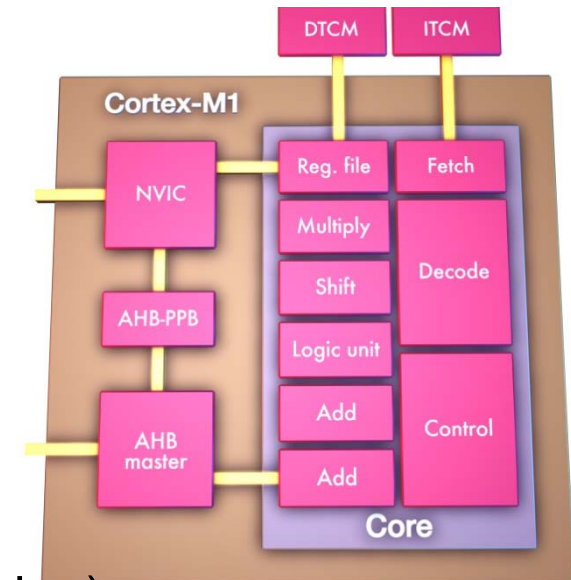
AMBA: Advanced Microcontroller Bus Architecture

DMIPS: Dhrystone MIPS (Million Instructions Per Second)

Cortex-M1

Overview of Features

- Soft 32-bit IP Core for FPGAs
 - High-frequency, Low-area Design
 - Executes All Existing Thumb® Code
 - ARMv6-M Instruction Set (Thumb2)
 - Nested Vectored Interrupt Controller (NVIC)
- Balance between Size and Speed
 - Operates at up to 62MHz
 - Implemented in as few as 4300 Tiles (A3P / Fusion)
 - Uses Three-stage Pipeline
- Interfaces
 - ARM AMBA® AHB-Lite™ Interface (Single Master)
 - Separate Data & Instruction Memory Interfaces
 - Debug via JTAG Interface



Cortex-M1

Nested Vectored Interrupt Controller (NVIC)

Name of register	Type	Address
Interrupt Set Enable Register	R/W	0XE000E100
Interrupt Clear Enable Register	R/W	0XE000E180
Interrupt Set Pending Register	R/W	0XE000E200
Interrupt Clear Pending Register	R/W	0XE000E280
Priority 0 Register	R/W	0XE000E400
Priority 1 Register	R/W	0XE000E404
Priority 2 Register	R/W	0XE000E408
Priority 3 Register	R/W	0XE000E40C
Priority 4 Register	R/W	0XE000E410
Priority 5 Register	R/W	0XE000E414
Priority 6 Register	R/W	0XE000E418
Priority 7 Register	R/W	0XE000E41C

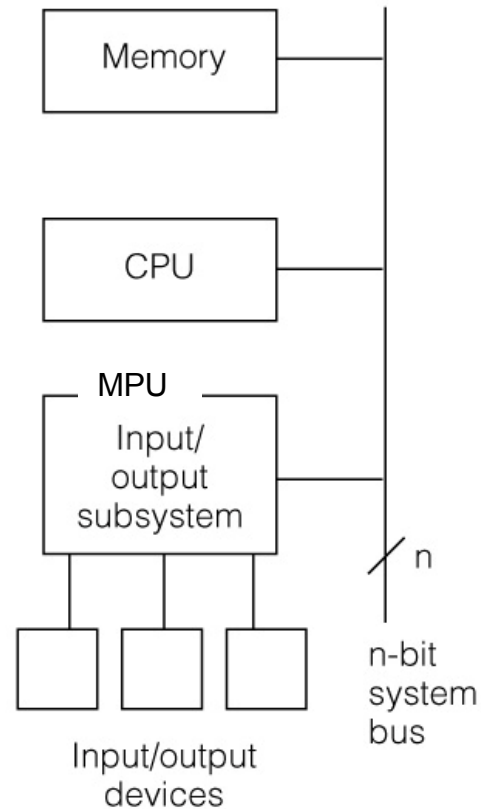
Source: ARM Inc.



Cortex-M1 Architecture

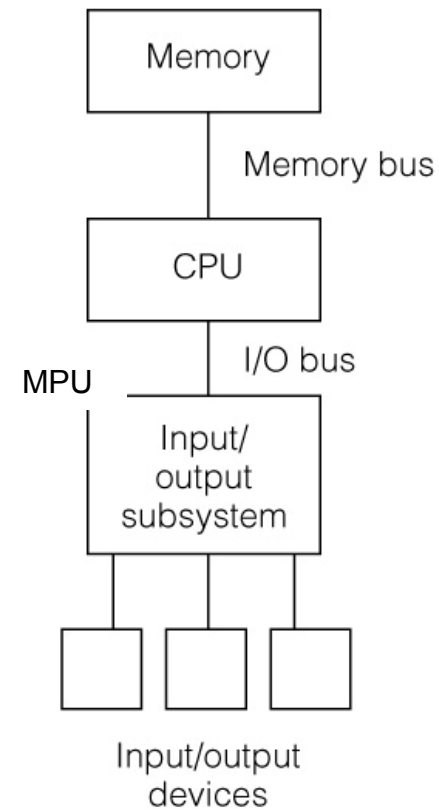
Processor Architectures

von Neumann



One Bus - Program and Data Located in *Same* Address Space

Harvard



Two Buses - Program and Data Located in *Separate* Address Spaces

Cortex-M1

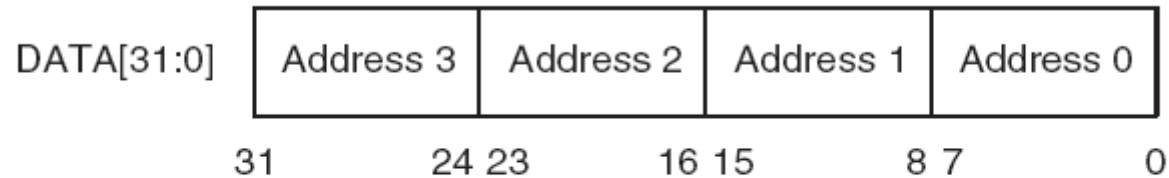
Architecture and Memory Organization

- Cortex-M1 is von Neumann Architecture ...
 - Linear 32-bit Address Space (4 GB)
 - Only Load, Store, and Swap Instructions Can Access Data from Memory
- ... Separate Tightly-Coupled Memories Make It 'Slightly Harvard'
 - Separate Instruction and Data TCMs
 - Limited to 1MB (Maximum) Each
- Cortex-M1 Processor Supports Three Data Types
 - Word (32-bit)
 - Half-word (16-bit)
 - Byte (8-bit)
- Data Alignment
 - Words Must Be Aligned to Four-byte Boundaries
 - Half-words Must Be Aligned to Two-byte Boundaries
 - Bytes Can Be Placed on any Byte Boundary
- Supports Both Big- and Little-endian Formats

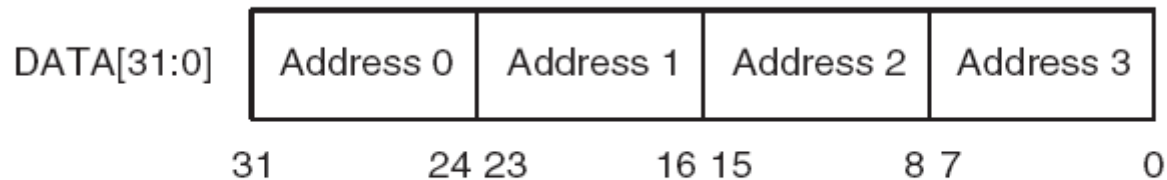
Cortex-M1

Architecture and Memory Organization

- Historically, Intel uses Little-endian and RISC Big-endian
- ARM allows the user to choose the byte order



Little endian byte order



Big endian byte order

Cortex-M1

Architecture and Memory Organization

- When data transfers are a full word, the data on the bus is identical for both big-endian and little-endian byte order
- *Endianness* becomes important for both master and slave when data transfers on the bus are for 1 or 2 bytes
- One of the first questions to always ask for a new design is, “What is the byte order of the design, big or little endian?”

Cortex-M1

Architecture and Memory Organization

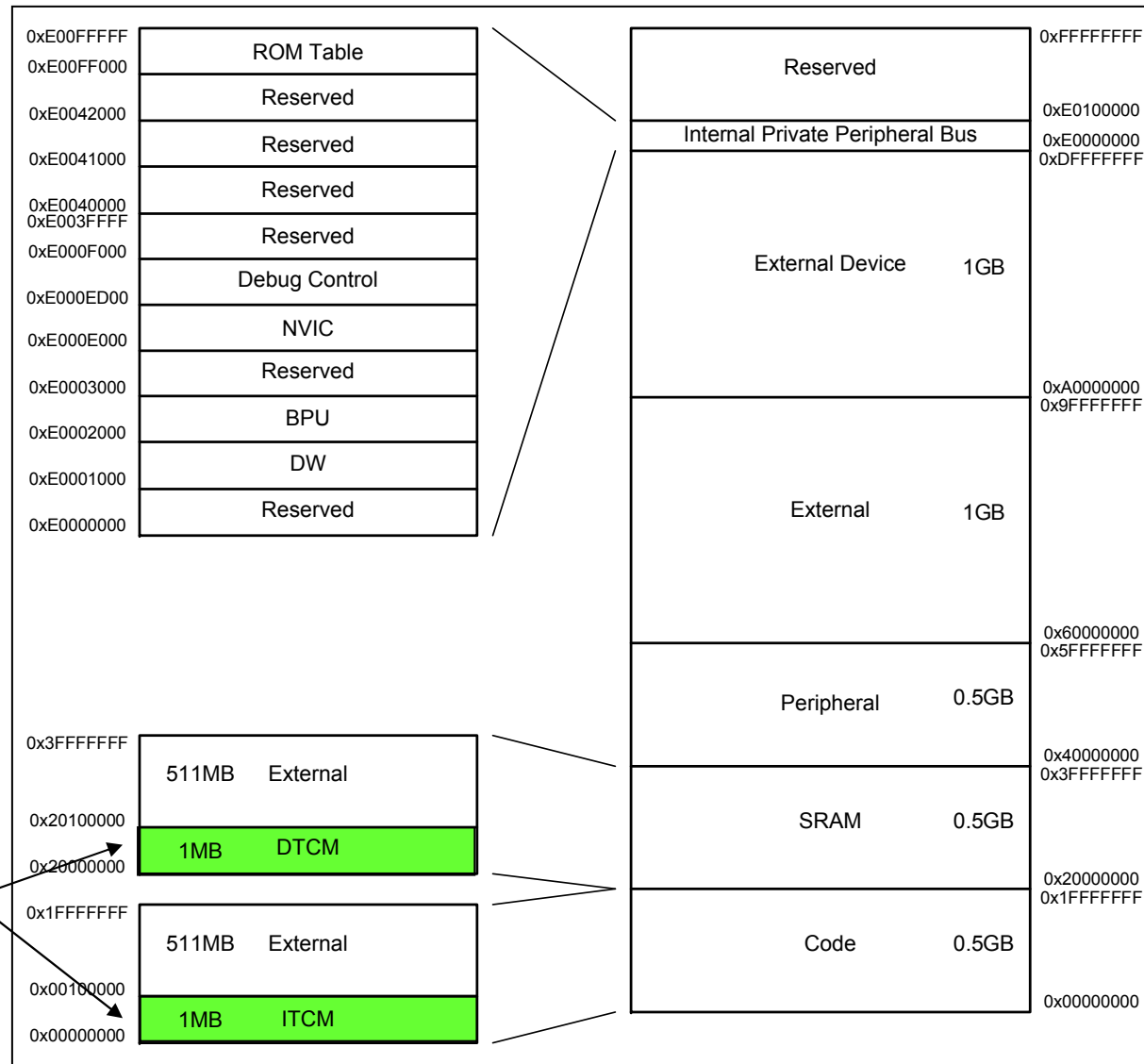
- RISC 32-bit architecture gives best performance to operate on 32-bit data
- Drawback: amount of memory required to hold 32-bit instructions, especially in embedded systems
- Code Density in Embedded Systems
- ARM allows running 16-bit instructions called Thumb: 30% less memory for 30% less performance

Cortex-M1

Memory Map

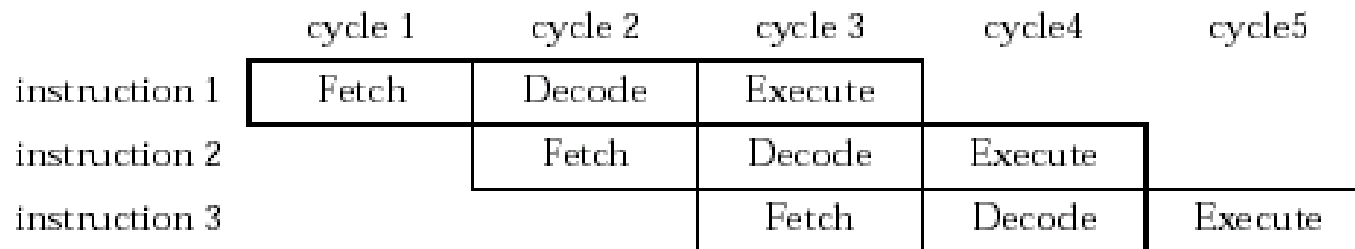
Little-endian Is Default Memory Access Format

Tightly-Coupled Memory Spaces



M1's 3-Stage Instruction Pipeline

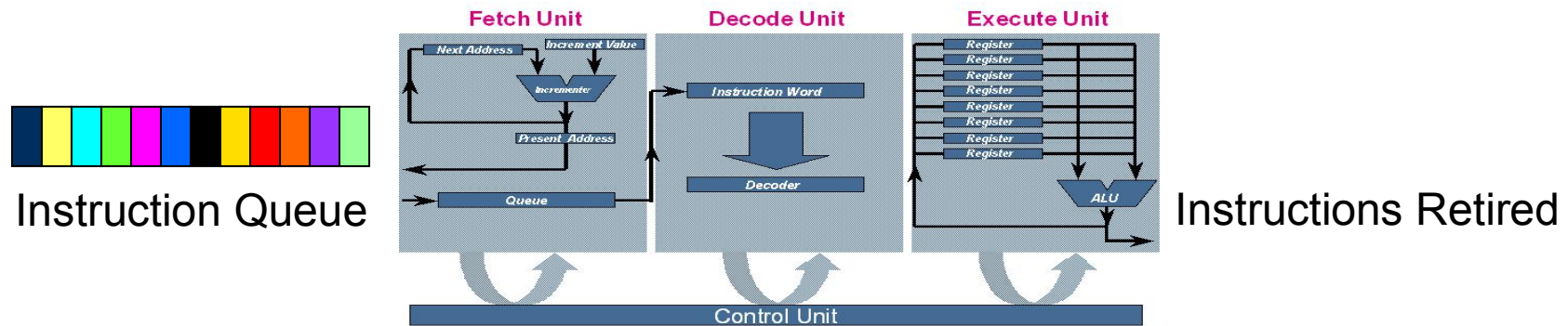
- Three Stages Are:
 - Fetch - Fetching an Instruction from Memory Containing Code
 - Decode - Decoding Instruction and Preparing Datapath Control Signals for Next Cycle
 - Execute – Reading Source Registers, Performing Shift or ALU Operations, and Writing Back Result to Destination Register
- One Instruction is Executed Every Cycle when Pipeline is Full
 - 3 cycles Needed to Completely Process One Instruction
 - 3 cycles Needed to Reload Pipeline – Branch Instructions
 - Pipeline Halted for One Cycle if Multiple Memory Accesses Needed to Execute Instruction



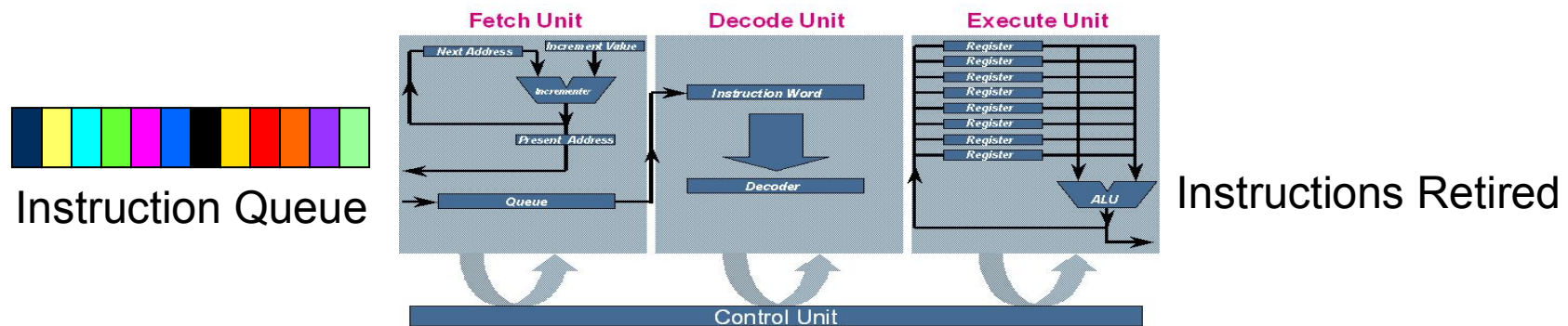
Increasing Processor Performance

Pipelining

Sequential Processor



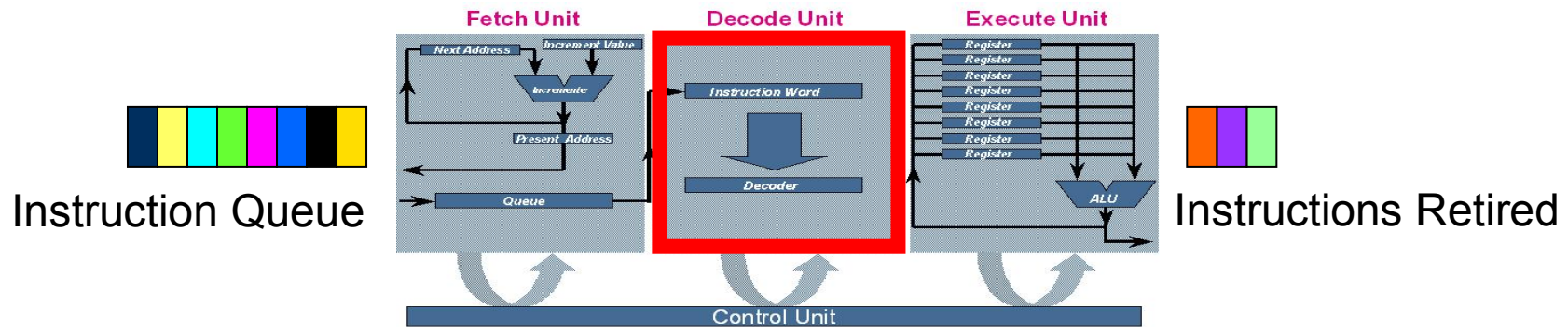
Pipelined Processor



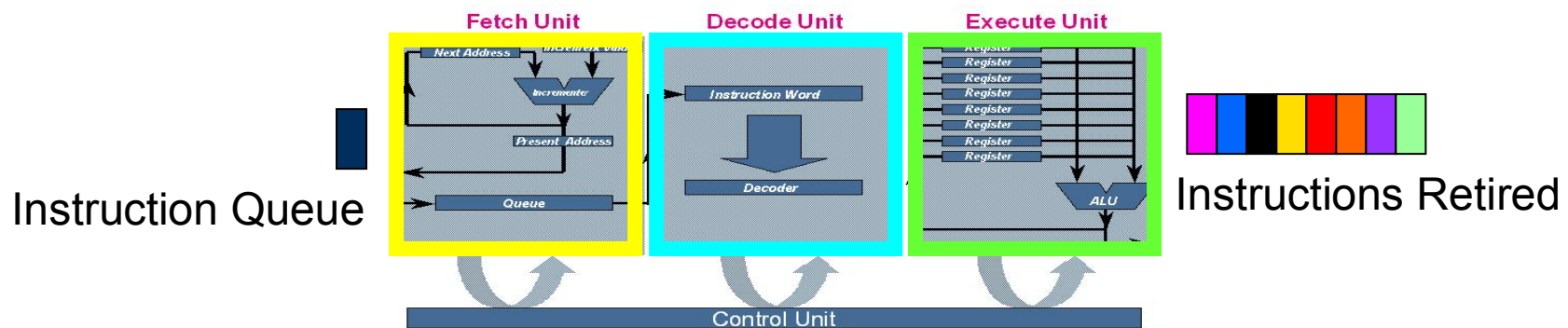
Increasing Processor Performance

Pipelining

Sequential Processor



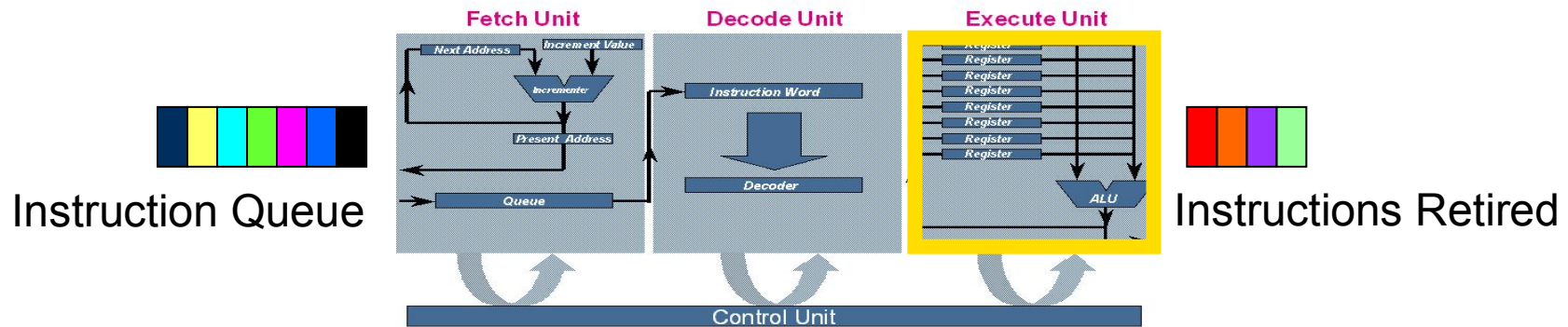
Pipelined Processor



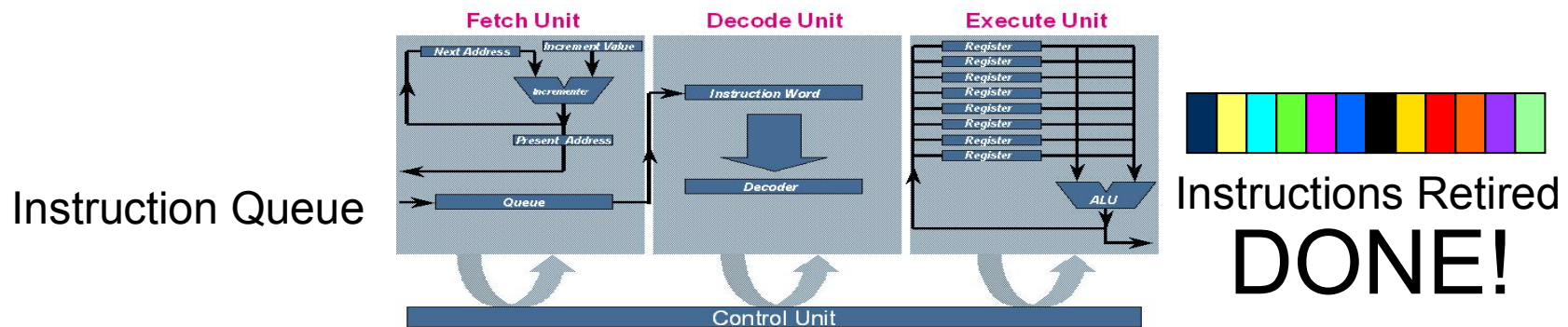
Increasing Processor Performance

Pipelining

Sequential Processor

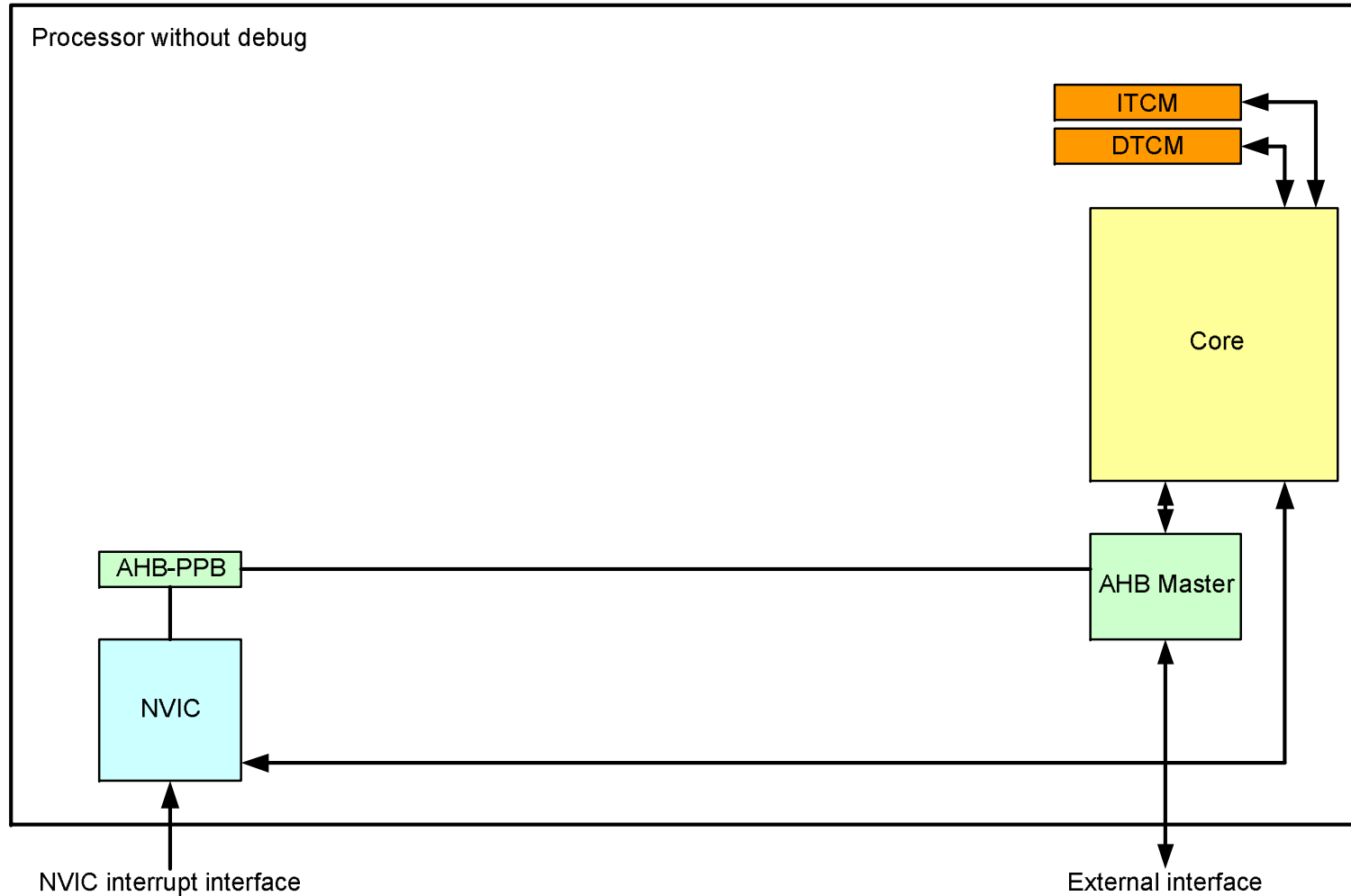


Pipelined Processor



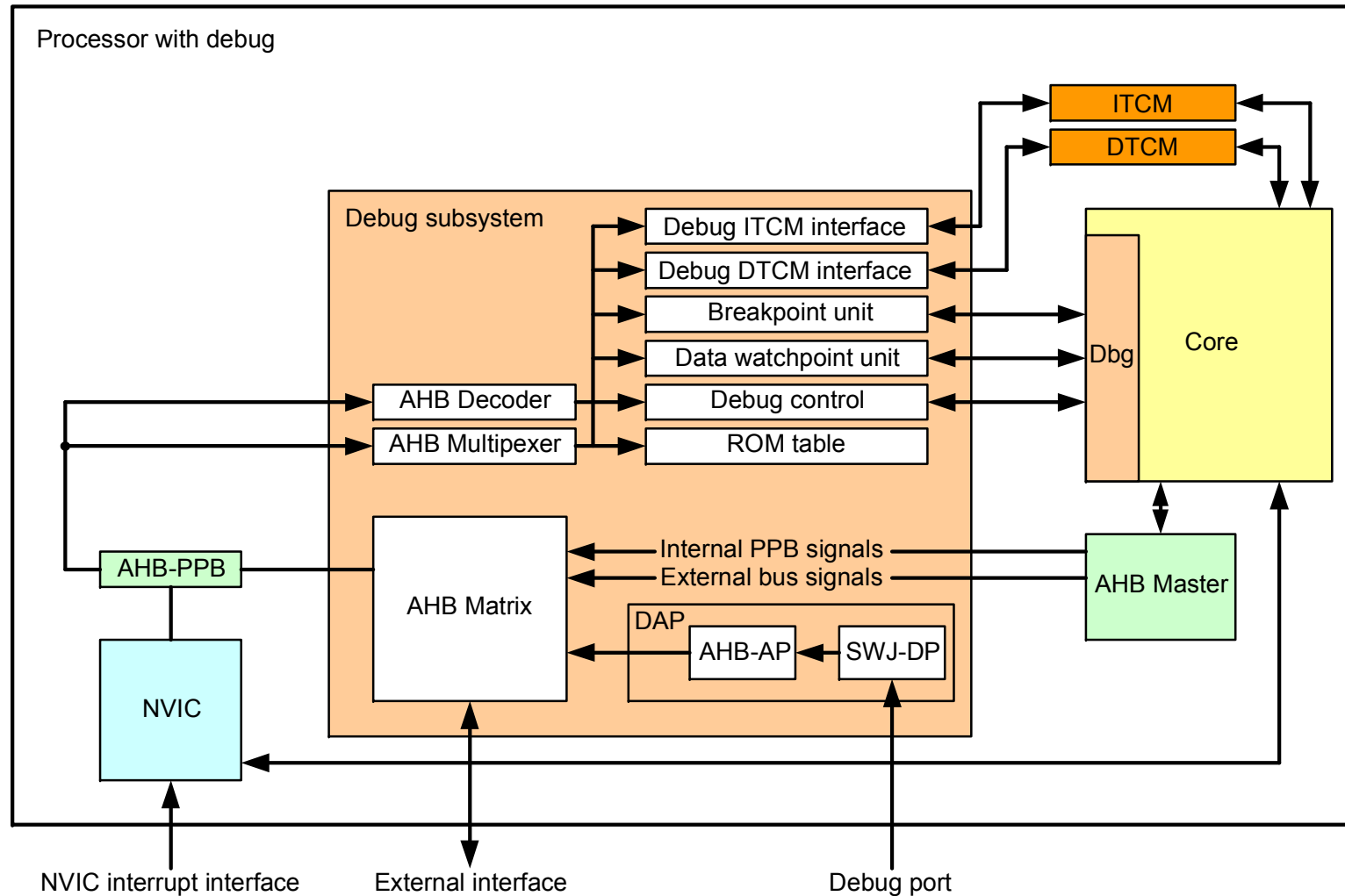
Cortex-M1 Without Debug

Block Diagram



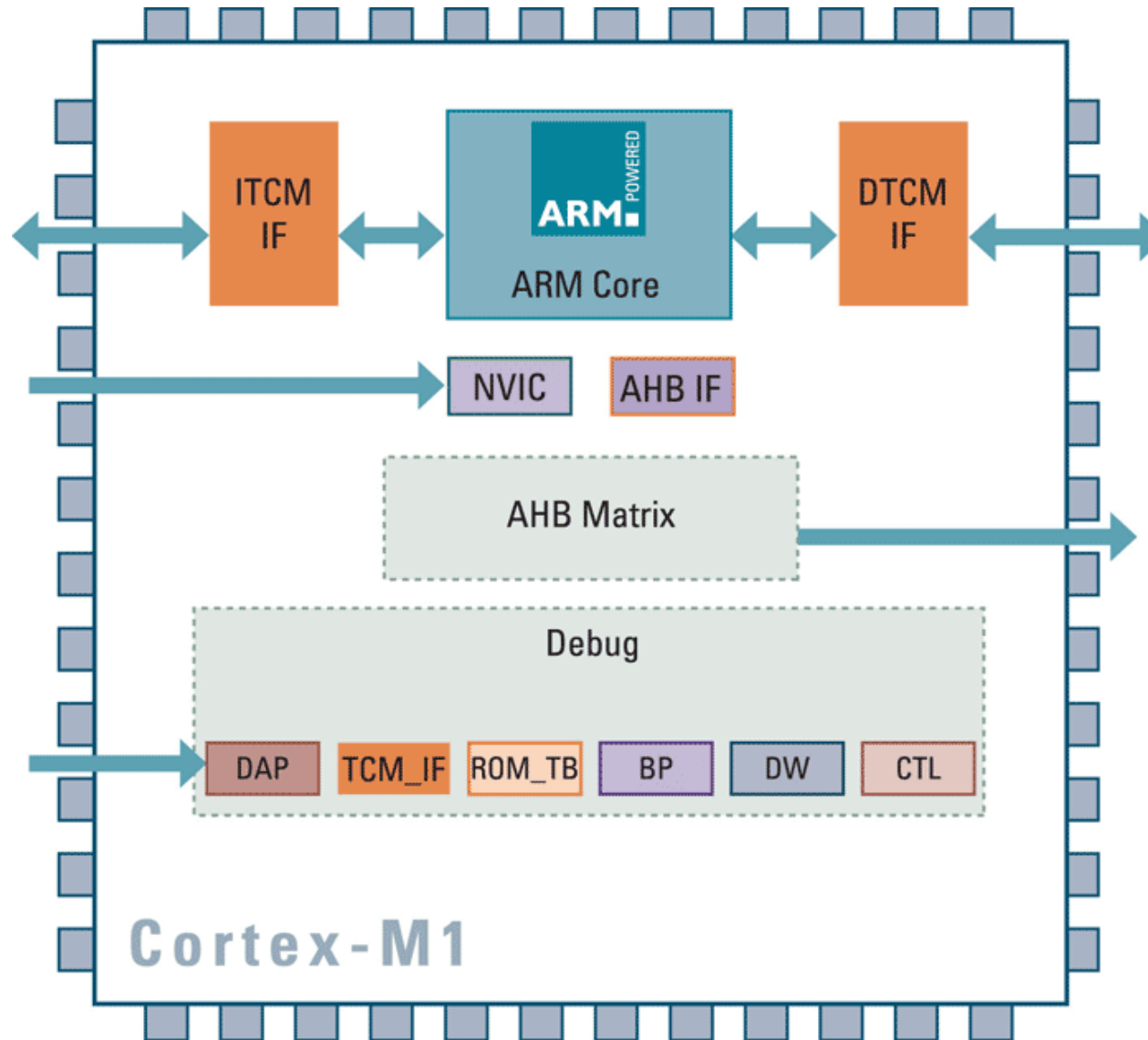
Cortex-M1 with Debug

Block Diagram



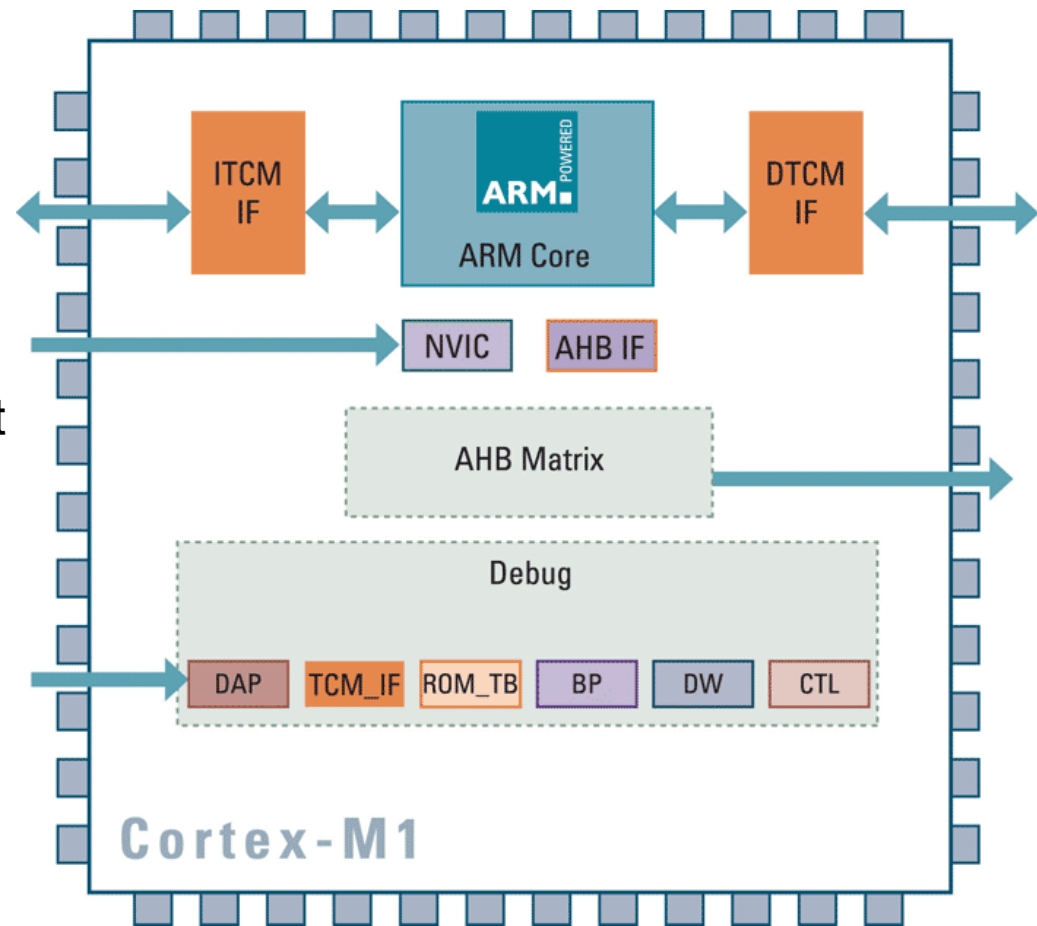
Cortex-M1 Architectural Diagram

Block by Block



Cortex-M1 Architectural Diagram *Acronyms*

- TCM
 - Tightly-Coupled Memories
- NVIC
 - Nested Vectored Interrupt Controller
- AHB IF
 - AMBA AHB Bus Interface
- DAP
 - Debug Access Port (See Next Slide)
- ROM_TB
 - ROM Table
- BP
 - Breakpoint Unit
- DW
 - Data Watchpoint Unit



Debug Control Registers

Name	Description
ABORT	Access Port Abort Register
IDCODE	ID Code Register
CTRL/STAT	Debug Port Control/Status Register
SELECT	Access Port Select Register
RDBUFF	Read Buffer

- Access to Debug and AHB Access Control Registers is via Debug Port
 - UJTAG Macro Is Included in Debug Port Logic

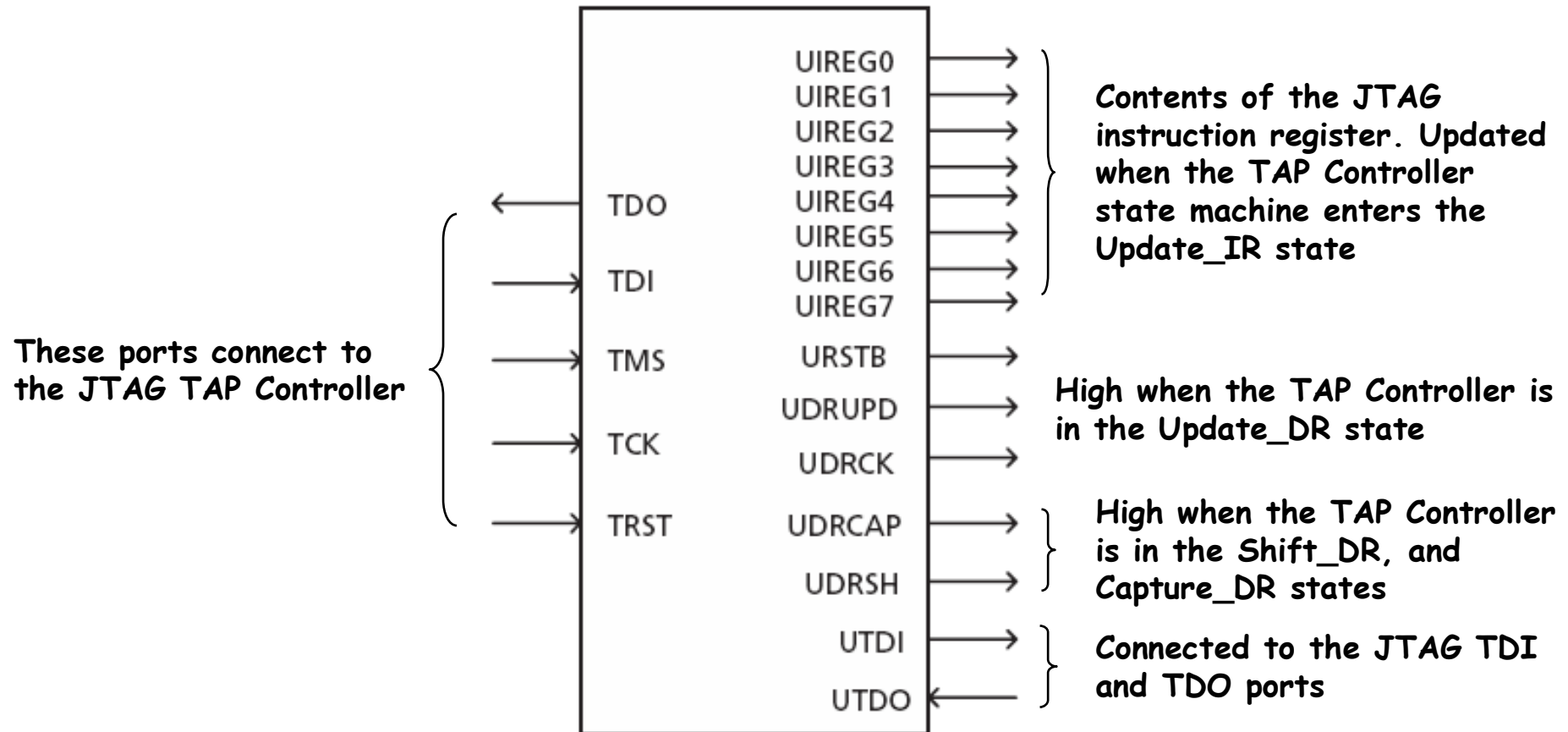
Debug Control Registers

- Two configurations for debug:
 - The full debug configuration has four breakpoint comparators and two watchpoint comparators. Default configuration.
 - The reduced debug configuration has two breakpoint comparators and one watchpoint comparator.
- Debug facilitates:
 - core halt
 - core stepping
 - core register access while halted
 - read/write to: TCMs, AHB address space, internal Private Peripheral Bus (PPB)
 - Breakpoints
 - watchpoints

Main debug components

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

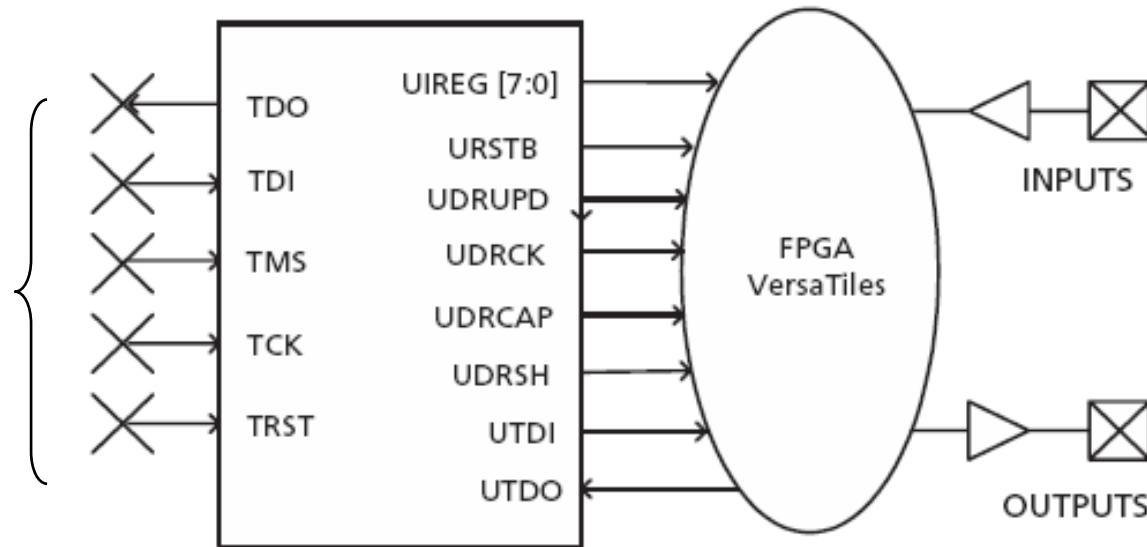
UJTAG Macro Symbol



UJTAG Macro Is Already Instantiated in Cortex-M1 Debug Unit!

UJTAG Interconnection

These ports must
NOT be connected to
any I/O buffer in
the netlist



Cortex-M1

Technical Overview

■ Core

- Three-stage Pipeline
- ARMv6-M Instruction Set Architecture
 - All 16-bit Thumb Instructions and Some Thumb-2 32-bit Instructions
- Datapath Optimized for Thumb and for FPGA Implementation
 - Multiplier, Adder, Shifter and Logic Unit, All in Parallel
- Tightly-coupled Interface to Interrupt Controller
- Supports Tightly-Coupled Memory (TCM) for Instructions and Data

■ Registers

- 13 General-purpose 32-bit Registers
- Link Register (LR), Program Counter (PC), Program Status Register (xPSR), Stack Pointer (SP)

Cortex-M1

Technical Overview (cont.)

■ Debug

- Debug via JTAG or 2-pin Serial-wire Interface
- Provides Access to All Registers and Memory
- Includes Break-point Unit (BP) and Data-watchpoint (DW) Unit
 - 4 Breakpoints and 2 Watchpoints

■ Nested Vectored Interrupt Controller (NVIC)

- NVIC Is Tightly Coupled to Processor Core
- Low-latency Exception Processing
- Level and Pulse Interrupts Supported
- Processor State Automatically Saved and Restored when Switching to Interrupt Service Routine (ISR)

AMBA Bus Interface Signals

AHB-Lite Bus

Signal	Function
HCLK	Bus Clock
HRESETn	ACTIVE-LOW Bus Reset
HADDR(31:0)	System Address Bus
HTRANS(1:0)	Transfer Type
HWRITE	Transfer Direction (1=Write, 0=Read)
HSIZE(2:0)	Transfer Size
HBURST(2:0)	Indicates if Transfer Forms Part of a Burst
HPROT	Protection Control
HWDATA(31:0)	32-bit Write Data (from Master)
HRDATA(31:0)	32-bit Read Data (to Master)
HREADY	Transfer Complete
HRESP(1:0)	Transfer Response
HMASTLOCK	Master Signal

Cortex-M1

Debug Interfaces

- Note: Debug Interface Signals Are ALWAYS Present!

Name	Type	Description
RV_TCK	Input	RealView JTAG
RV_nTRST	Input	RealView JTAG
RV_TMS	Input	RealView JTAG
RV_TDI	Input	RealView JTAG
RV_nSRST_IN	Input	RealView JTAG
RV_TRCK	Input	RealView JTAG
RV_TDOOUT	Output	RealView JTAG
RV_nTDOEN	Output	RealView JTAG
UJTAG_TCK	Input	FlashPro3 JTAG
UJTAG_TDI	Input	FlashPro3 JTAG
UJTAG_TMS	Input	FlashPro3 JTAG
UJTAG_TRSTB	Input	FlashPro3 JTAG
UJTAG_TDO	Output	FlashPro3 JTAG
EDBGRQ	Input	External debug request

Cortex-M1

Miscellaneous Signals

Name	Type	Description
HCLK	Input	Main processor clock
NSYSRESET	Input	External push-button/power-up reset
WDOGRES	Input	Watchdog reset to Cortex-M1
WDOGRESn	Output	Reset of watchdog timer
HRESETn	Output	Reset to other components in AHB system
IRQ[31:0]	Input	External Interrupts
NMI	Input	Non-maskable Interrupt
nTRST	Input	JTAG reset
JTAGTOP	Output	State Controller Indicator
nTDOEN	Output	JTAG data out enable
LOCKUP	Output	Core is locked up
HALTED	Output	Core is in Halt debug state

Cortex-M1

Processor Operating States and Modes

■ States

- Thumb
 - Normal Execution
 - Runs 16-bit Halfword-aligned Thumb and Thumb-2 Instructions plus 32-bit BL, MRS, MSR, ISB, DSB, and DMB Instructions
 - Data Types Supported – 32-bit Words, 16-bit Halfwords, 8-bit Bytes
- Debug State
 - Used for Halting Debug

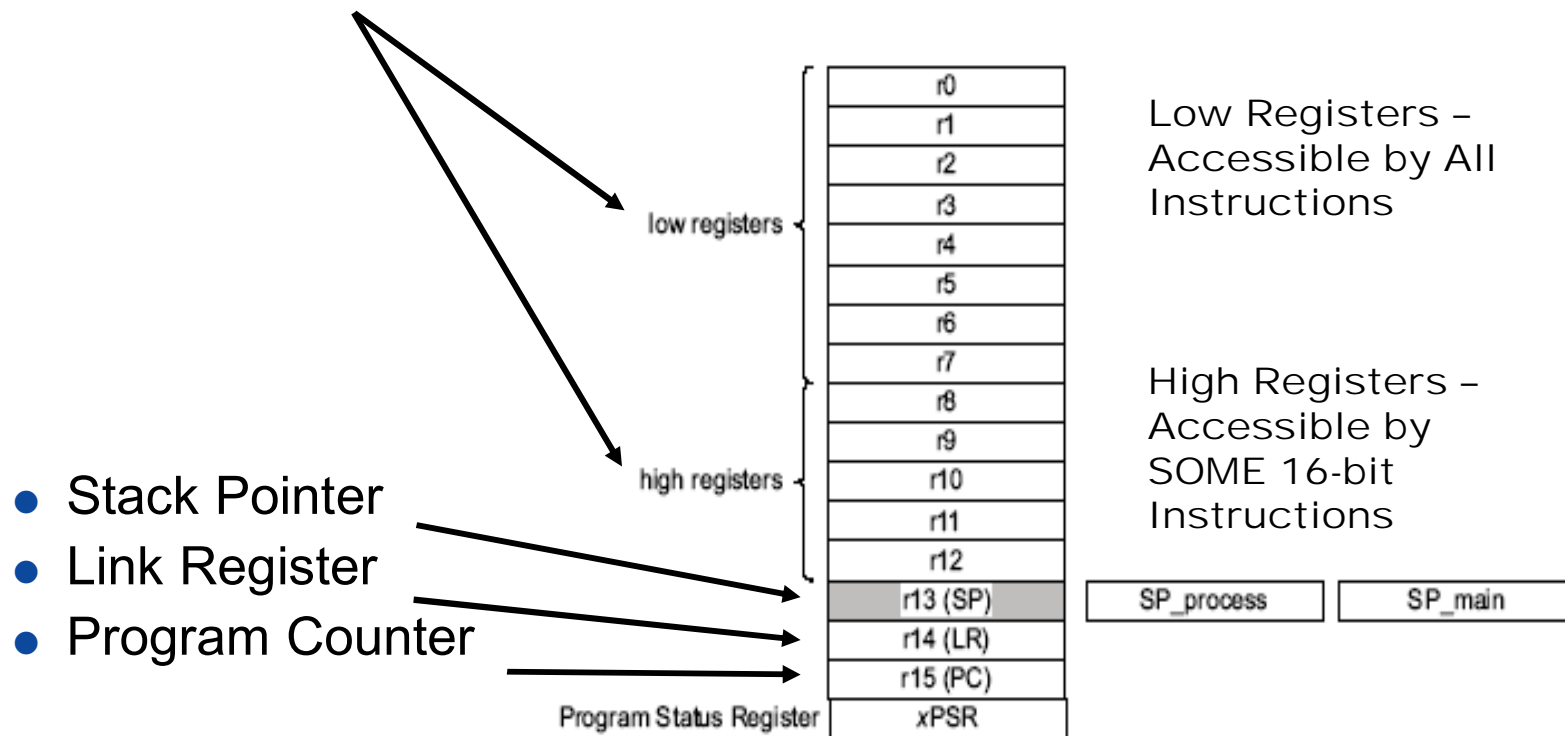
■ Modes

- Thread Mode
 - Entered on Reset
 - Can Be Re-entered after Exception Return
- Handler Mode
 - Used for Handling Exceptions

Cortex-M1

Registers

- 17 Registers
 - 13 General-Purpose Registers



Cortex-M1

SP, LR and PC Registers

■ Stack Pointer

- Auto-aligned to Word Boundary
- Has Banked Register Aliases `SP_process` and `SP_main`
- Handler Mode Always Uses `SP_main`, but You Can Configure Thread Mode to Use Either `SP_main` or `SP_process`

■ Link Register

- Receives Subroutine Return Address from PC when *Branch and Link* (BL) Instruction Is Executed
- LR Is also Used for Exception Return
- At All Other Times, Treat R14 as General-purpose Register

■ Program Counter

- Auto-aligned to Halfword Boundaries

Cortex-M1

Special-Purpose Program Status Registers (xPSR)

- Processor Status at System Level is Broken into Three Categories
- Registers Can Be Accessed Individually or Two or Three at a Time Using MRS and MSR Instructions
- Application PSR
 - Contains Condition-code Flags
 - Before Entering Exception, Processor Saves Condition-code Flags on the Stack
 - Accessed with MSR and MRS Instructions
- Interrupt PSR
 - Contains Interrupt Service Routine (ISR) Number Current Exception
- Execution PSR (EPSR)
 - Contains the Thumb state bit (T-bit)
 - Not Directly Accessible except in Debug State
 - All Fields Read as Zero using an MRS Instruction
 - MSR Instruction Writes Are Ignored
- On Entering an Exception, the Processor Saves Combined Information from the Three Status Registers on the Stack

Cortex-M1

Exception Types and Priorities

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	This exception type cannot be: <ul style="list-style-type: none">masked or prevented from activation by any other exceptionpre-empted by any other exception other than Reset.	Asynchronous
3	Hard Fault	-1	All classes of Fault.	Synchronous or asynchronous
4-10	-	-	Reserved.	-
11	SVC	Configurable	System service call using the 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
16-47	External Interrupt	Configurable	Asserted from outside the processor or pended by software.	Asynchronous

Servicing an Exception

- Push 8 Registers (xPSR, ReturnAddress(), R0, R1, R2, R3, R12, and LR) on Selected Stack
- Read Vector from Appropriate Vector Table Entry
 - Example: $(0x0) + (\text{exception_number} * 4)$
 - Only after ALL EIGHT Registers in Previous Step Are Pushed onto Stack
- On Reset Only, Update SP_main from First Entry in the Vector Table
 - Other Exceptions Do Not Modify SP_main this Way
- Update PC with Vector Table Read Location
 - No Other Late-arriving Exceptions Can Be Processed until the First Instruction of This Exception Starts to Execute
- Set LR to EXC_RETURN to Exit from Exception



Cortex-M1 Instruction Set

ARM Instructions

- 32-bit Instruction Length
- 36 Instruction Formats
- *ALL ARM Instructions Can Be Conditional!*

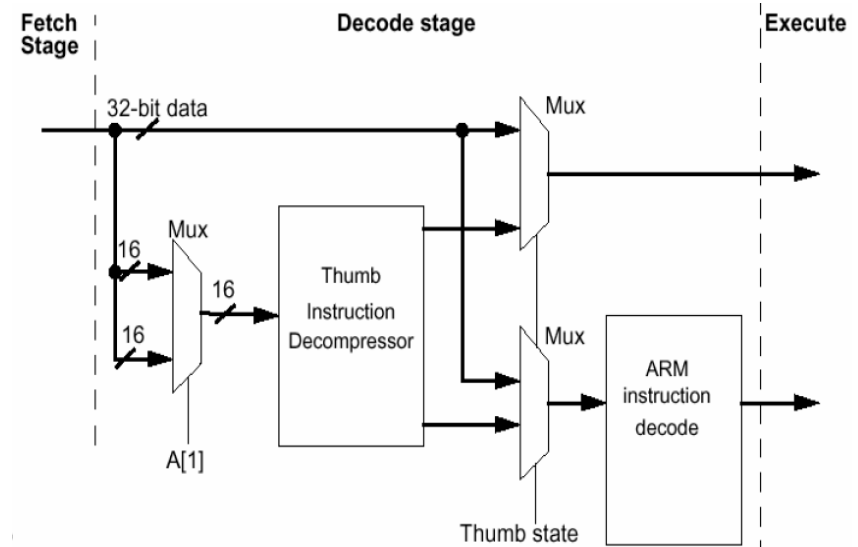
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond	0	0	1	Opcode				S	Rn	Rd	Operand 2																	<i>Data Processing / PSR Transfer</i>				
Cond	0	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm	<i>Multiply</i>														
Cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rn	1	0	0	1	Rm	<i>Multiply Long</i>															
Cond	0	0	0	1	0	B	0	0	Rn	Rd	0	0	0	0	1	0	0	1	Rm	<i>Single Data Swap</i>												
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn	<i>Branch and Exchange</i>			
Cond	0	0	0	P	U	0	W	L	Rn	Rd	0	0	0	0	1	S	H	1	Rm	<i>Halfword Data Transfer: register offset</i>												
Cond	0	0	0	P	U	1	W	L	Rn	Rd	Offset				1	S	H	1	Offset	<i>Halfword Data Transfer: immediate offset</i>												
Cond	0	1	1	P	U	B	W	L	Rn	Rd	Offset																<i>Single Data Transfer</i>					
Cond	0	1	1																	1						<i>Undefined</i>						
Cond	1	0	0	P	U	S	W	L	Rn	Register List																	<i>Block Data Transfer</i>					
Cond	1	0	1	L	Offset																							<i>Branch</i>				
Cond	1	1	0	P	U	N	W	L	Rn	CRd	CP#	Offset															<i>Coprocessor Data Transfer</i>					
Cond	1	1	1	0	CP	Opc	CRn	CRd	CP#	CP	0	CRm															<i>Coprocessor Data Operation</i>					
Cond	1	1	1	0	CP	Opc	L	CRn	Rd	CP#	CP	1	CRm														<i>Coprocessor Register Transfer</i>					
Cond	1	1	1	1	Ignored by processor																							<i>Software Interrupt</i>				

Thumb Instructions

- 16-bit Instruction Length

- Maps 32-bit Instructions into 16-bit Instructions
- Thumb Instructions Are Most-often-used 32-bit Instructions
- Thumb Instructions Transparently Expand in Real Time to Full 32-bit ARM Instructions

- **Thumb-instruction Decoder Placed in Pipeline**
- **Change to Thumb Mode Effected by Changing State of Multiplexers Feeding Instruction Decoders and Data Bus**



Thumb Instruction Set

- Instruction Word Length Shrank to 16-bits
 - Some Functionality Is Not Available
 - 19 Different Thumb Instruction Formats
- Instructions Have their Own Syntax
 - Each Instruction Has Native ARM Instruction Counterpart
- Only ONE Conditional Instruction!

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
1	0	0	0	Op	Offset5				Rs	Rd				Move shifted register									
2	0	0	0	1	1	I	Op	Rn/offset3		Rs	Rd				Add/subtract								
3	0	0	1	Op	Rd				Offset8				Move/compare/add /subtract immediate										
4	0	1	0	0	0	0	Op	Rs				Rd				ALU operations							
5	0	1	0	0	0	1	Op	H1	H2	Rs/Hs				Rd/Hd				Hi register operations /branch exchange					
6	0	1	0	0	1	Rd				Word8				PC-relative load									
7	0	1	0	1	L	B	0	Ro				Rb				Rd				Load/store with register offset			
8	0	1	0	1	H	S	1	Ro				Rb				Rd				Load/store sign-extended byte/halfword			
9	0	1	1	B	L	Offset5				Rb				Rd				Load/store with immediate offset					
10	1	0	0	0	L	Offset5				Rb				Rd				Load/store halfword					
11	1	0	0	1	L	Rd				Word8				SP-relative load/store									
12	1	0	1	0	SP	Rd				Word8				Load address									
13	1	0	1	1	0	0	0	0	S	SWord7				Add offset to stack pointer									
14	1	0	1	1	L	1	0	Rlist				Push/pop registers											
15	1	1	0	0	L	Rb				Rlist				Multiple load/store									
16	1	1	0	1	Cond				Soffset8				Conditional branch										
17	1	1	0	1	1	1	1	1	Value8				Software Interrupt										
18	1	1	1	0	0	Offset11				Unconditional branch													
19	1	1	1	1	H	Offset				Long branch with link													



ARM Code vs. Thumb Code

Comparison

- Performance
 - 32-bit Memory
 - ARM Code Is 40% Faster than Thumb Code
 - 16-bit Memory
 - Thumb Code Is 45% Faster than ARM Code
- Code
 - Thumb Code Requires 65% – 70% of the Space of ARM Code
 - Thumb Code Uses 40% More Instructions than ARM Code

- Thumb Code Has Higher Density and Better Performance with 16-bit Memory
- ARM Code Has Better Performance with 32-bit Memory

What's Thumb-2?

- Remember – 16-bit Thumb Instructions Are Mapped into 32-bit ARM Instructions for Execution
 - Some ARM Special Features Were NOT Supported in Thumb
 - ARM Wanted to Eliminate ...
 - ... Tradeoff of Size vs. Speed
 - ... Confusion about Which Instruction Set to Select
- Thumb-2 Combines 16- and 32-bit Instructions in a Single Instruction Set
 - Allows Mixing of Instructions without Mode Switching
 - Halfword Pairs of Instructions Are Inserted in Thumb Instruction Stream
- Thumb-2 Features
 - Mostly 16-bit Instructions
 - 5% Better Code Density than Existing Thumb Code
 - 2-3% Faster than Thumb Code
 - Code Density – 74% of ARM Code Density
 - Code Performance – 98% of ARM Code Performance

Thumb-2 Instruction Set

Changes vis-à-vis Thumb

■ ~~IF-THEN Instruction~~

- Additional C **NOT in Cortex M1!**

■ ~~New Bit Instructions~~

- Bit Fields
- Bit Reversal

■ ~~New Branch Instructions~~

- Table Branch
 - Facilitates SIMD Execution
- Compare and Branch ('Branch if Zero')

■ ~~Coprocessor Access Instructions~~

- Don't Need to Mix Thumb Code and ARM Code

■ ~~16-bit Constants~~

```
LDREQ r0, [r1]
LDRNE r0, [r2]
ADDEQ r0, r3, r0
ADDNE r0, r4, r0
```

```
BNE L1
LDR r0, [r1]
ADD r0, r3, r0
B L2
L1
LDR r0, [r2]
ADD r0, r4, r0
L2
```

`.thumb` → Thumb-2

```
ITETE EQ
LDR r0, [r1]
LDR r0, [r2]
ADD r0, r3, r0
ADD r0, r4, r0
```

Thumb-2 32-bit Instructions

Summary

■ ARM-like

- Data Processor **Most Not Supported in Cortex-M1!**
- DSP and M
- Load and Store instructions
- Branch Instructions
- System Control – BXJ, RFE, SRS, etc.
- Coprocessor (VFP, MOVE™, etc.)

■ New

- Bit-field Insert/Extract/Clear – BFI, {S|U}BFX, BFC
- Bit Reverse – RBIT
- 16-bit Immediate instructions – MOVW, MOVH
- Table Branch – TB{B|H} [Rbase, Rindex]
- Additional Memory System Hints – PLD

Cortex-M1 Instruction Set

Overview

- Branch instructions (6)
 - B Conditional Branch, Unconditional, with Link...
 - BX Branch and Exchange instruction set.
- Data processing instructions
 - ADD, CMP, MOV, MULT, SUB...
- Load and store register instructions
- Load and store multiple instructions
 - Two instructions, LDMIA & STMIA, to support block copy
- Exception generating instructions
 - Software Interrupt (SWI) instruction used to cause a SWI exception to occur
 - Main mechanism in the Thumb instruction set by which User mode code can make calls to privileged Operating System code

Cortex-M1 Instruction Set

16-bit Instructions

- **CPS** – Change Processor State
 - Enables/Disables Interrupts
- **NOP** – No Operation
 - Certain New Thumb-2 Instructions Are Ignored (**IF-THEN**)
- **SEV** – Send Event
 - Event Signaled to All CPUs in Multiprocessor System
- **WFE** – Wait for Event
 - Suspend Execution until Reset, Exception, etc. Occurs
- **WFI** – Wait for Interrupt
 - Suspend Execution until Interrupt Occurs
- **YIELD** – Yield (Multithreaded Systems)
 - Software Can Indicate that Task Can Be Swapped Out



Actel's Cortex-M1

Cortex-M1

Configurable Options

- Full Range of Configurable Options Will Be Available ...

Feature	Configurable Option
Debug	Debug Functionality Can Be Included or Excluded
Interrupts	1, 4, 8, 16 or 32 Interrupts (0 Interrupts Not Supported)
Instruction TCM	0KB (None), 1KB, 2KB, and Powers of 2 to 1MB
Data TCM	0KB (None), 1KB, 2KB, and Powers of 2 to 1MB
Multiplier	Normal (3-cycle Execution) or Small (33 Cycles)
OS Extensions	Present or Absent
Endianness	Little-endian or Big-endian (BE8)

- ... Actel Will Deliver Restricted Range of Configurations
 - Fixed, Black-box Implementations with Predefined Configurations (Similar to CoreMP7)

Cortex-M1 Configurations

- Current Release – v2.6

Variant	Configuration
M1AFSxxxxDebug M1A3PxxxxDebug M1A3PExxxxDebug M1A3PLxxxxDebug M1AGLxxxxDebug M1AGLExxxxDebug	Debug Support (2 breakpoints, 1 watchpoint) No TCMs, Small Multiplier, 1 Interrupt, No OS Extensions, Little-endian Only
M1AFSxxxxNodebug M1A3PxxxxNodebug M1A3PExxxxNodebug M1A3PLxxxxNodebug M1AGLxxxxNodebug M1AGLExxxxNodebug	No Debug Capability No TCMs, Small Multiplier, 1 Interrupt, No OS extensions, Little-endian Only

- Future Releases

- More Variants, More Devices

Supported Devices and Availability

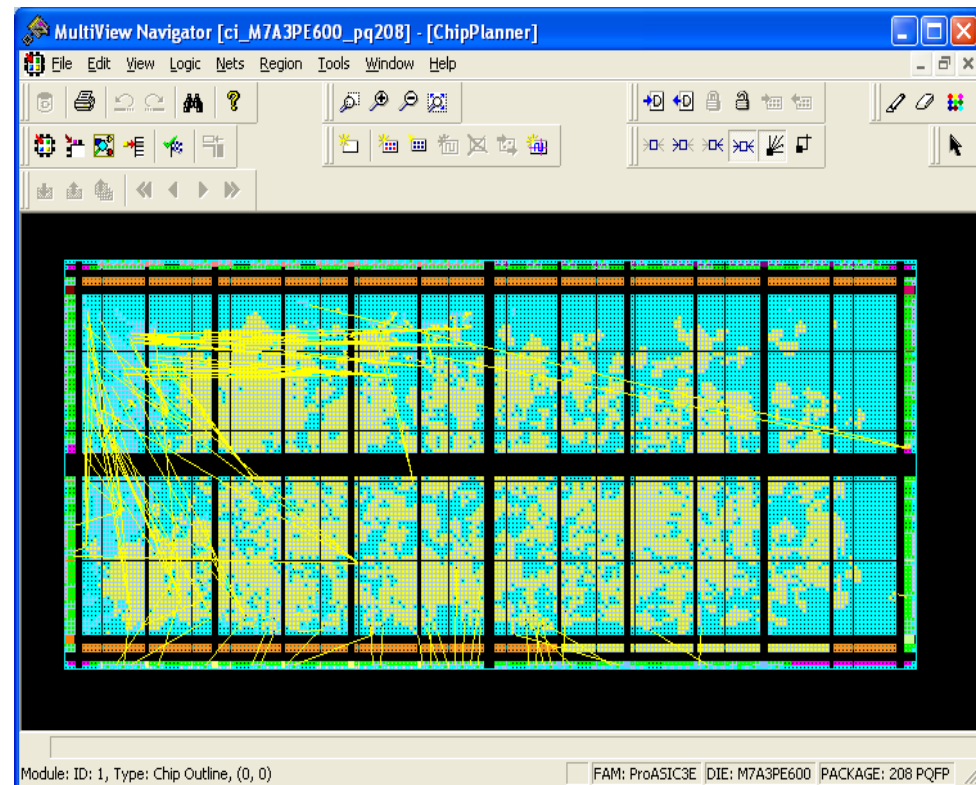
M1-enabled Devices	250	400	600	1000	E1500	E3000
M1 IGLOO	✓		✓	✓		✓
M1 ProASIC3	✓	Planned	✓	✓	✓	✓
M1 ProASIC3L			✓	✓		Planned
M1 Fusion	✓		✓		✓	

Information updated March 2009

- ARM Cortex-M1 processor Actel M1 devices
 - One user selectable option: with or without debug.
 - Pre-configured settings
 - 0K ITCM, 0K DTCM, small multiplier, little-endian,
 - No OS extensions, and 1 interrupt
- Fully Configured M1AFS1500 - 2009
 - 8kB ITCM, 4kB DTCM, 16 interrupts, OS extensions, fast multiplier, little-endian.
 - Contains upgraded BFM

Optimized Cortex-M1 in M1 Fusion

- Optimized Firm-macro Implementation
 - Optimized for M1 Fusion Devices
 - Porous Implementation Placed in FPGA Fabric by Actel
- Simplified Routing of User-added IP
- Enables Rapid Design Development
- Cortex-M1 Black Box
 - M1 Functionality
 - User Can Add IP and Program into M1 Fusion
 - Timing Shell and BFM Included



Cortex-M1

Utilization and Performance

Device	Variant	v2.6		
		Frequency (MHz)	Area	Utilization
M1AFS250-2	No-debug	62.26	4353	70.85%
M1AFS600-2	No-debug	60.96	4353	31.49%
	Debug	55.14	7282	51.68%
M1AFS1500-2	No-debug	62.69	4353	11.34%
	Debug	56.39	7282	18.96%
M1AGL250V2	No-debug	25.56	4353	70.85%
M1AGL250V5	No-debug	43.04	4353	70.85%
M1AGL600V2	No-debug	25.46	4353	31.49%
	Debug	24.40	7282	52.68%
M1AGL600V5	No-debug	40.23	4353	31.49%
	Debug	39.90	7282	52.68%
M1AGL1000V2	No-debug	24.74	4353	17.71%
	Debug	23.53	7282	29.63%
M1AGL1000V5	No-debug	40.87	4353	17.71%
	Debug	40.57	7282	29.63%
M1AGLE3000V2	No-debug	24.89	4353	5.78%
	Debug	24.54	7282	9.68%
M1AGLE3000V5	No-debug	36.89	4353	5.78%
	Debug	38.05	7282	9.68%
M1A3P250-2	No-debug	62.46	4353	70.85%
M1A3P600-2	No-debug	62.63	4353	31.49%
	Debug	59.24	7282	52.68%
M1A3P1000-2	No-debug	62.18	4353	17.71%

Cortex-M1 Summary

- New FPGA-targeted Processor from ARM
- Small and Fast
 - 4300 Tiles, 62 MHz without Debug
- Some Nice Features
 - Thumb-2 Instruction Set
 - Mostly 16-bit Instructions – Good Code Density)
 - Good Interrupt Support (Closely-coupled NVIC)
- Easier to Program than CoreMP7
 - Porting CoreMP7 Software Fairly Straightforward
- V2.6 Available Now



System-on-Chip and SmartDesign Introduction to AMBA

System-on-Chip and SmartDesign

Agenda

- AMBA Bus Architectures
- SmartDesign and IP Cores
- Using SmartDesign
- IP Database
- Testing



AMBA Bus Architectures

Bus: definition

- Set of *wires* connected to many *bus actors*
- Wires are grouped in functional groups:
 - *Address*: bus actor identifiers
 - *Data*
 - *Control*: physical support for the protocol
- Wires are shared
 - Access protocols are required
 - Control logic + control wires embody the protocol

What Is AMBA?

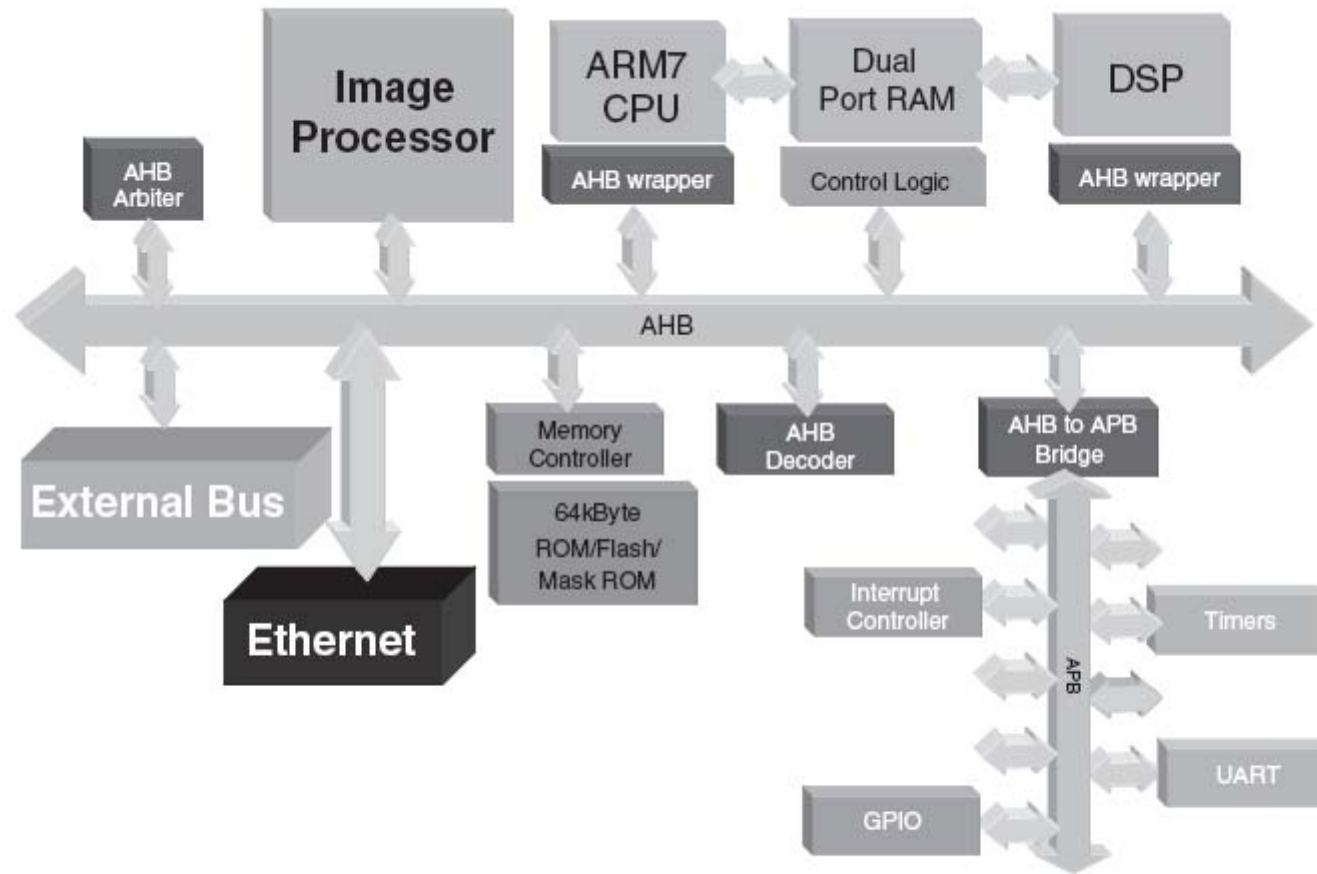
- Open-standard, On-chip Bus Specification
 - Interconnection & Management of SoC Functional Blocks
 - Widely-adopted Protocol and Standard with ARM Processors
- Three distinct buses
 - AHB (the Advanced High-performance Bus).
 - High-performance system backbone bus.
 - Multi-Master, Split Transactions, Burst Transfers and Other Modes
 - Example Components – Cortex-M1, CoreMP7, and Memory Controllers
 - ASB (the Advanced System Bus).
 - An alternative system bus.
 - APB (the Advanced Peripheral Bus).
 - Minimal power consumption.
 - Reduced interface complexity.
 - Example Components – Watchdog and UART
- AHB & APB AMBA Protocols Supported by SmartDesign
- When Both AHB and APB Are Used in System, They Must Be Bridged (AHB2APB Bridge Provided)

What Is AMBA?

- Three distinct buses
 - AHB (the Advanced High-performance Bus).
 - High-performance system backbone bus.
 - ASB (the Advanced System Bus).
 - An alternative system bus.
 - APB (the Advanced Peripheral Bus).
 - Minimal power consumption. □ Reduced interface complexity.

AMBA Subsystem

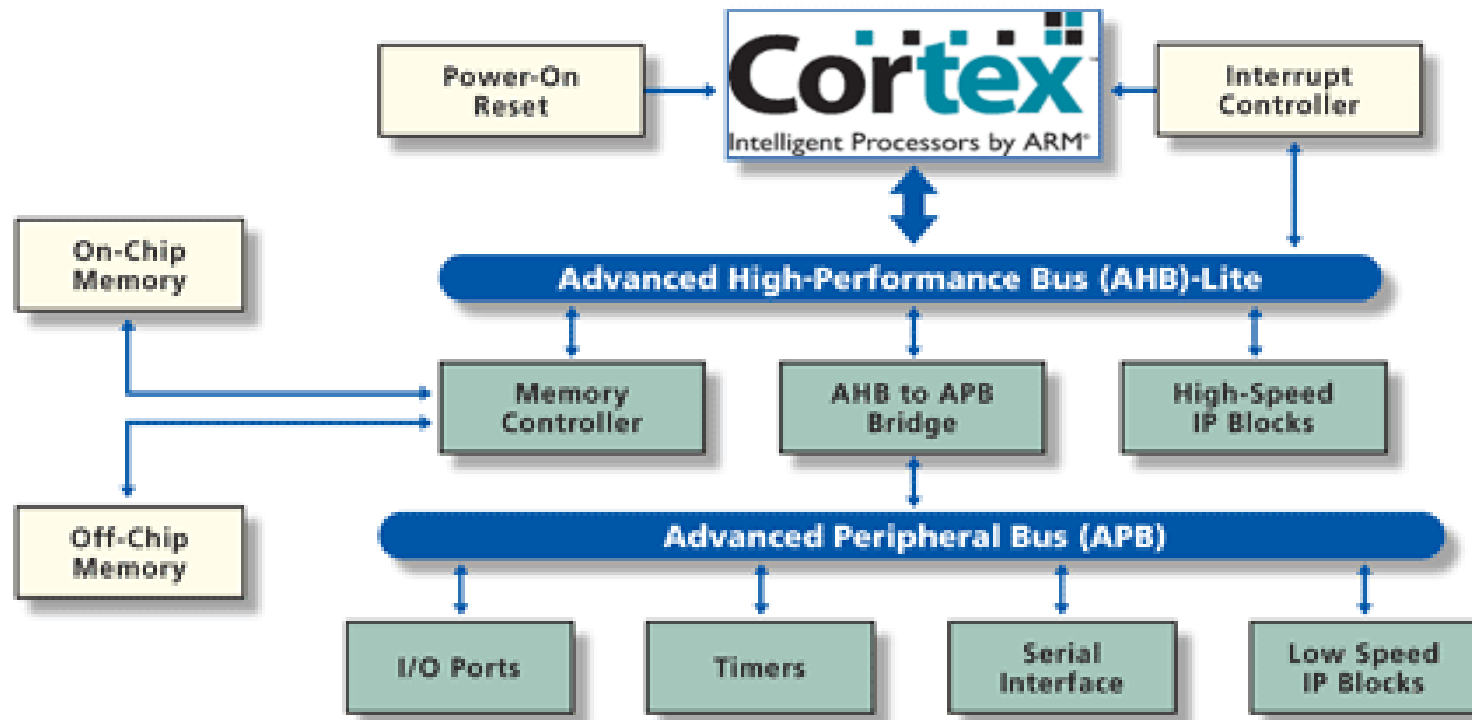
Block Diagram



“typical” AMBA system

AMBA Subsystem

Block Diagram



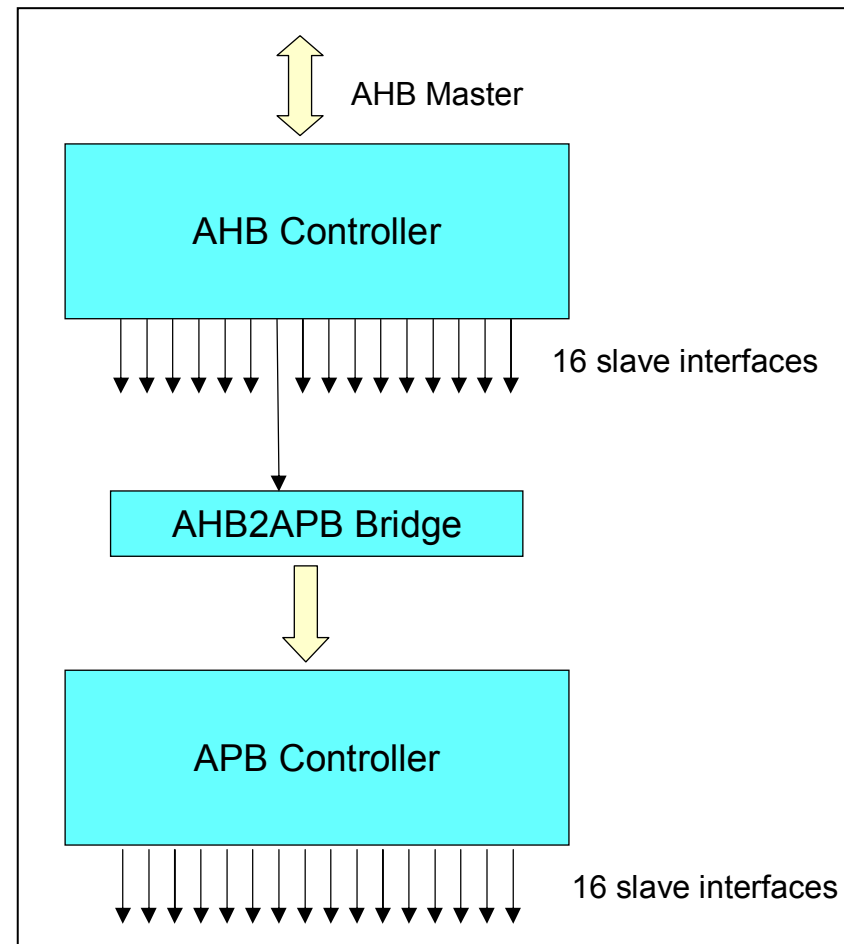
ASB Bus

Features

- First-generation system bus that evolved from ARM7TDMI bus protocol
- Supports pipelining, bursts, and multiple bus masters
- The four bus agents in ASB are:
 - Arbiter: Implements a simple request/grant structure to support multiple bus masters
 - Decoder: Centralized address decoder to determine which slave is responsible for servicing a bus transaction
 - Master: Initiates reads and writes on the bus
 - Slave: Responds to master initiated reads and writes
- Drawbacks
 - Use of double-edging clock
 - Bi-directional data bus

AMBA Bus Fabric

- Bus Fabric Components
 - AHB Controller
 - AHB to APB Bridge
 - APB Controller
- AHB and APB Controllers Incorporate All Bus Switching and Decoding Logic
- Two AHB Controller Configurations
 - Single Master (AHB-Lite)
 - Multi-Master (3 Masters)
 - Supports Full AHB Slaves



AHB Bus

Features

- High Bandwidth
 - No Maximum Clock Frequency Specified
 - Designs up to 2 GHz!
- Burst Transfers
- Split Transactions
- Single-cycle Bus Master Handover
- Single-clock-edge Operation
- Non-tristate Implementation
- Support for Multiple Masters
- Bridging to APB Bus

AHB Bus

Burst vs. Split

■ Burst transaction

- Slave to increment (or decrement) address
- No need of sending many sequential addresses through bus
- Reduces power (computations at the slave)

■ Split transaction

- When a master transaction can potentially take a long time (i.e. communicating with APB peripherals).
- To avoid holding the bus for many cycles
- Masters issues a request, then releases the bus, and waits for notification, which can come many cycles after.

AMBA Bus Interface Signals

AHB Single-Master and AHB-Lite Buses

Name	Source	Description
HCLK Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of HCLK .
HRESETn Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0] Address bus	Master	The 32-bit system address bus.
HTRANS[1:0] Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0] Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.

Source: ARM Inc.

AMBA Bus Interface Signals

AHB Single-Master and AHB-Lite Buses

HBURST[2:0] Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
HPROT[3:0] Protection control	Master	<p>The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection.</p> <p>The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.</p>

Source: ARM Inc.

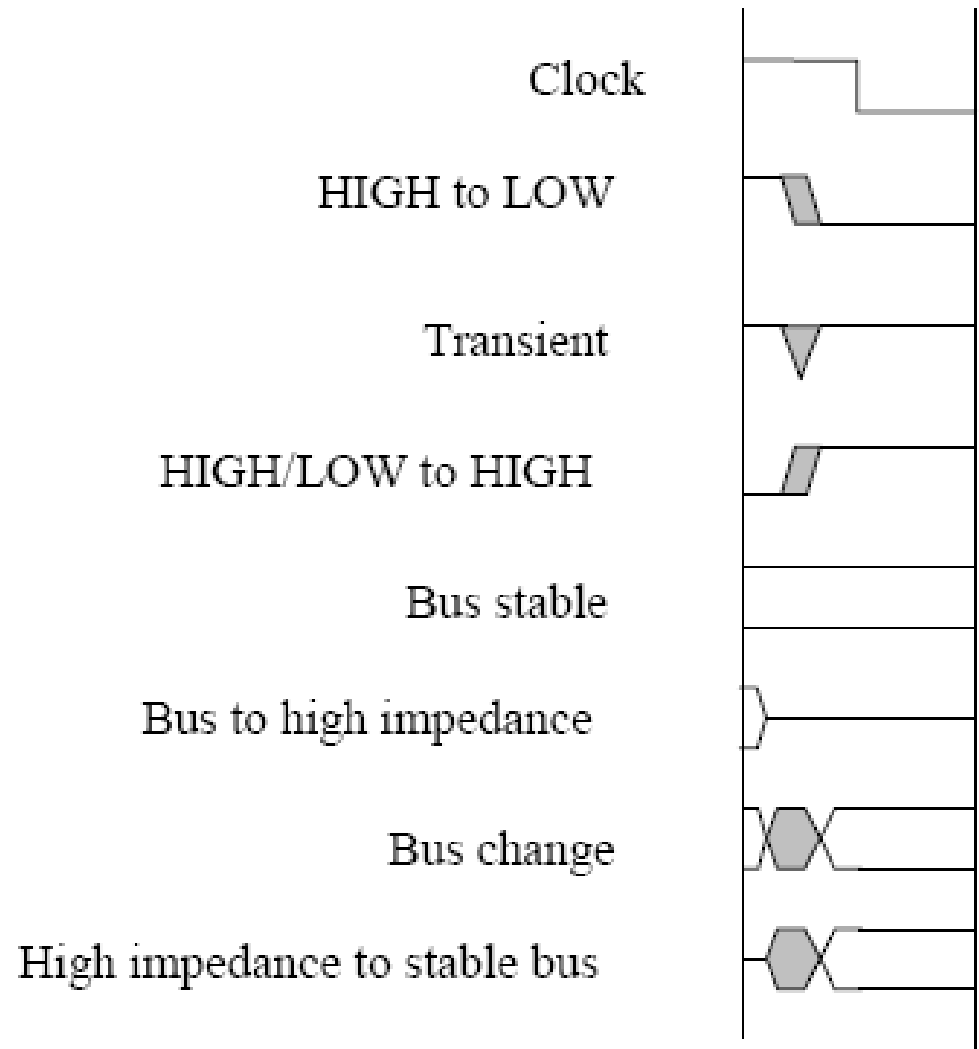
AMBA Bus Interface Signals

AHB Single-Master and AHB-Lite Buses

Name	Source	Description
HWDATA[31:0] Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELx Slave select	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
HRDATA[31:0] Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY Transfer done	Slave	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require HREADY as both an input and an output signal.
HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

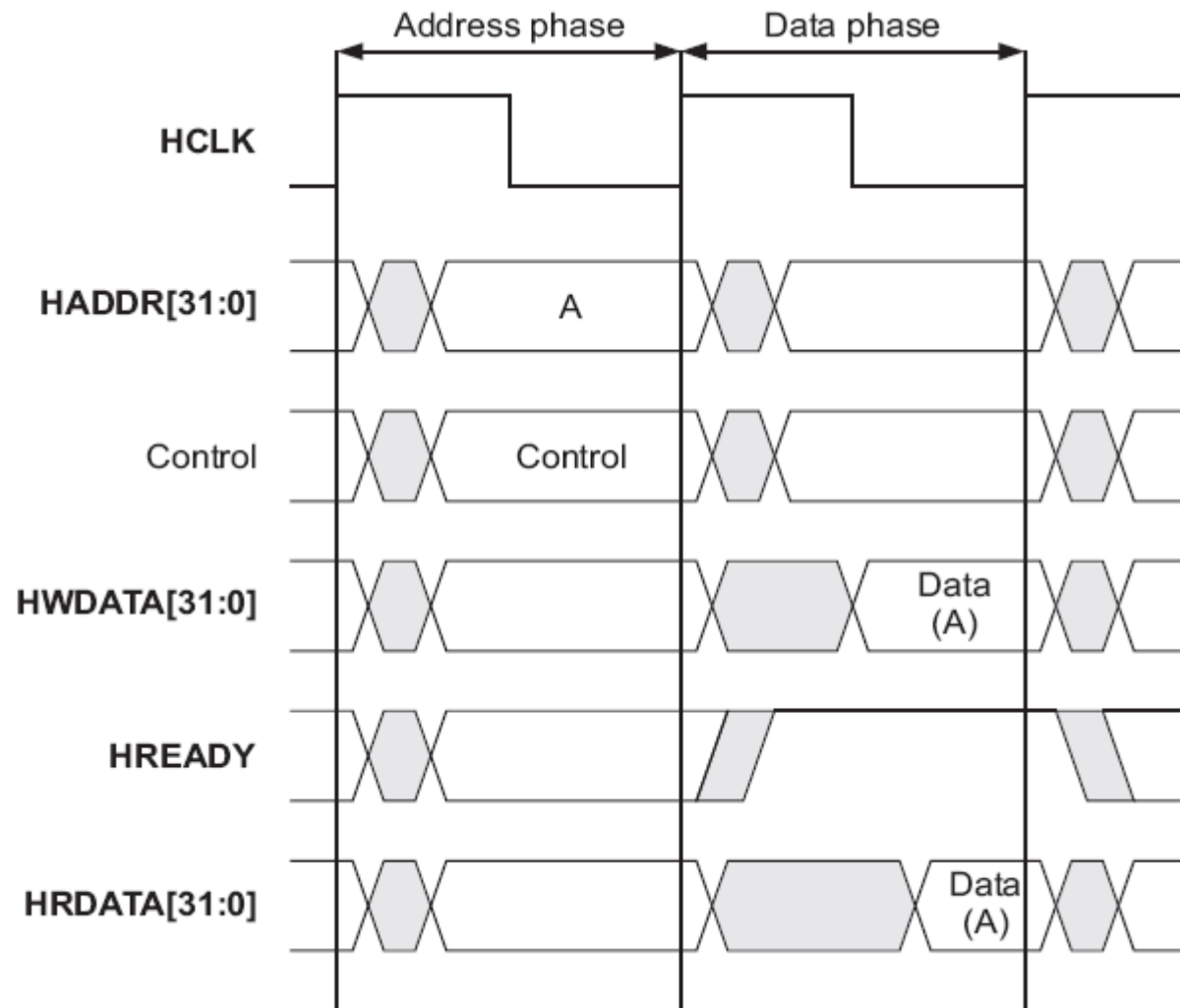
Source: ARM Inc.

Timing Diagram Conventions



Source: ARM Inc.

Simple AHB Transfer



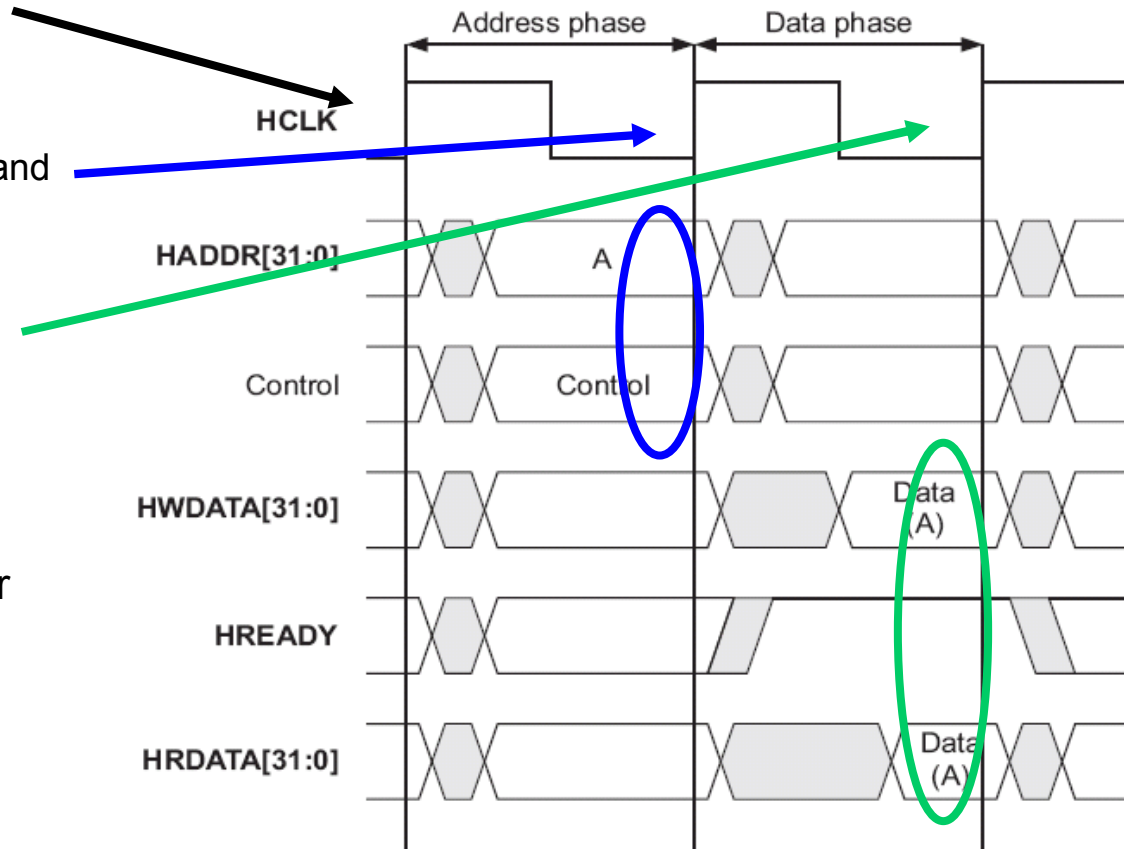
Source: ARM Inc.

AHB Transfer

No Wait States

- First Rising Edge of HCLK
 - Master Drives Address and Control Signals onto Bus after this Edge
- Second Rising Edge of HCLK
 - Slave then Samples Address and Control Information on Next Rising Edge of Clock
- Third Rising Edge of HCLK
 - Slave Can Start to Drive Appropriate Response
 - Sampled by Bus Master

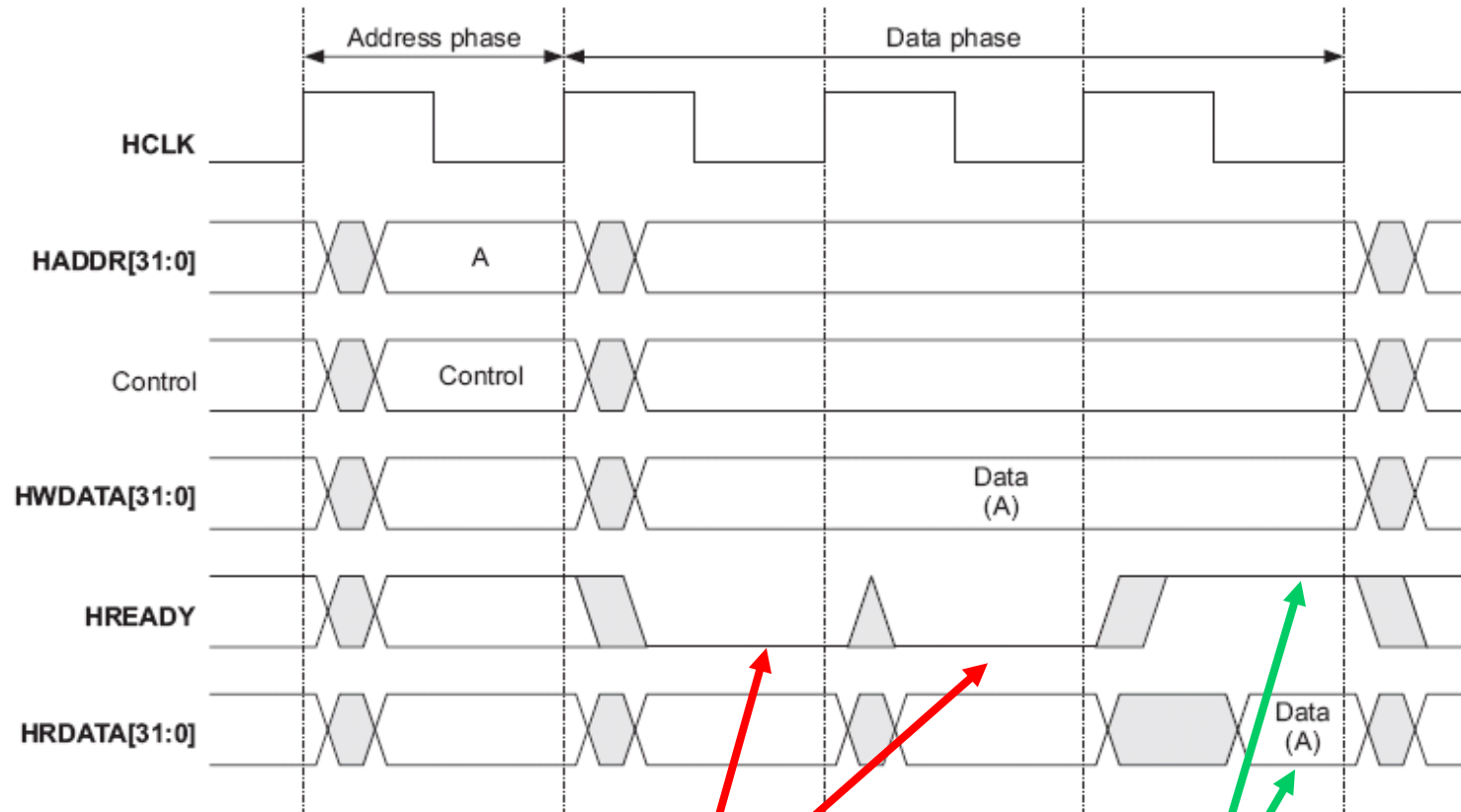
- Address Phase of Any Transfer Occurs during Data Phase of Previous Transfer
 - Pipelining Is Key Part of AHB Architecture



Source: ARM Inc.

AHB Transfer

Wait States



Source: ARM Inc.

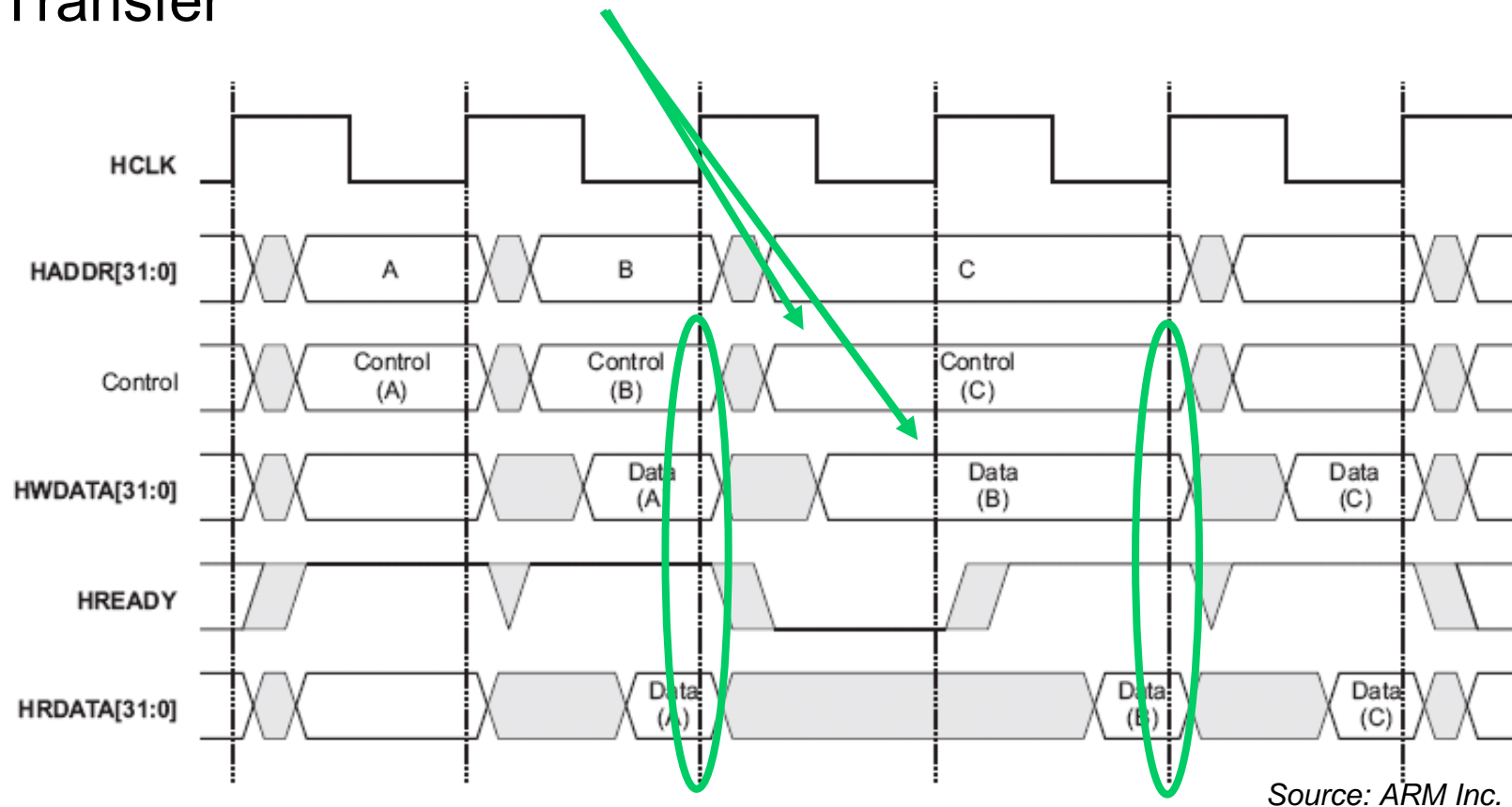
**Wait States
When HREADY = 0**

**Data Transferred
when HREADY = 1**

AHB Transfer

Pipelined Operations

- Control of Next Transfer Overlaps Data from Previous Transfer



AHB Bus

Burst Signal Encoding

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Source: ARM Inc.

- Note that Transfer Size Can Be Larger than 16
 - INCR Indicates Unspecified Length

AHB Bus

Transfer Type Encoding

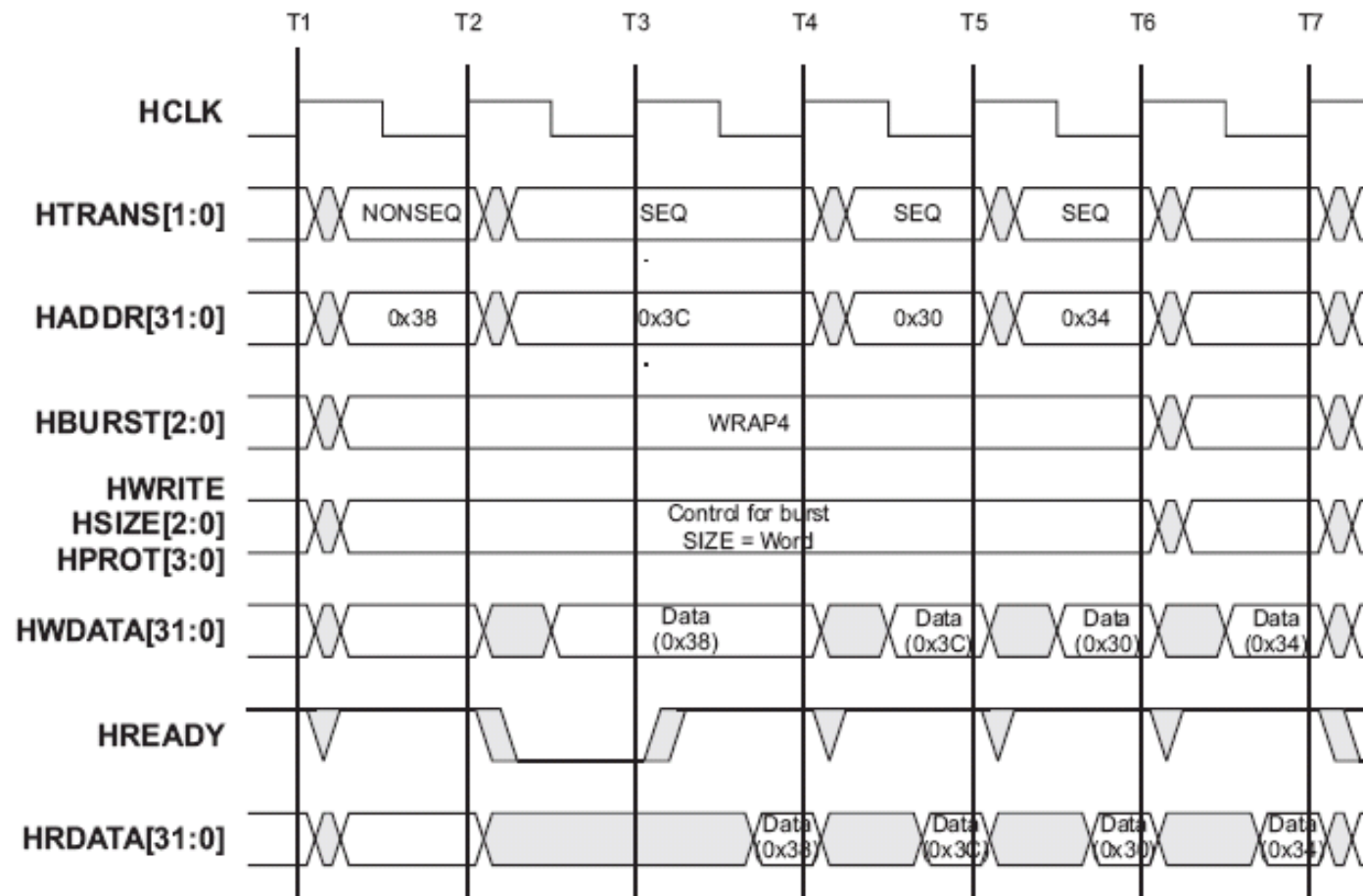
HTRANS[1:0]	Type	Description
00	IDLE	Indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be ignored by the slave.
Granted & unused		
01	BUSY	The BUSY transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfers. This transfer type indicates that the bus master is continuing with a burst of transfers, but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. The transfer should be ignored by the slave. Slaves must always provide a zero wait state OKAY response, in the same way that they respond to IDLE transfers.
Locks the bus but no transfer within a burst		
10	NONSEQ	Indicates the first transfer of a burst or a single transfer. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of one and therefore the transfer type is NONSEQUENTIAL.
Starts a Burst or a single transfer		
11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the size (in bytes). In the case of a wrapping burst the address of the transfer wraps at the address boundary equal to the size (in bytes) multiplied by the number of beats in the transfer (either 4, 8 or 16).
Continues a burst		

Source: ARM Inc.

AHB Bus

Four-beat Wrapping Burst

- Sequential Addresses 38, 3C, 30, 34

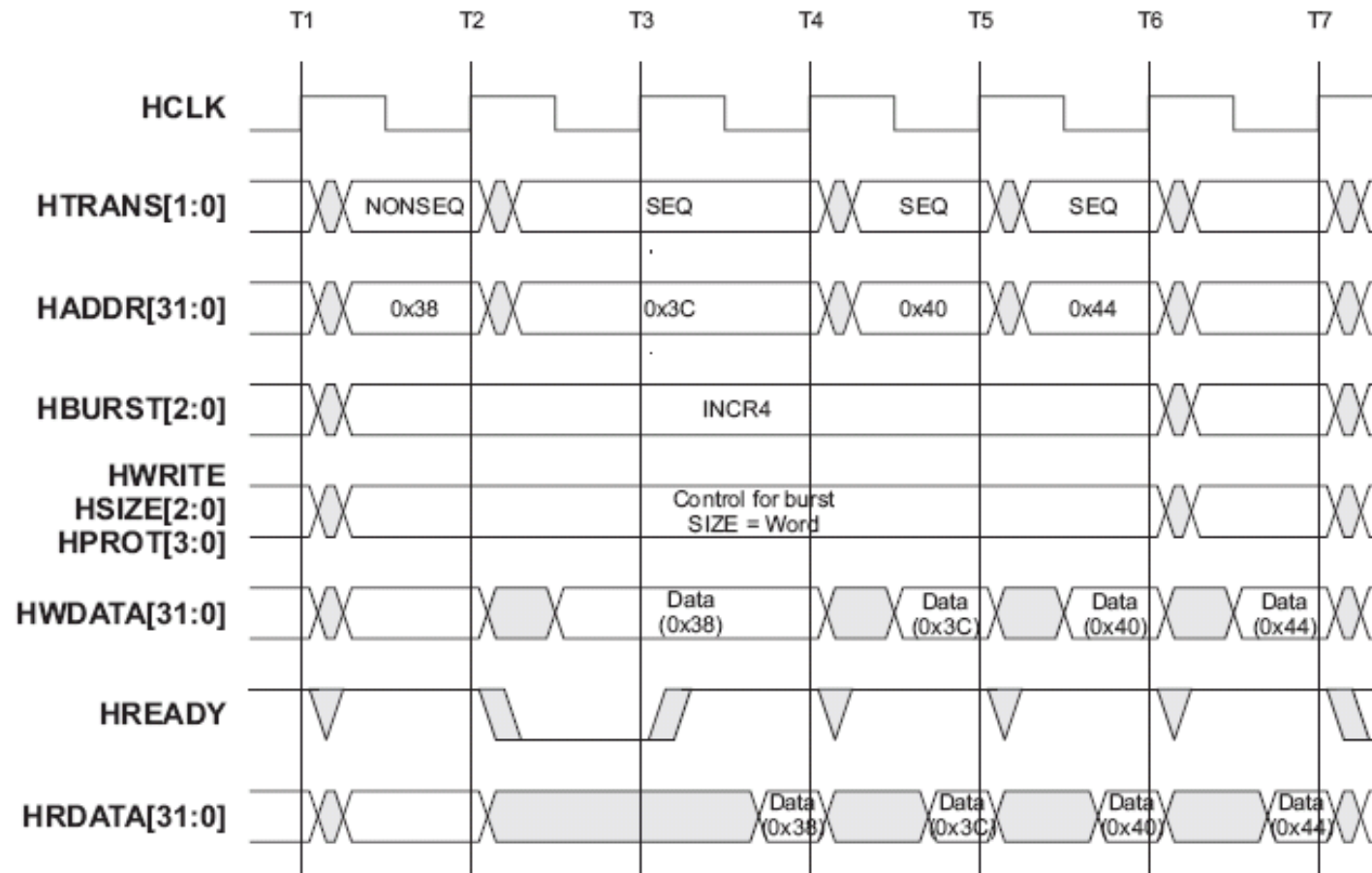


Source: ARM Inc.

AHB Bus

Four-beat Incrementing Burst

- Sequential Addresses 38, 3C, 40, 44

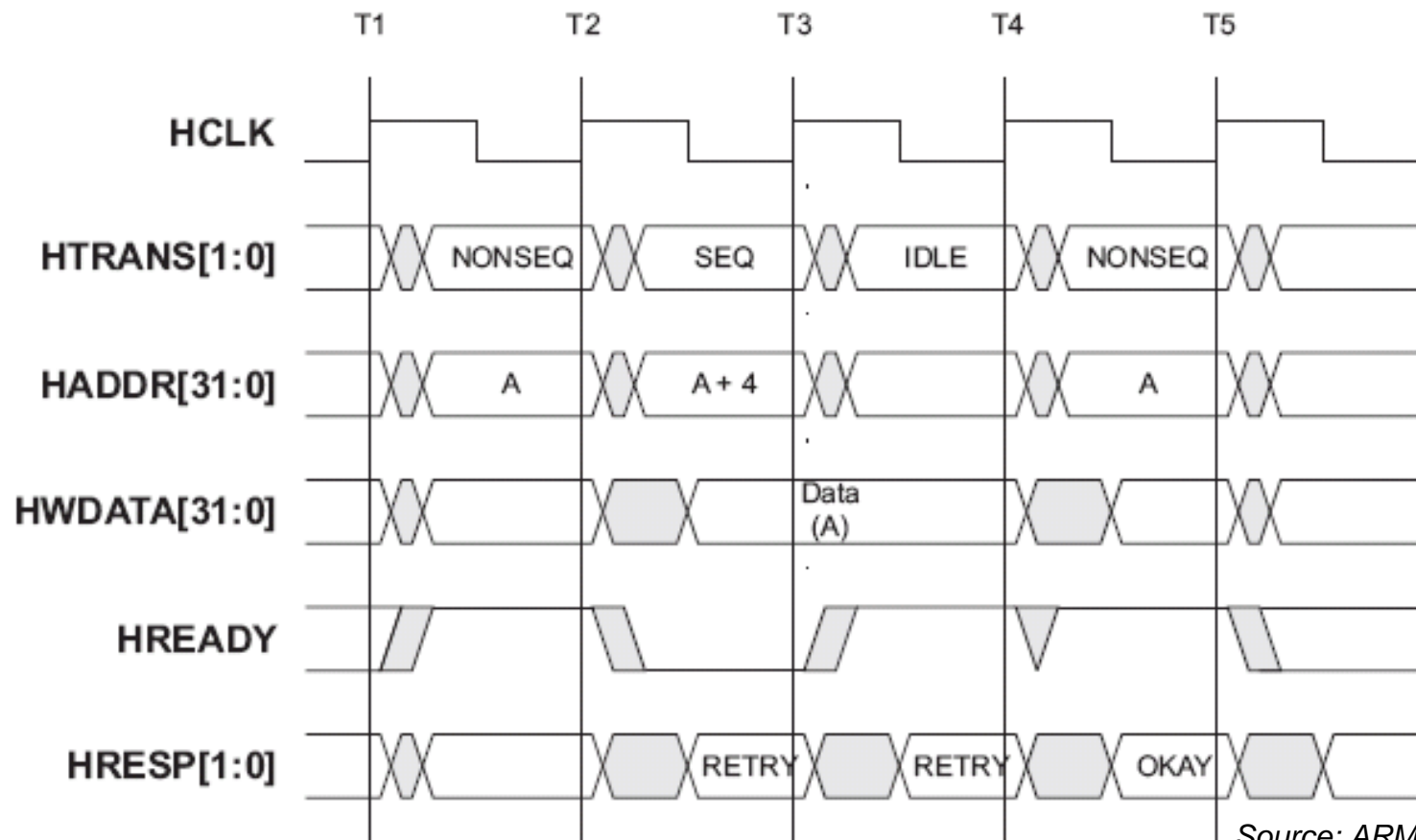


Source: ARM Inc.

AHB Bus

Transfer with Slave Retry Response

- Slave Indicates Response on HRESP(1:0)
 - Can Be OK, ERROR, RETRY, or SPLIT



Source: ARM Inc.

AHB Bus

SPLIT vs. RETRY Responses

- SPLIT and RETRY Responses BOTH Allow ...
 - ... Slaves to Release Bus when They Cannot Immediately Supply Data for Transfer
 - ... Transfer to Finish on Bus and therefore Higher-priority Master Can Access Bus
- Difference between SPLIT and RETRY ... How Arbiter Allocates Bus after SPLIT or RETRY Has Occurred

- RETRY
 - Arbiter Continues to Use Normal Priority Scheme
 - Only Masters with Higher Priority Get Bus Access
- SPLIT
 - Arbiter Adjusts Priority Scheme so that Any Other Master Requesting Bus Gets Access, even if Lower Priority
 - Transfer Completion Requires Informing Arbiter when Slave Has Data Available
 - Requires Extra Complexity in Both Slave and Arbiter, but Completely Frees Bus for Use by Other Masters
- Master Should Treat SPLIT and RETRY the SAME WAY
 - Should Continue to Request Bus and Attempt the Transfer until Successful Completion or ERROR Response Termination

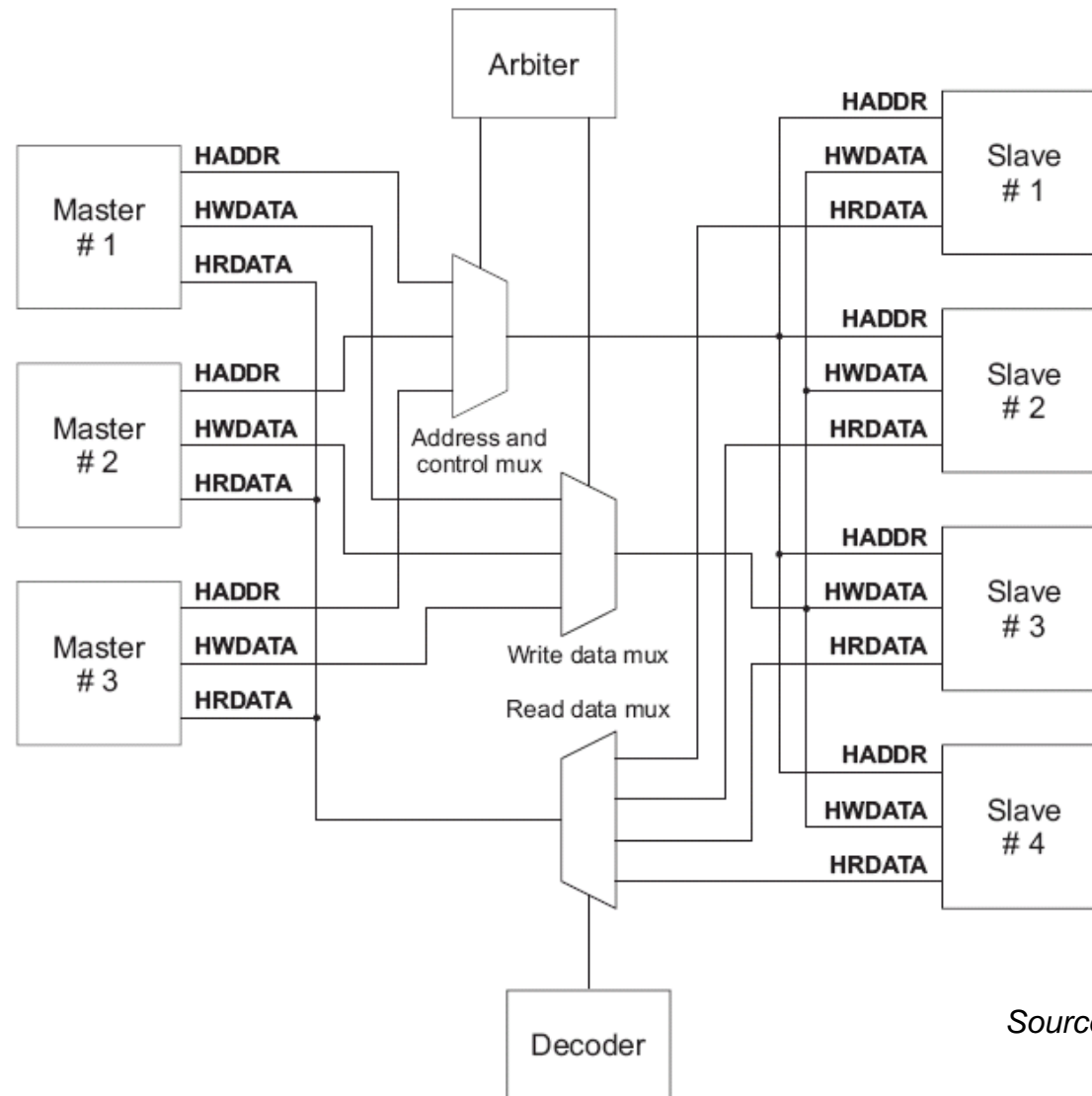
AMBA Bus Interface Signals

AHB Multi-Master Bus

Signal	Function
HBUSREQx	Request from Master x to Use Bus
HLOCKx	Indicates that Master Requires Locked Access to Bus
HGRANTx	Grant to Master x to Use Bus
HMASTx	Indication from Arbiter that Master x Is Using Bus
HSPLITx(15:0)	Indication from Slave x as to which Bus Masters May Re-Attempt Split Transaction

AHB Multi-Master Subsystem

Three Masters, Four Slaves



Source: ARM Inc.

CoreAHB

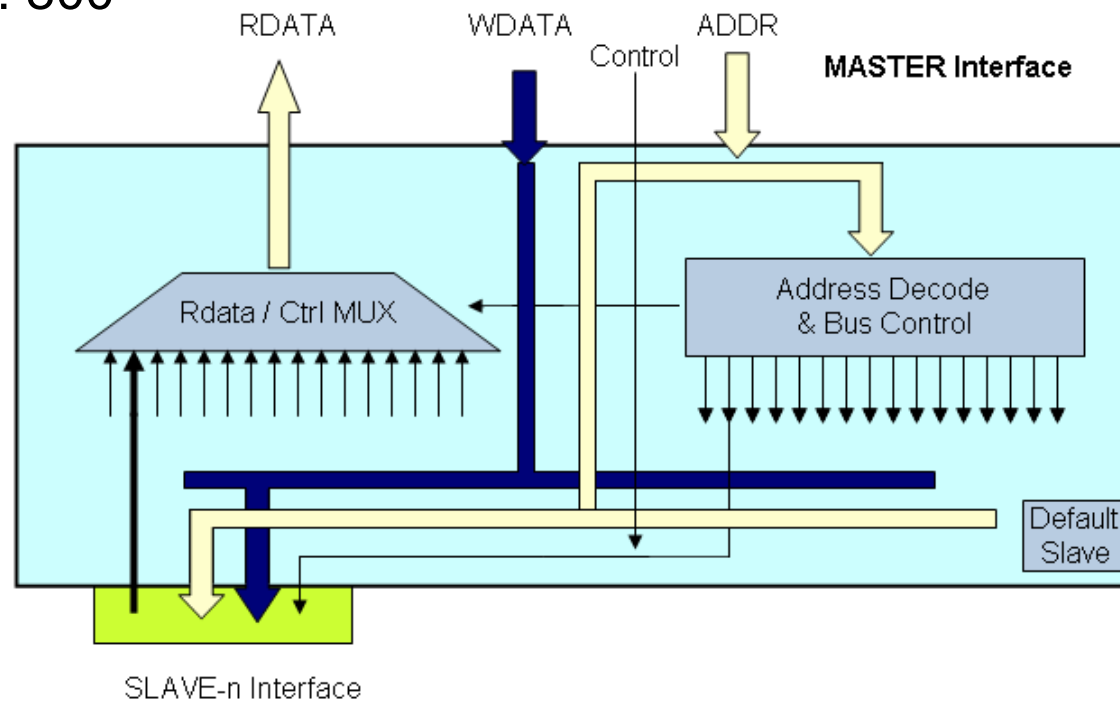
Arbitration

- CoreAHB Uses Simple Priority Arbitration Scheme
 - Master 3 Has Highest Priority
 - Master 0 which Is Internal 'Dummy' Master Is Next-highest
 - Master 2 Is Next
 - Master 1 (Default) Is Lowest Priority
 - This Is Normally Processor Master
- Master 0 Handles Several Situations
 - Example – Breaks Deadlock when Multiple Masters Are Locked in SPLIT Transactions

CoreAHBLite

Single Master

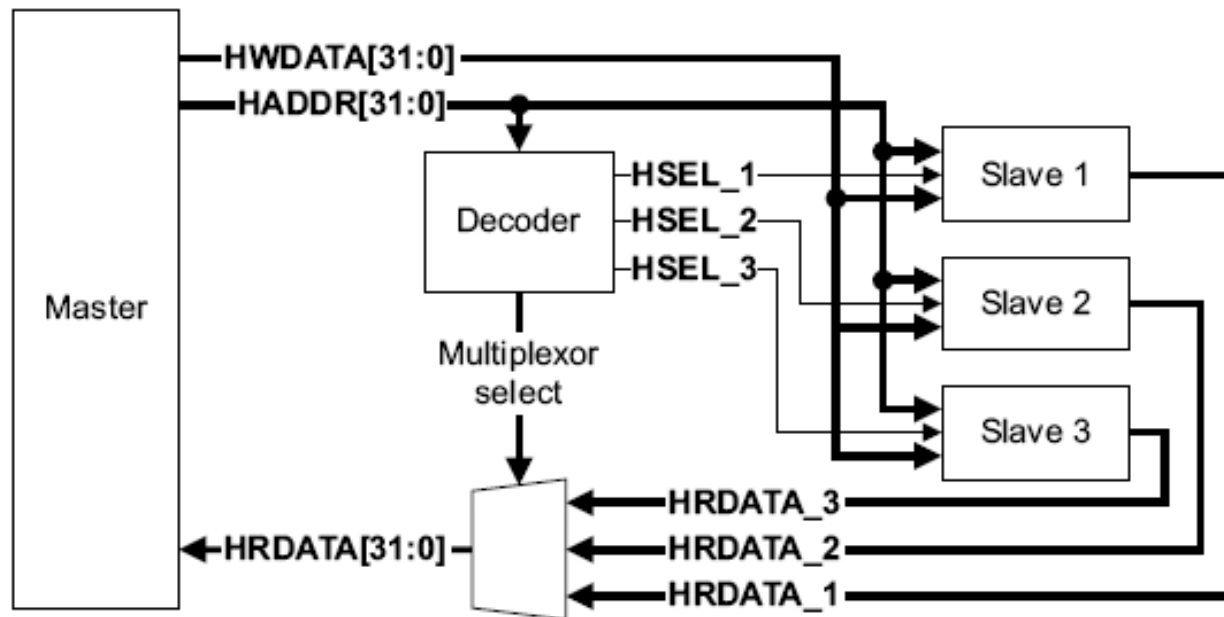
- Single AHB Master Interface
 - Default Interface for Cortex-M1
- 16 AHB Slave Interfaces
 - Each Is Allocated Fixed Memory Location
 - Two 'Standard' Locations
 - Mem at 0
 - Interrupt at Top of Memory
- Tile Count: 800



AHB-Lite System

Single Master, Three Slaves

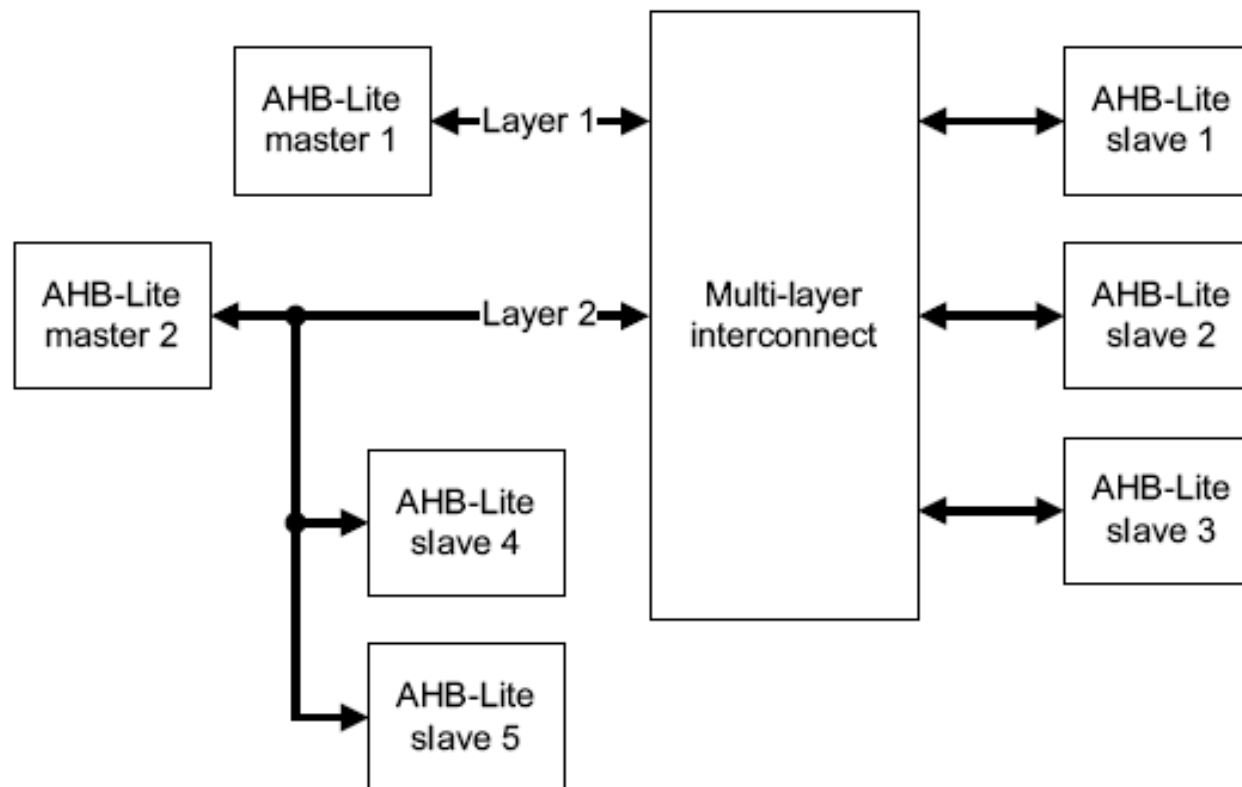
- CoreAHBLite Works Fine Here ...



AHB-Lite System

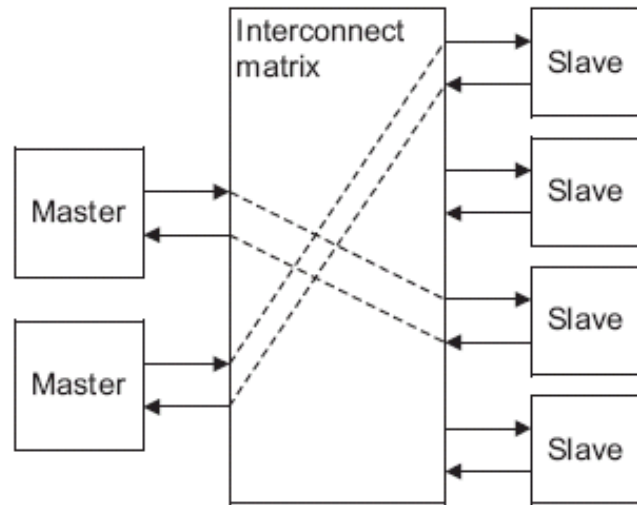
TWO Masters, Multiple Slaves

- ... BUT ... How Do We Implement THIS?

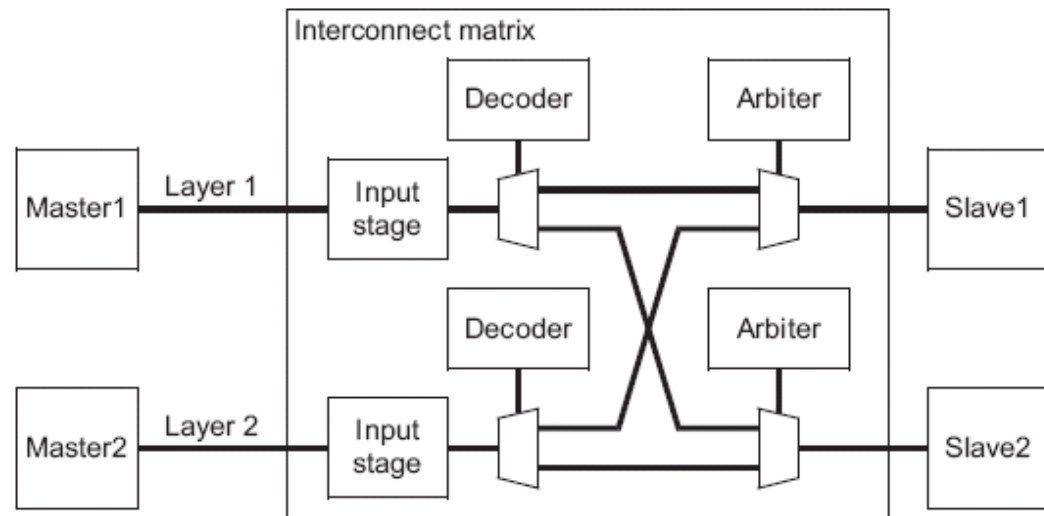


AHB or AHB-Lite

Multi-Layer Interconnect Scheme



- Interconnect Matrix Can Be Extended to Multiple Masters and Slaves
- Masters Can Be AHB or AHB-Lite



AMBA Bus Interface Signals

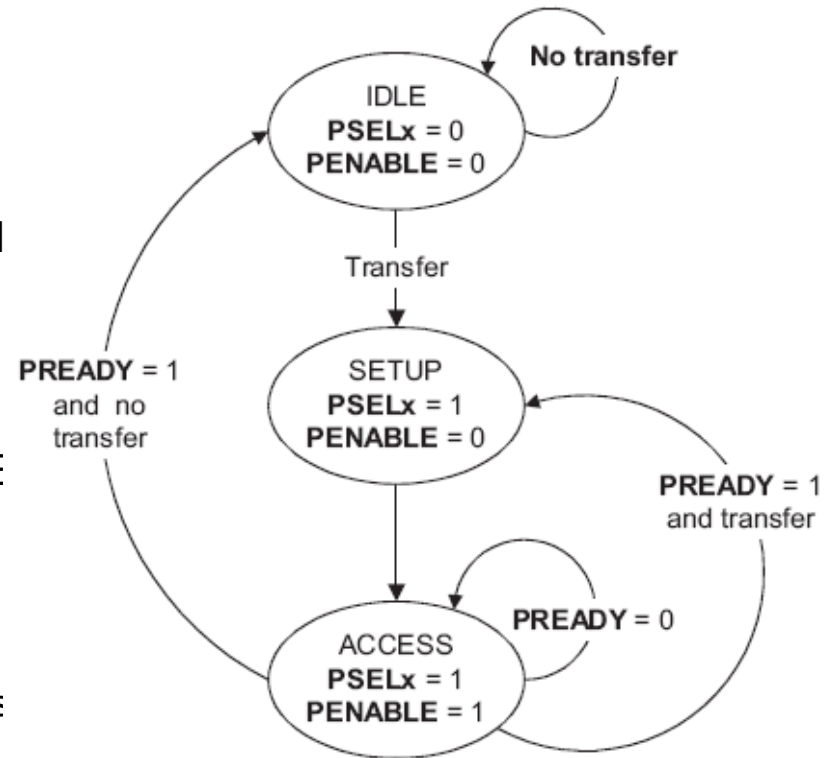
APB Bus

Signal	Function
PCLK	Bus Clock
PRESETn	ACTIVE-LOW Bus Reset
PADDR(31:0)	System Address Bus (up to 32 Bits Wide)
PSELx	Select Signal for APB Slave
PENABLE	Indicates Second and Subsequent APB Transfer Cycles
PWRITE	Transfer Direction (1=Write, 0=Read)
PWDATA	Write Data (up to 32 Bits from Master)
PRDATA	Read Data (up to 32 Bits from Slave)
PREADY	Transfer Complete (Used by Slave to Insert Wait States)
PSLVERR	Indicates APB Transfer Failure (OPTIONAL)

APB Bus

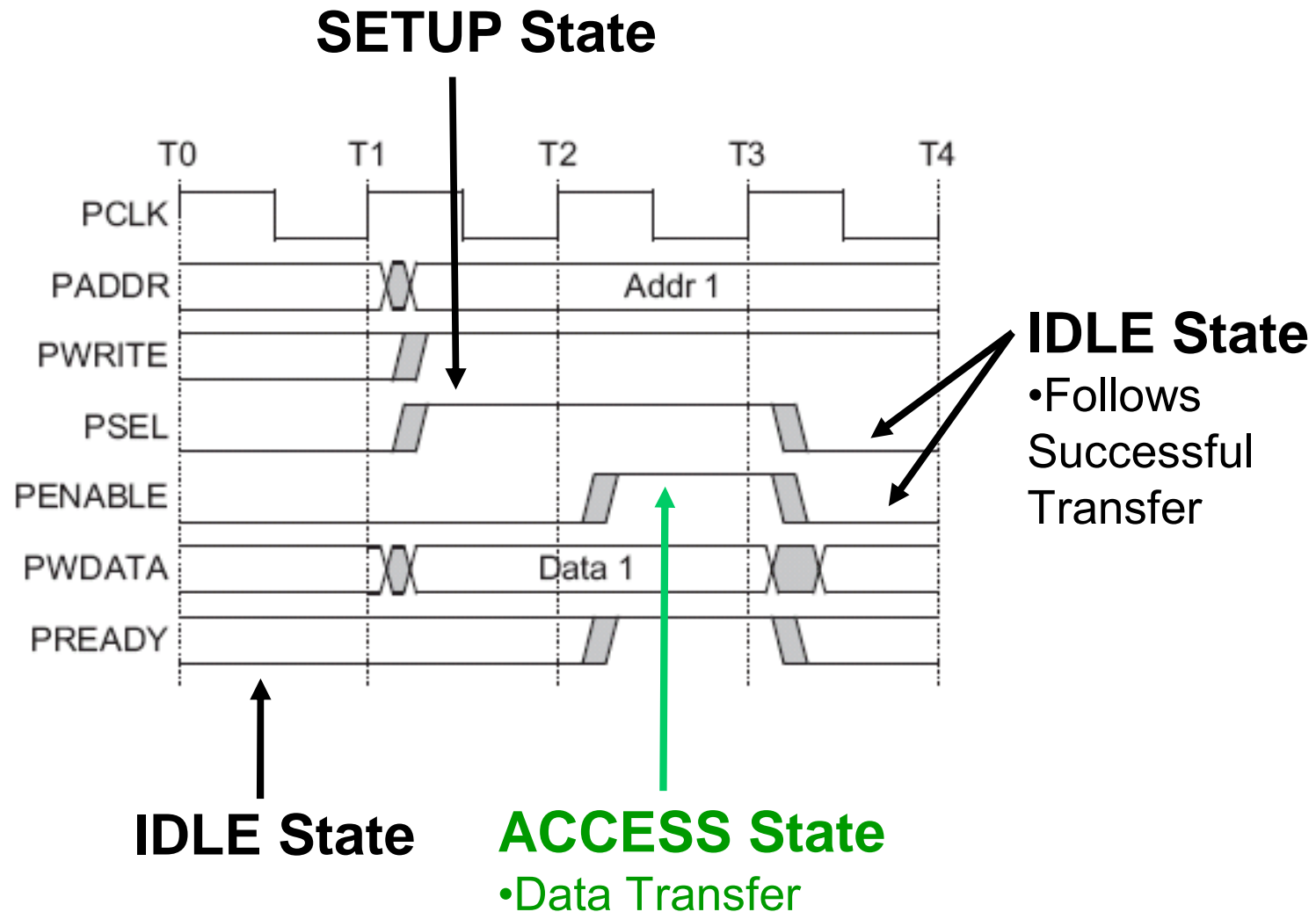
State Machine

- **IDLE**
 - Default APB State
- **SETUP**
 - Bus Moves Here when Transfer Is Required
 - Select PSELx for Targeted Slave Asserted
 - Bus Remains Here *Only One* Clock Cycl
 - Moves to Access State on Next Clock Rising Edge
- **ACCESS**
 - Enable PENABLE Asserted
 - PADDR, PWRITE, PSELx, and PWRITE Must Remain Stable during Transition from SETUP to ACCESS State
 - Exit from ACCESS State Controlled by PREADY from Slave
 - PREADY HIGH – Successful Transfer
 - PREADY LOW – State Machine Remains in ACCESS State



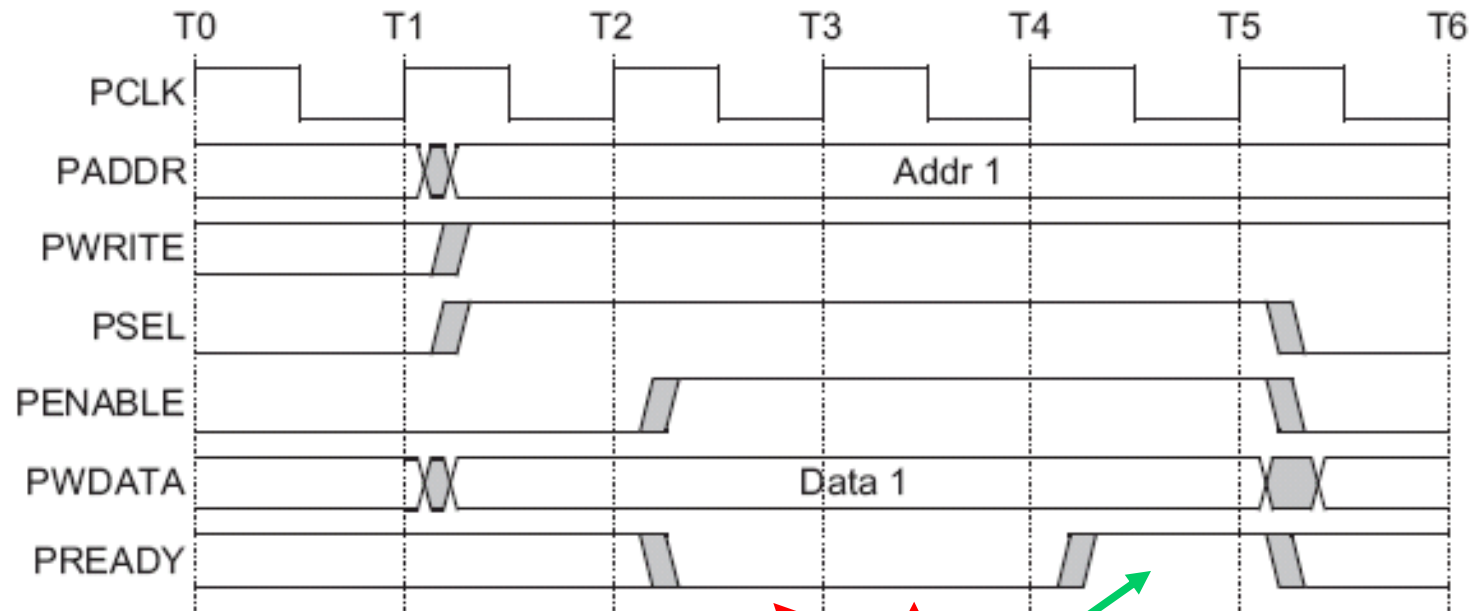
APB Write Transfer

No Wait States



APB Write Transfer

With Wait States

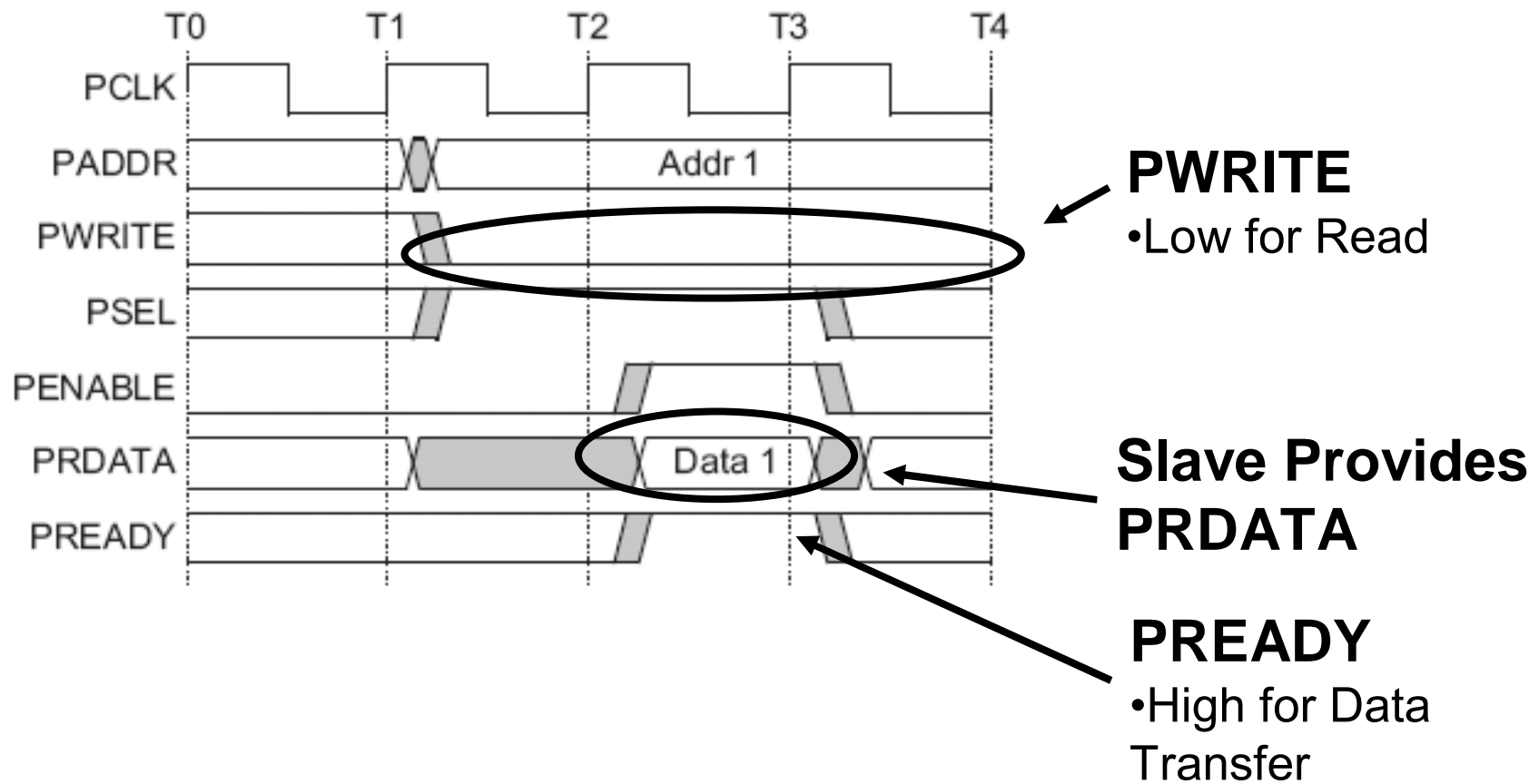


ACCESS States

- Remains Here until $PREADY = 1$

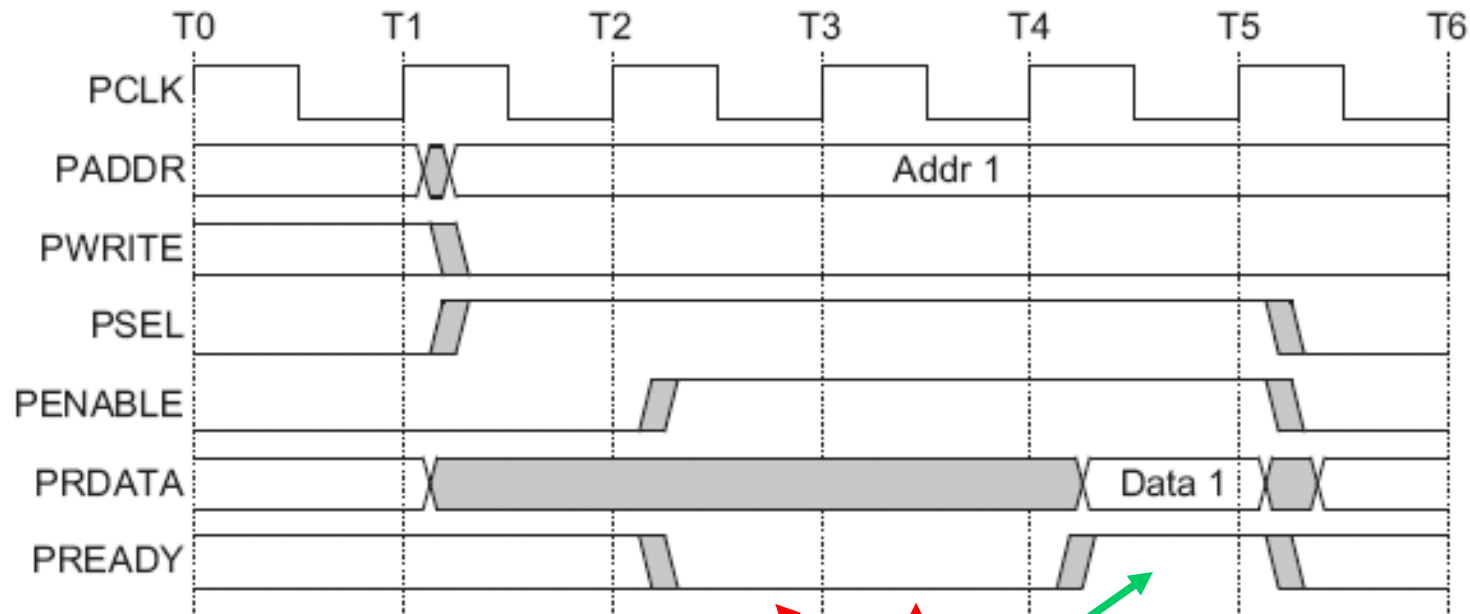
APB Read Transfer

No Wait States



APB Read Transfer

With Wait States

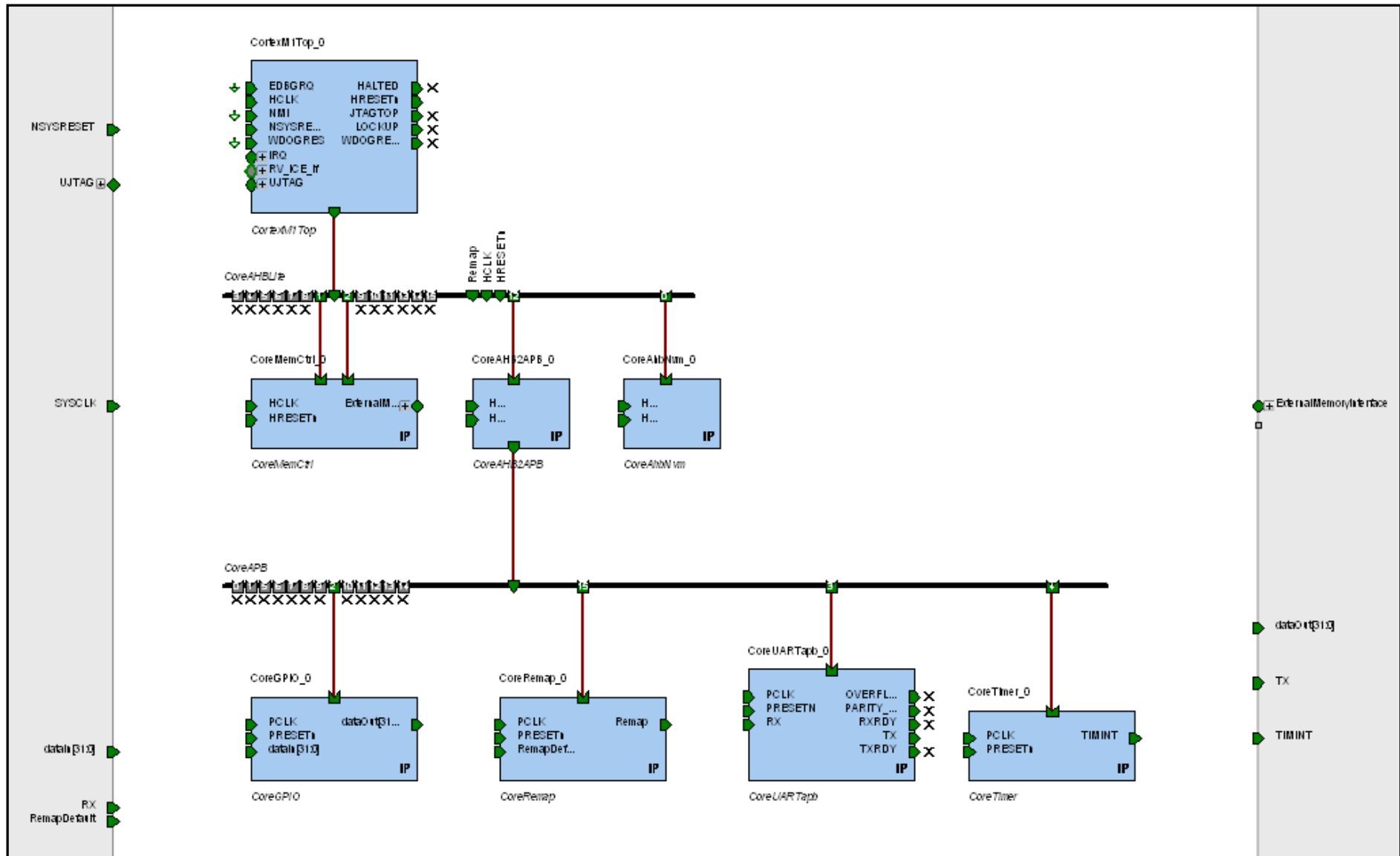


ACCESS States

- Remains Here until $PREADY = 1$

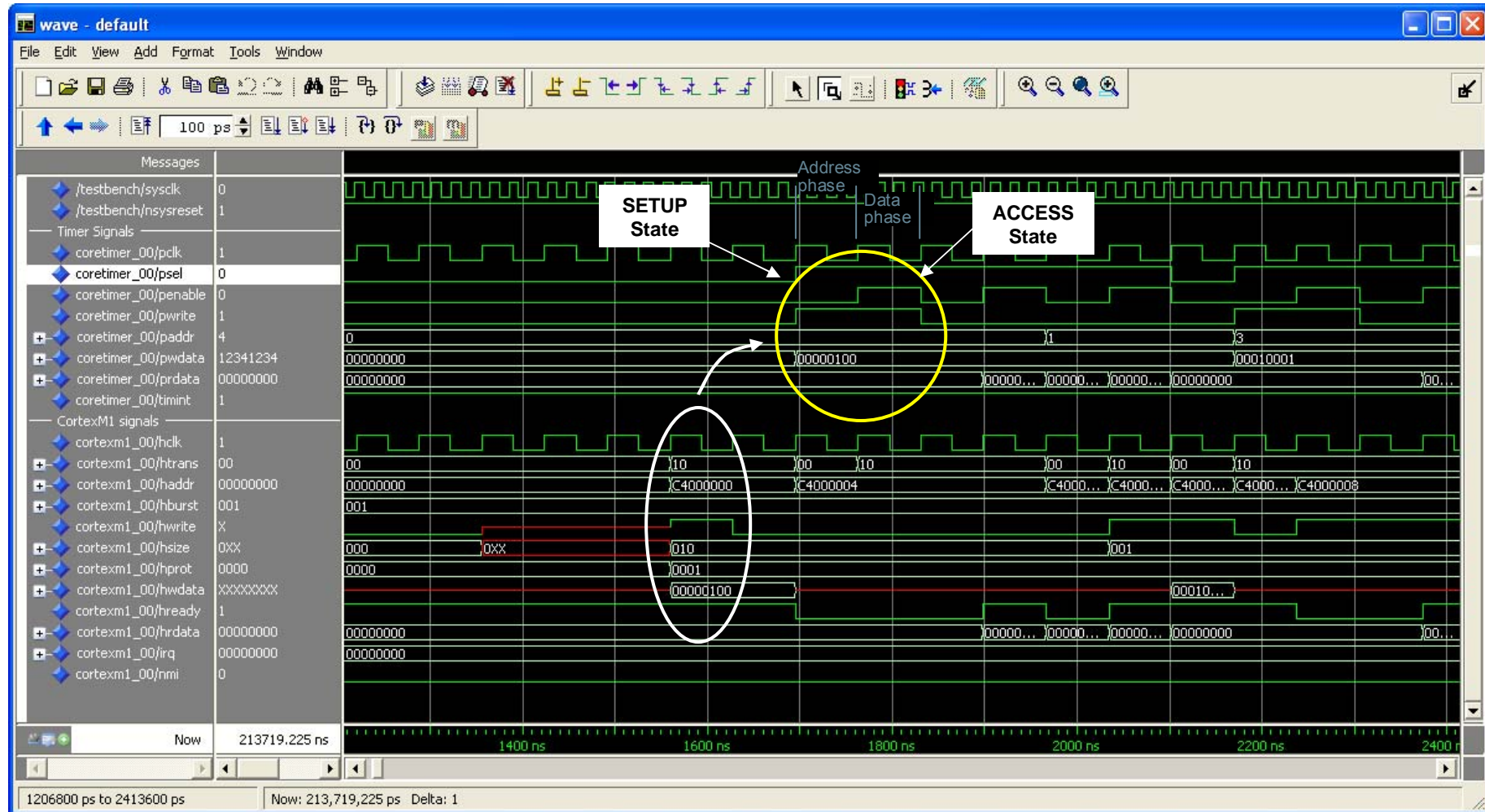
AHB to APB Transactions

Example



BFM Simulation

ModelSim Wave Window



APB write cycle (yellow) – write 0x0000100 to CoreTimer TimerLoad register

Note that paddr = 0x000 – CoreAHB2APB strips off the upper 8 address bits and drives the appropriate PSEL

AMBA Bus Architecture

Summary

- AHB is a High-performance Bus
 - Supports Bursting and Split Transfers

- APB is a Simple, Low-performance Bus



SmartDesign and IP Cores

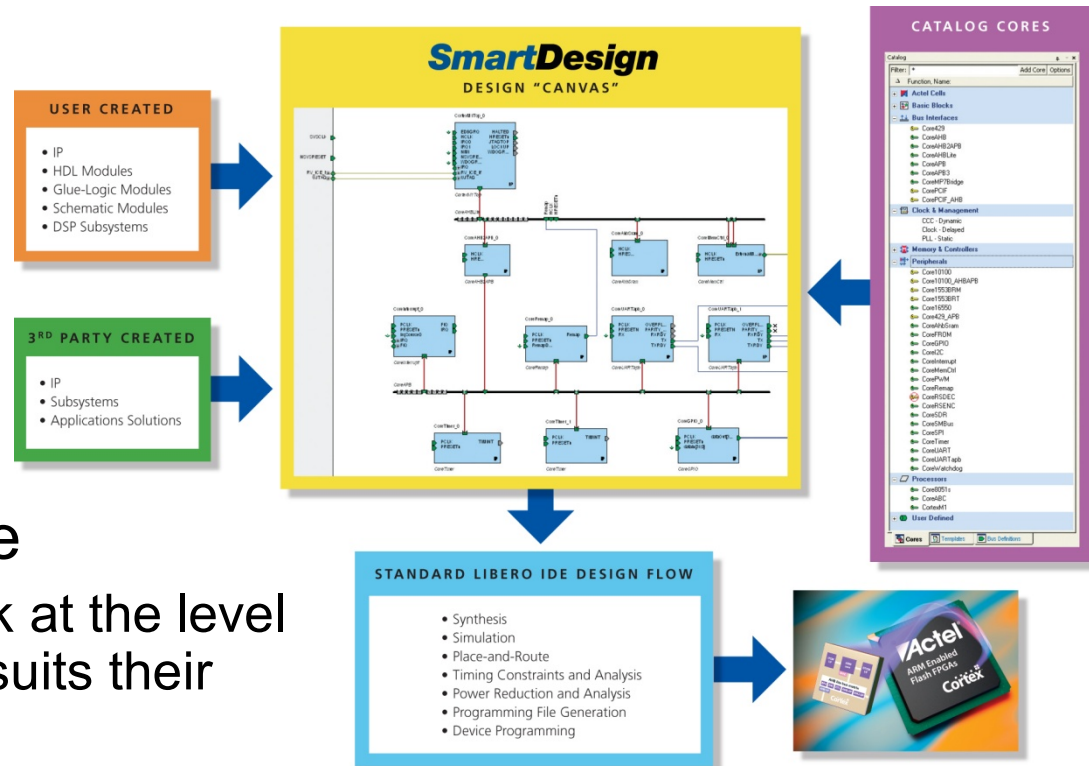


Using SmartDesign

SmartDesign Vision

■ Next Generation Design Entry Tool

- First tool in the industry that can be used for designing System on a Chip designs, custom FPGA designs or a mixture of both types in the same design.



■ Simple and Intuitive

- Designers can work at the level of abstraction that suits their needs

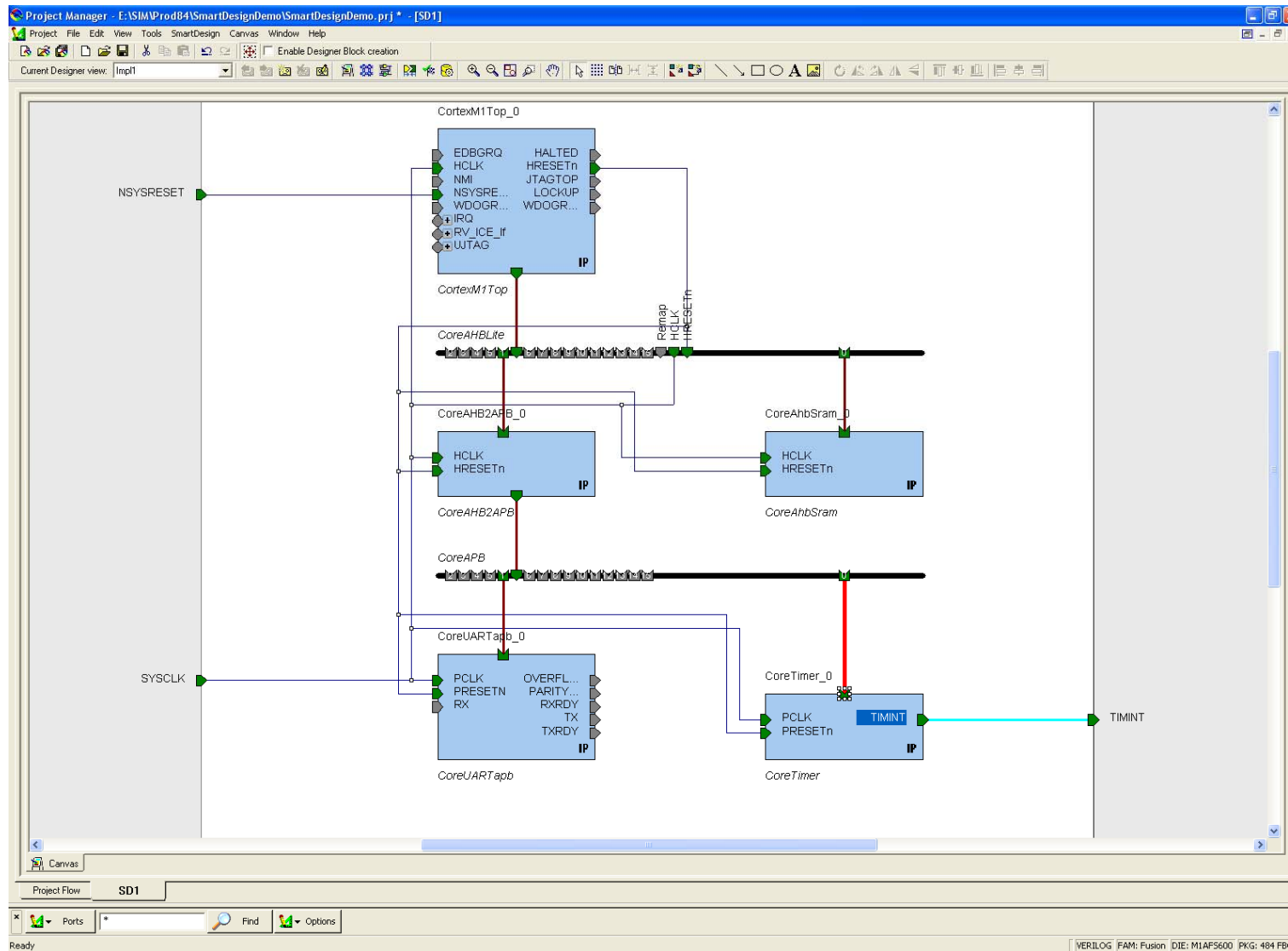
What is SmartDesign?

- Powerful Block-based Visual Design Creation Tool
 - Instantiate blocks from a variety of sources
 - DirectCore IP, SmartGen, User HDL, Companion Cores, Actel library cells, and the list goes on.
 - Supported for all platforms
- Simple and Intuitive Design Creation
 - Auto Connect
 - Fast manual connectivity between blocks
 - Hierarchical design support
- DRC
 - Checks rules to guarantee correct by construction design
 - Connectivity errors
 - Configuration errors
 - Special silicon rules
- SOC Features
 - Auto Connect
 - clocks and resets for processors and peripherals
 - Other known DirectCore connections
 - Memory Map Configuration Dialog
 - Testbench and Bus Functional Model (BFM) script generation

SmartDesign Canvas

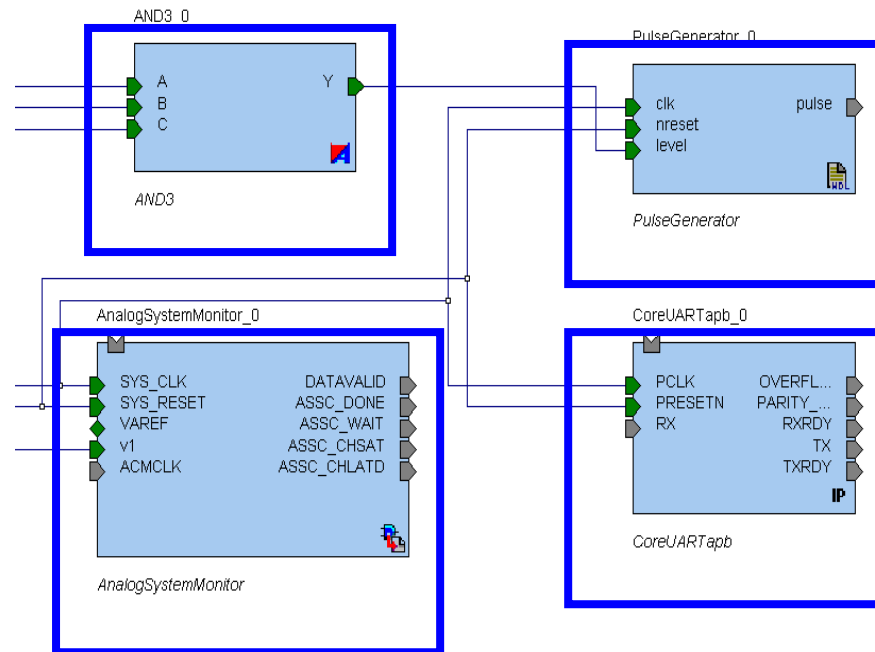
- SmartDesign Canvas
 - Instance pins are displayed on canvas
 - Connections are shown using nets
 - Displaying of Nets is optional
 - Selective enabling / disabling of showing nets
 - Drag and Drop directly from the Catalog into the Canvas
- All Design Operations Available in the Canvas
 - Connect / Disconnect
 - Promote To Top
 - Tie Low / Tie High / Tie Constant / Inversion
 - Float
 - Split (if bus)
 - Group

SmartDesign Canvas

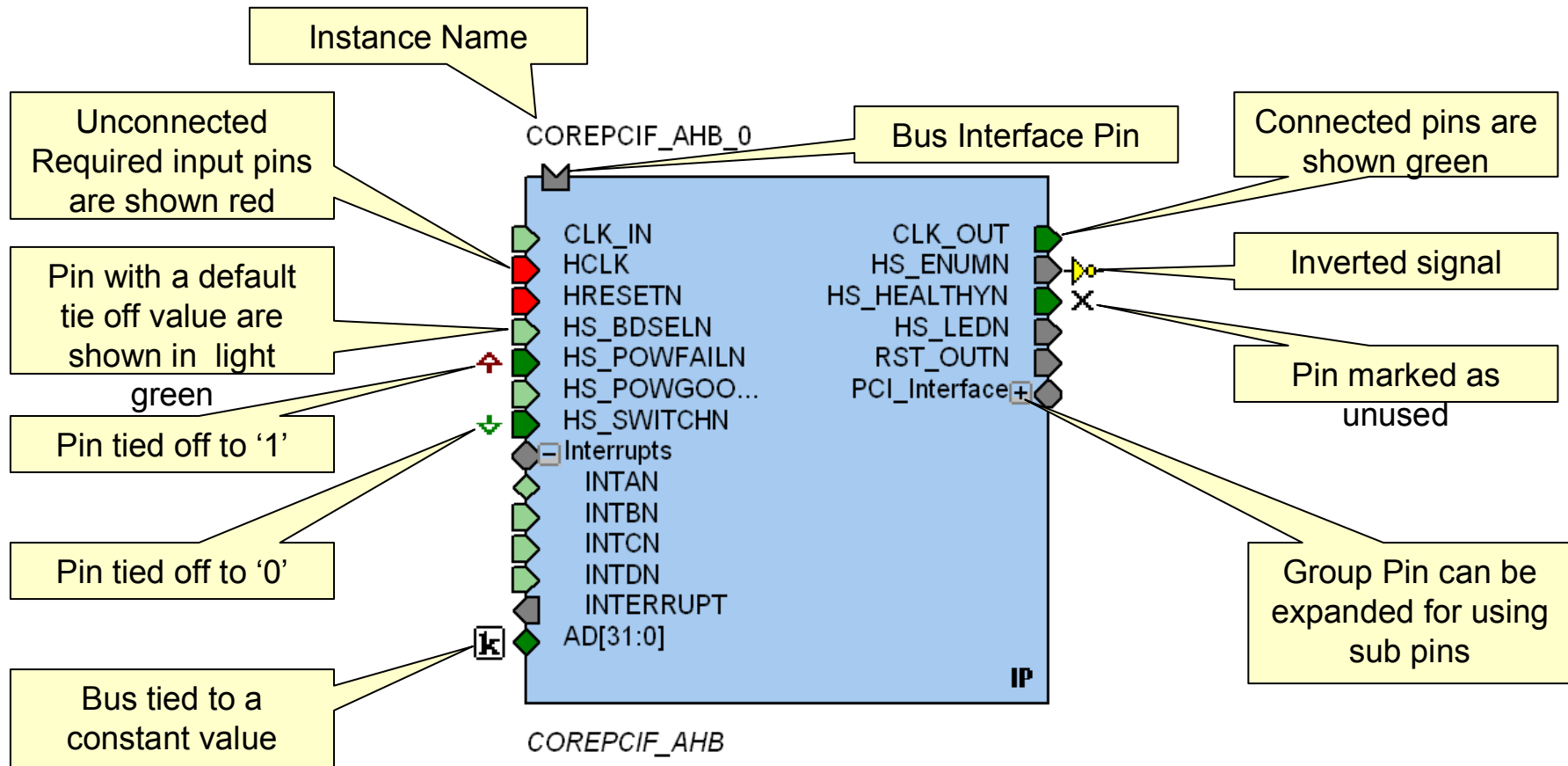


Design with any Block Types

- One Tool to Connect Your Design
 - DirectCore IP
 - SmartGen Cores
 - User HDL
 - Actel Macros (And, Or, I/Os, etc)

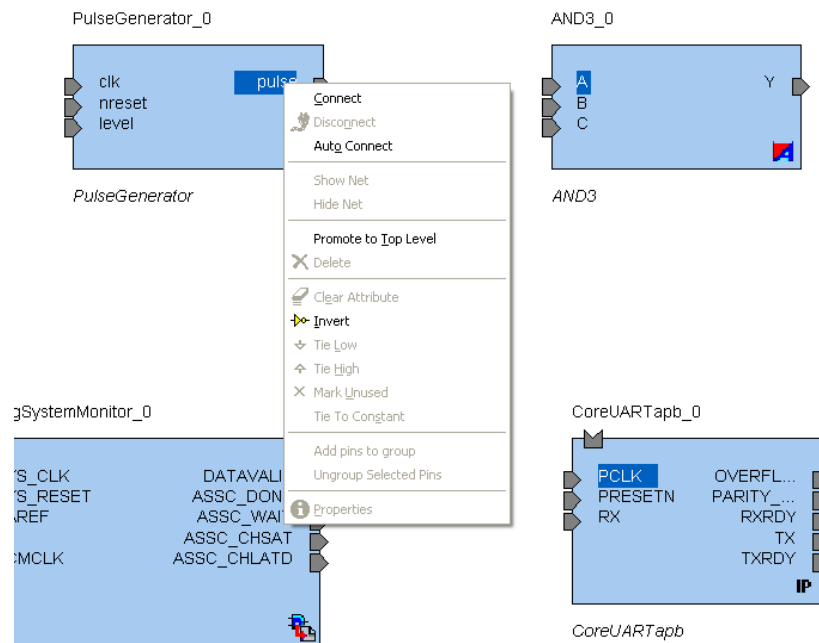


Canvas Instance



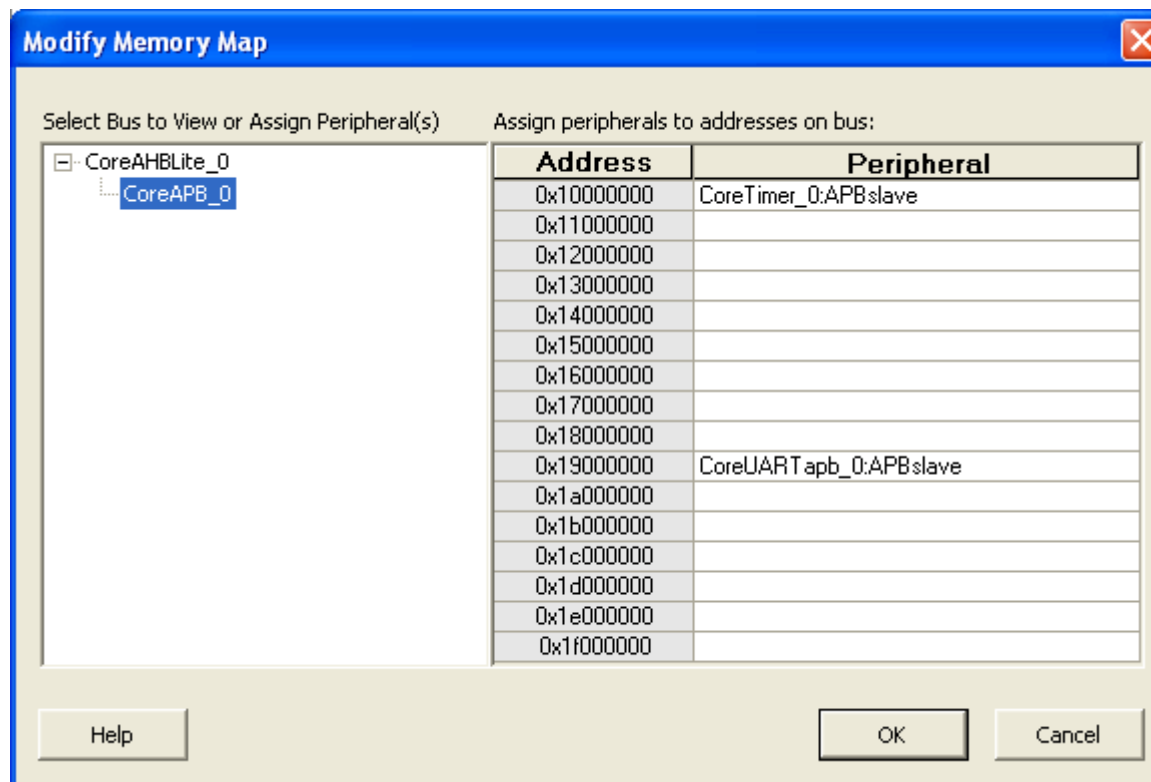
Making Connections

- Right-click on a pin for Available Operations
- Select 2 or More Pins With the CTRL key, Right-click and Connect



Modifying Memory Map

- Easy and Intuitive Method to Connect Peripherals at Particular Addresses on the Bus
- Interactive and Immediate Updating of Base Addresses



Testbench Generation

- Design Testbench
 - Generates a top level testbench
 - Clock and Reset drivers automatically generated and connected

- Bus Functional Model Script Generation
 - Generates BFM script file for processor based designs
 - Based on your peripheral connectivity in your designs
 - Look at Processor Core Handbooks (ex: CortexM1) for more details on BFMs and usage

Design Rule Checker

- Checks your Design for Errors

- Invoke with Checker icon (see picture)

- Connectivity

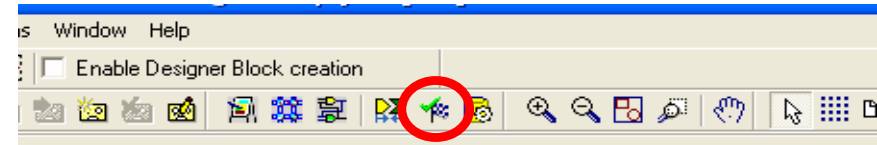
- Unconnected/Required input pins
- Floating output pins
- Silicon required connection

- Example: RTC must be driven by Crystal Oscillator

- Configuration

- Check consistency between configurations

- Example: CoreMP7 and CoreMP7Bridge debug configuration



- Errors Reported in a Connectivity Grid

- Directly fix your connectivity mistakes

- Enables fast sorting / filtering

A screenshot of a Connectivity Grid window. The window displays a table with columns for Message, Instance, Port Name, Slice, Attribute, and SD1. The table contains several rows of error messages, including 'Floating Driver', 'Unconnected Bus Interface', and 'Undriven Pin'. A context menu is open over the 'Undriven Pin' row, showing options like 'Invert', 'Tie Low', and 'Tie High'.

Message	Instance	Port Name	Slice	Attribute	SD1
Floating Driver					0
Unconnected Bus Interface					0
Undriven Pin	CoreAHLite_0	Remap			
	CoreUARTapb	BX			
	CortexM11op_0				0

Datasheet Generation

- Design Datasheet
 - Pin outs of the design
 - Cores used and their description
 - Memory Map

Project Settings

FAM: Fusion
Die: M1AF
Package: 484 F
HDL: Verilo
Location: E:/SIM
State: SAVE

[IO's](#)
[Cores](#)
[Memory Map](#)

[CoreTimer_0](#)
[CoreUARTapb_0](#)
[CortexM1Top_0](#)

[top of page](#)

Instance Name:
Type:
Vendor:
Library:
Core Name:
Version:
Description:

[instance list, top of page](#)

Instance Name:
Type:
Vendor:
Library:
Core Name:
Version:
Description:

[instance list, top of page](#)

CortexM1Top_0 Sub

Master(s) on this bus:

- CortexM1Top_0

[CoreAh](#)

[CoreMe](#)

[CoreMe](#)

[CoreGPI](#)

[Core](#)

[CoreTime](#)

[CoreRema](#)

CoreTimer_0 : RegisterMap Memory Map

range: 0x01000000

Address	Type	Width	Reset Value	Name	Description
base address + 0x00	read-write	32	0x00000000	TimerLoad	
base address + 0x04	read-only	32	0xFFFFFFFF	TimerValue	
base address + 0x08	read-write	32	0x00	TimerControl	
base address + 0x0C	write-only	32	0x0	TimerPrescale	
base address + 0x10	write-only	32	0x0	TimerIntClr	
base address + 0x14	read-only	32	0x0	TimerRIS	
base address + 0x18	read-only	32	0x0	TimerMIS	

[back to CortexM1Top_0 Memory Map](#)

TimerLoad register details:

Bit Offset	Type	Bit Width	Name	Description
0	read-write	32	LoadValue	Load value for counter

[back to CoreTimer_0 Registers](#)

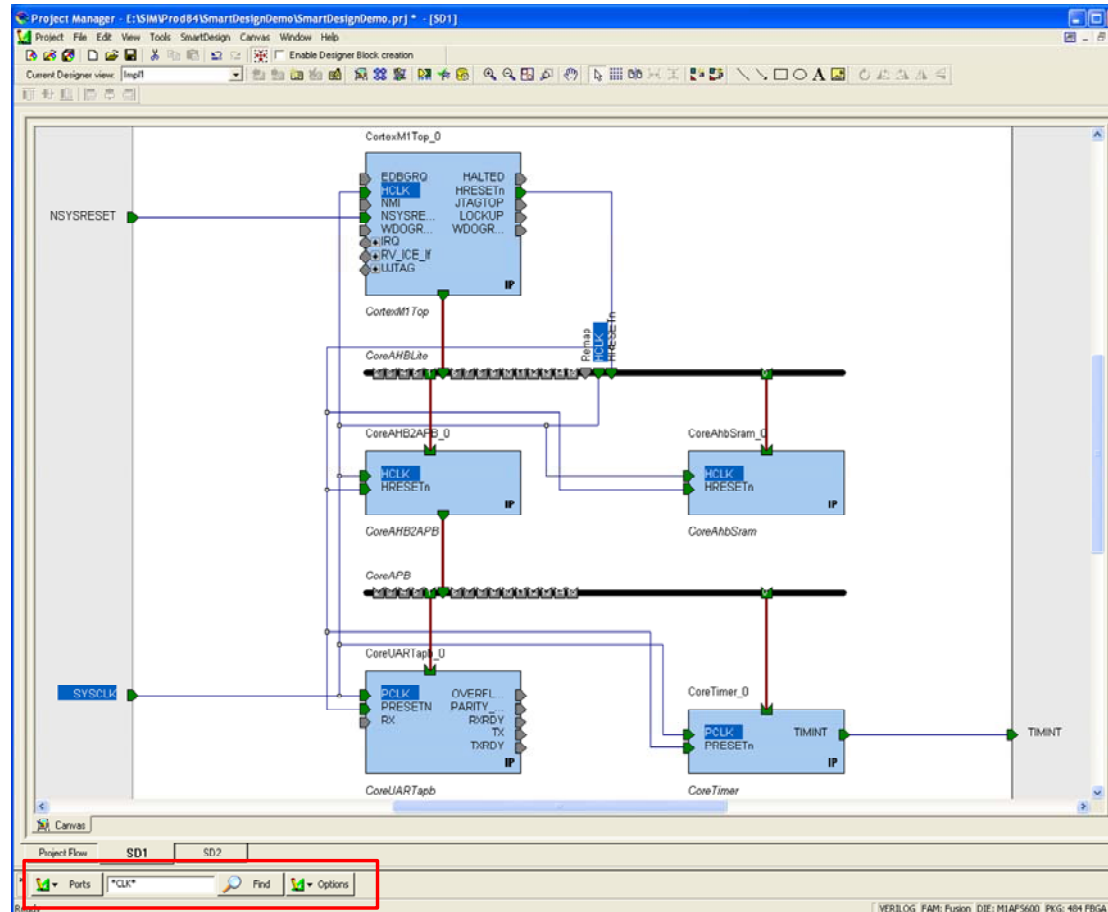
TimerValue register details:

Bit Offset	Type	Bit Width	Name	Description
0	read-only	32	CurrentValue	Current value of counter

[back to CoreTimer_0 Registers](#)

Fast Design Search

- Design Level Search
 - Find Pins, Instances, and Nets quickly and easily
 - Wild card query
 - Matching objects will be highlighted in the Canvas



Multiple Design Representations

- Connectivity Grid
 - Spreadsheet like view of your design
 - Enables quick filtering / sorting

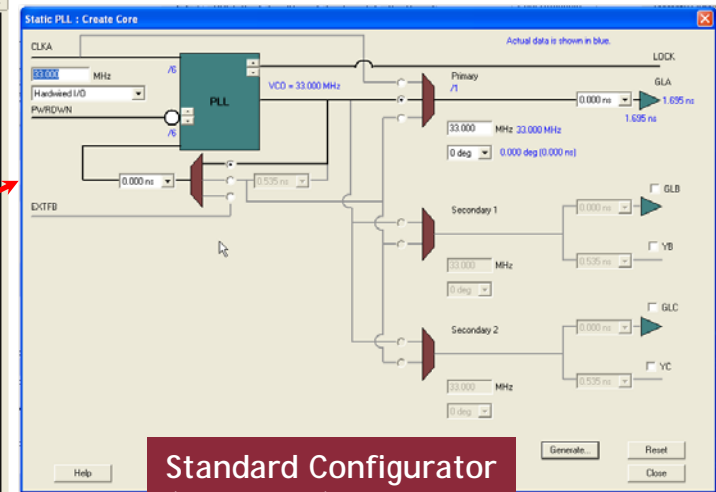
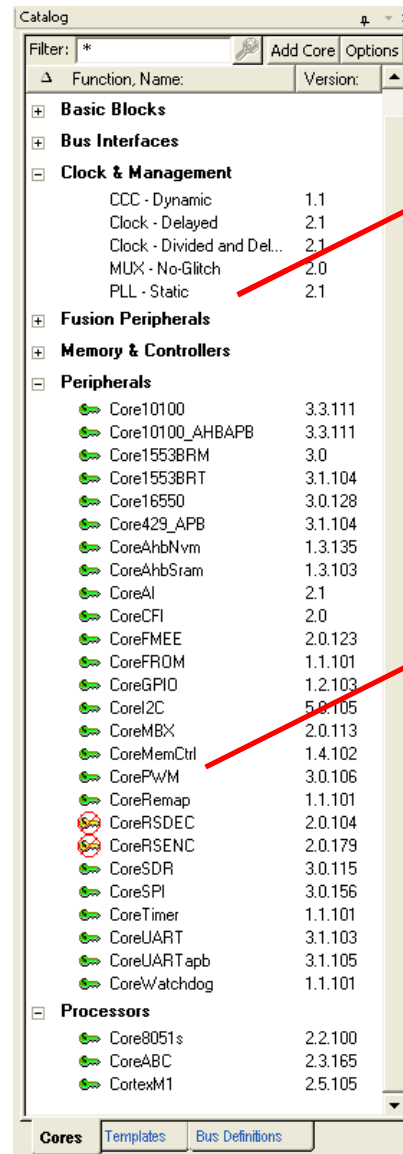
- Schematic
 - Shows all pins and nets
 - Traditional schematic view

The screenshot displays a schematic view of a Cortex-M1 core (CORE8051S_0) with various pins and nets connected. Overlaid on the schematic is the 'Grid' window, which provides a spreadsheet-like view of the design's connectivity. The grid is organized into columns for different components and their attributes.

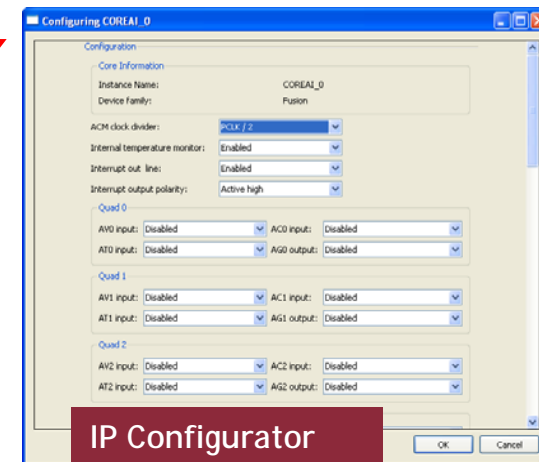
Instance-Instance View		Instances		
Net-Instance View		Attribute	SD_8051	CORE8051S_0
Instance	Port Name			
SD_8051	APB3master			APB3master
	AUXOUT			AUXOUT
	BREAKIN			BREAKIN
	BREAKOUT			BREAKOUT
	CLK			CLK
	DATAVALID		DATAVA...	
	DBGMEMPSWR			DBGMEMPSWR
	DebugIf			
	InitCigAnalog_bif		InitCigAn...	
	MEMBANK [3:0]			MEMBANK(3:0)
	PRESETN			PRESETN
	PSEL			
	RTCMATCH		RTCMAT...	
	TCK			TCK
	TDI			TDI
	TDO			TDO
	TMS			TMS
	TRIGOUT			TRIGOUT
	TRSTN			TRSTN
	VAREF	PAD	VAREF	VAREF
	WDOGRES			WDOGRES
AB1_0	ACMCLK			
	ASSC_CHLATD			
	ASSC_CHSAT			
	ASSC_DONE			
	ASSC_WAIT			
	DATAVALID		DATAVALID	
	InitCigAnalog_bif		InitCigAnalog...	
	RTCMATCH		RTCMATCH	
	RTCTL_bif	BIF		
	SYS_CLK			
	SYS_RESET			
	VAREF	PAD	VAREF	VAREF
CORE8051S_0	APB3master			APB3master
	AUXOUT			AUXOUT
	BREAKIN			BREAKIN
	BREAKOUT			BREAKOUT
	CLK			CLK
	DBGMEMPSWR			DBGMEMPS...
	DebugIf			

Cores Catalog

- Library of Proven Configurable Core Functions
- Actel Macros
- Quick-find
- Intuitive Configuration
- Drag and drop to Canvas

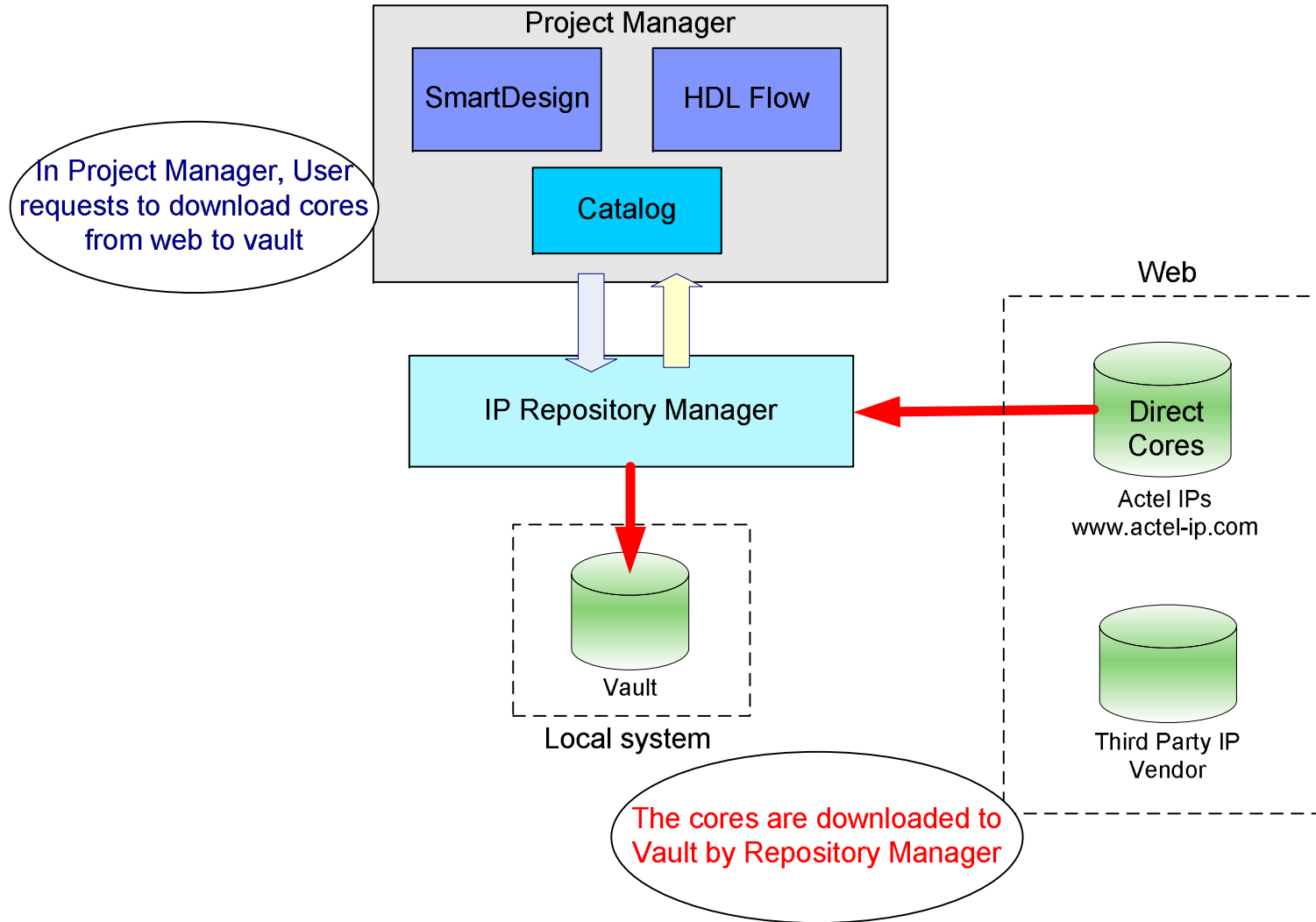


Standard Configurator (SmartGen)



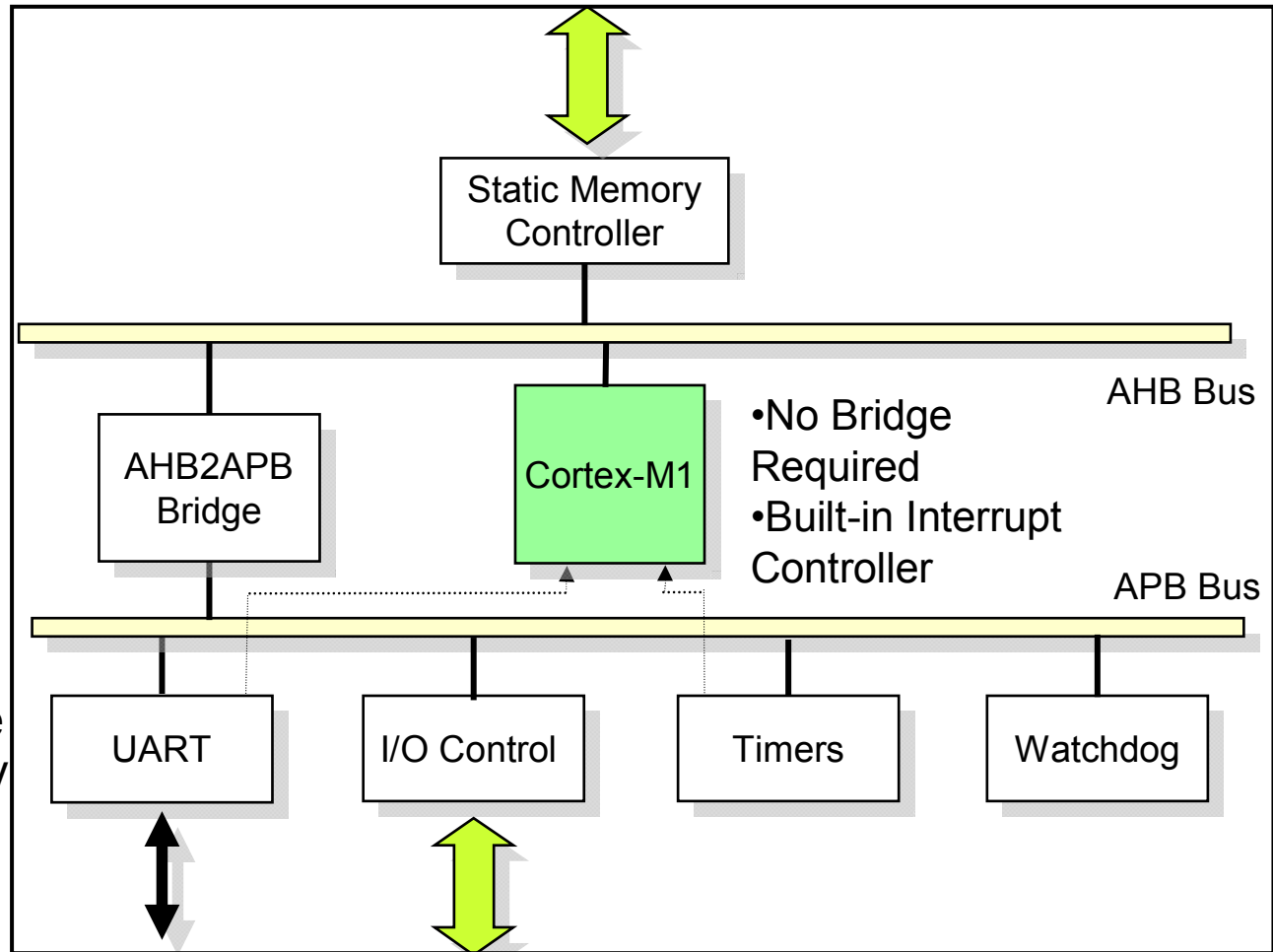
IP Configurator (DirectCore)

SmartDesign IP Delivery



Cortex-M1 System-on-Chip

- Processor System
 - Processor
 - Bus Fabric
 - Components
- Components
 - Cortex-M1
 - AMBA
 - IP Cores
- SmartDesign Automatically Creates Basic System
 - ... OR ...
 - User Can Create System Manually



SmartDesign

Building an SoC

- Decide on Components Needed to Meet System Requirements
- Add Cortex-M1, Busses, and Components
- Autostitch
- Configure Components where Necessary
- Add Any Other Required Connections
- Check Memory Map
- Generate System

Libero IP Catalog

The screenshot displays the Actel Libero Project Manager interface for a project named 'CortexM1_lab\Solutions\WHDL\W1_SOC\W1_SOC.prj'. The main workspace shows the 'Design Entry Tools' flowchart, which includes HDL Editor, SmartDesign, and ViewDraw leading to a Synthesis step (Synplify) for 'SOC_TOP.ex'. Below this, it shows 'Post-Synthesis Files'. The 'Design Explorer' on the left shows a project structure with components like 'BLACKBOX_PACKAGE', 'SOC_TOP', and 'COREUARTAPB_LIB'. The 'IP Catalog' on the right is filtered by 'Function, Name' and 'Version', showing categories like 'Memory & Controllers', 'Peripherals', and 'Processors'. A warning message at the bottom states: 'Warning: The project was not us... The synthesis profile... The M1_SOC project was opened.' The status bar at the bottom indicates 'VHDL FAM: ProASIC3 DIE: M1A3P1000 PKG: 484 FBGA'.

IP Catalog

IP Cores in Libero Catalog

■ Processors

- Cortex-M1, CoreMP7
- Core8051s, CoreABC

■ AMBA Interfaces

- CoreAHB, CoreAHBLite
- CoreAPB, CoreAPB3
- CoreAHB2APB

■ Other Interfaces

- Core10/100, Core429, CorePCIF
- Core1553BRT, Core1553BRM

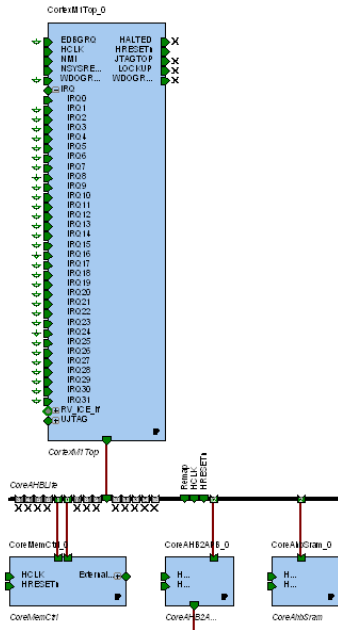
■ Subsystem Cores

- CoreAHBNvm
- CoreAHBSram
- CoreAI
- CoreCFI
- CoreDDR
- CoreFMEE
- CoreFROM
- CoreGPIO
- CoreI2C
- CoreInterrupt
- CoreMemCtrl
- CorePWM
- CoreRemap
- CoreSDR
- CoreSMBus
- CoreTimer
- CoreUART, CoreUARTapb
- CoreWatchdog

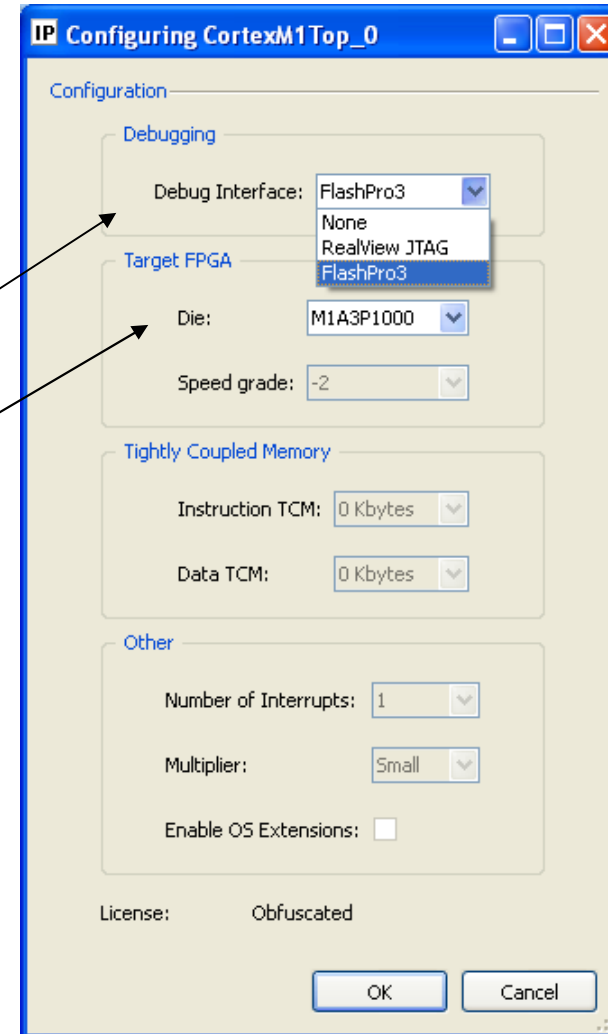


Cortex-M1

SmartDesign Configuration



- Select Debug Interface
 - None (Default)
 - RealView JTAG
 - FlashPro3
- Select Die
 - Die already Selected in Libero
 - M1AFS600, M1A3P1000, etc.
- Other Options Inactive



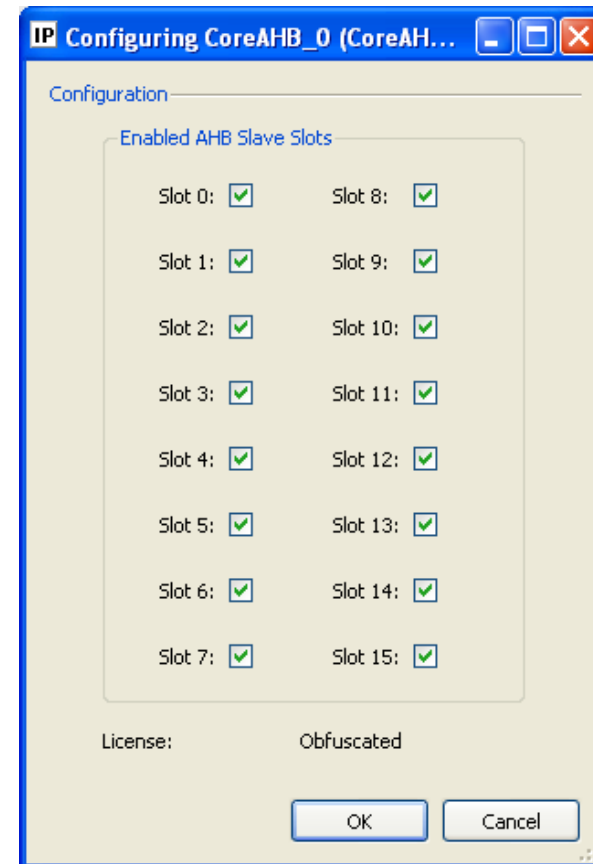
CoreAHBLite and CoreAHB

SmartDesign Configuration

■ CoreAHBLite



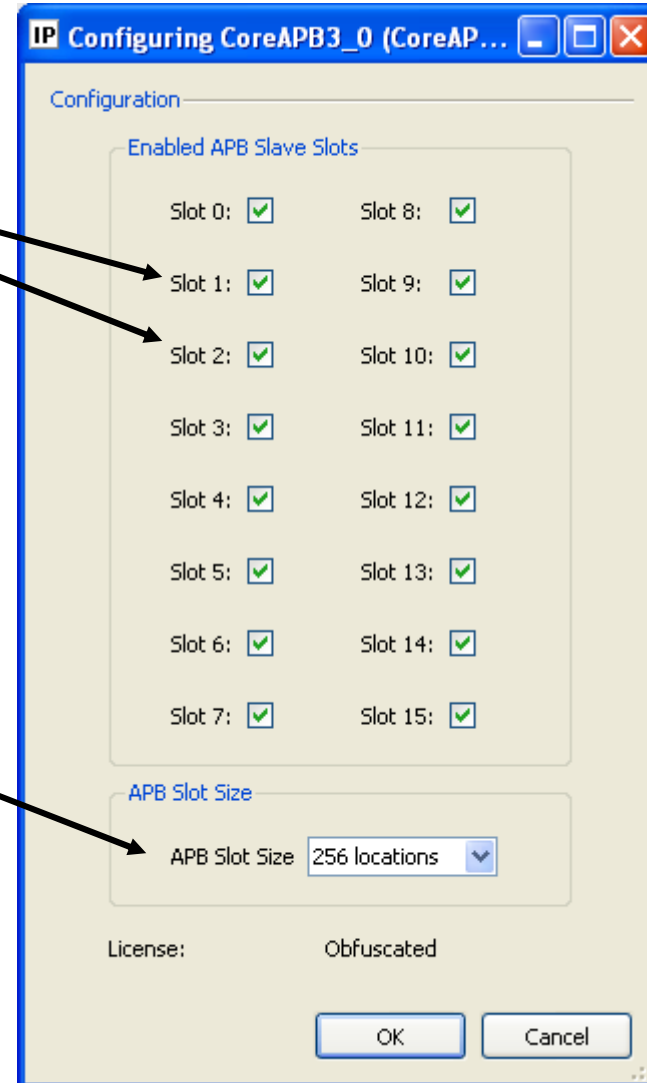
■ CoreAHB



- Identical Configuration
 - Enable/Disable Slots

CoreAPB3 Configuration Options

- Enable or Disable Each of 16 APB Slots

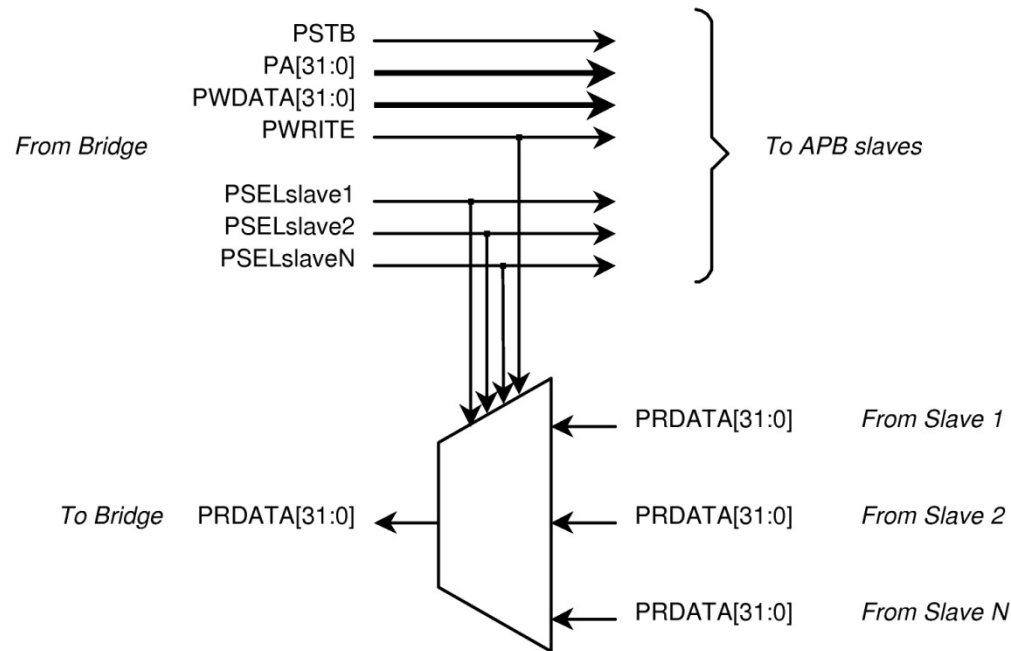


- APB Slot Size
 - Default Is 256 Locations per Slot

CoreAPB

SmartDesign Configuration

- Implements APB in Single Component
- Uses Selects from CoreAHB2APB
- Tile Count: 125



Other SmartDesign Cores

- UART
- Static Memory Controllers
 - Optimized for Internal and External Memory
- Timers
- General-Purpose I/O Controller (Small-footprint GPIO)
- Watchdog Controller
- CoreAI for Fusion
- PCI Controllers
 - Target, Master, and Target+Master
- Ethernet
- 1553
- All Actel IP Cores Are Now Distributed through SmartDesign

CoreUARTapb

- APB Slave
- Extension of CoreUART
 - Wrapped with APB Interface
 - Configuration Settings Exposed in Control Registers
 - Not Hardwired as in CoreUART
 - ReceiveFull and TxRdy Exposed
 - Can Be Used as Interrupt Sources
- Tile Count: 300

CoreMemCtl

- AHB Slave
- Optimized for Actel Development Kit
- Memory Interfaces
 - *External Asynchronous/Synchronous SRAM*
 - *External Asynchronous Flash*
- Two AHB Ports
 - Flash RAM (Typically Slot 0)
 - SRAM (Typically Slot 1)

- Tile Count: 100

CoreMemCtrl

Configuration

- SRAM Mode
 - Synchronous or Asynchronous
- Synchronous SRAM Mode
 - Pipeline or Flow-through
- Flash Data Bus Width
 - 32 or 16 bits
- Flash Read Wait States
 - 0, 1, 2, or 3
- Flash Write Wait States
 - 1, 2, or 3
- SRAM Read Wait States
 - 0, 1, 2, or 3
- SRAM Write Wait States
 - 1, 2, or 3
- Shared Flash/SRAM Read/Write Enables
 - Select 'Yes' or 'No'

IP Configuring CoreMemCtrl_0 (CoreMemCtrl - 1....)

Configuration

SRAM mode:
 Synchronous Asynchronous

Synchronous SRAM mode:
 Flow-through Pipeline

Flash data bus width:
 16 bit 32 bit

Number of wait states for Flash read:
 0 1 2 3

Number of wait states for Flash write:
 1 2 3

Number of wait states for SRAM read:
 0 1 2 3

Number of wait states for SRAM write:
 1 2 3

Read and write enables shared for Flash and SRAM:
 No Yes

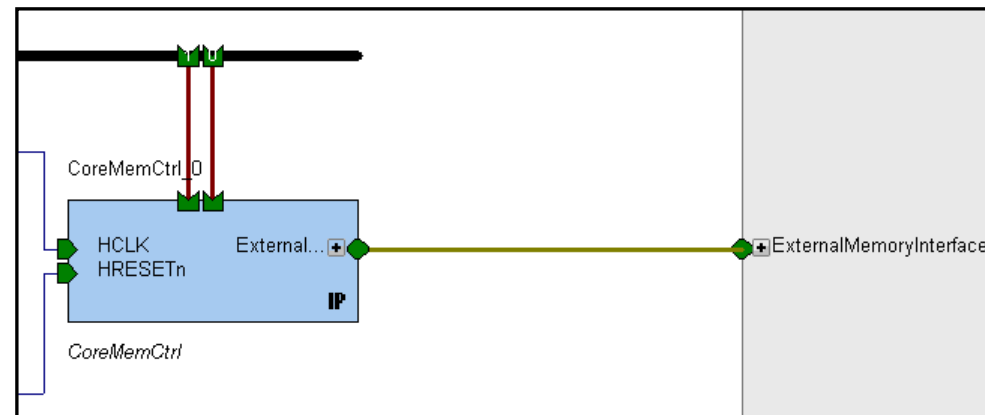
License: Obfuscated

OK Cancel

CoreMemCtl

External Memory Interfaces

- External Memory Interface of CoreMemCtl Should Be Routed to Subsystem Top Level
 - Generic Interface Accommodates a Variety of Flash and SRAM Configurations



- Memory Devices Typically Have Several Inputs Fixed at Static Levels
 - These Should Be Handled in the Top-level Description for FPGA (above Subsystem Top Level)

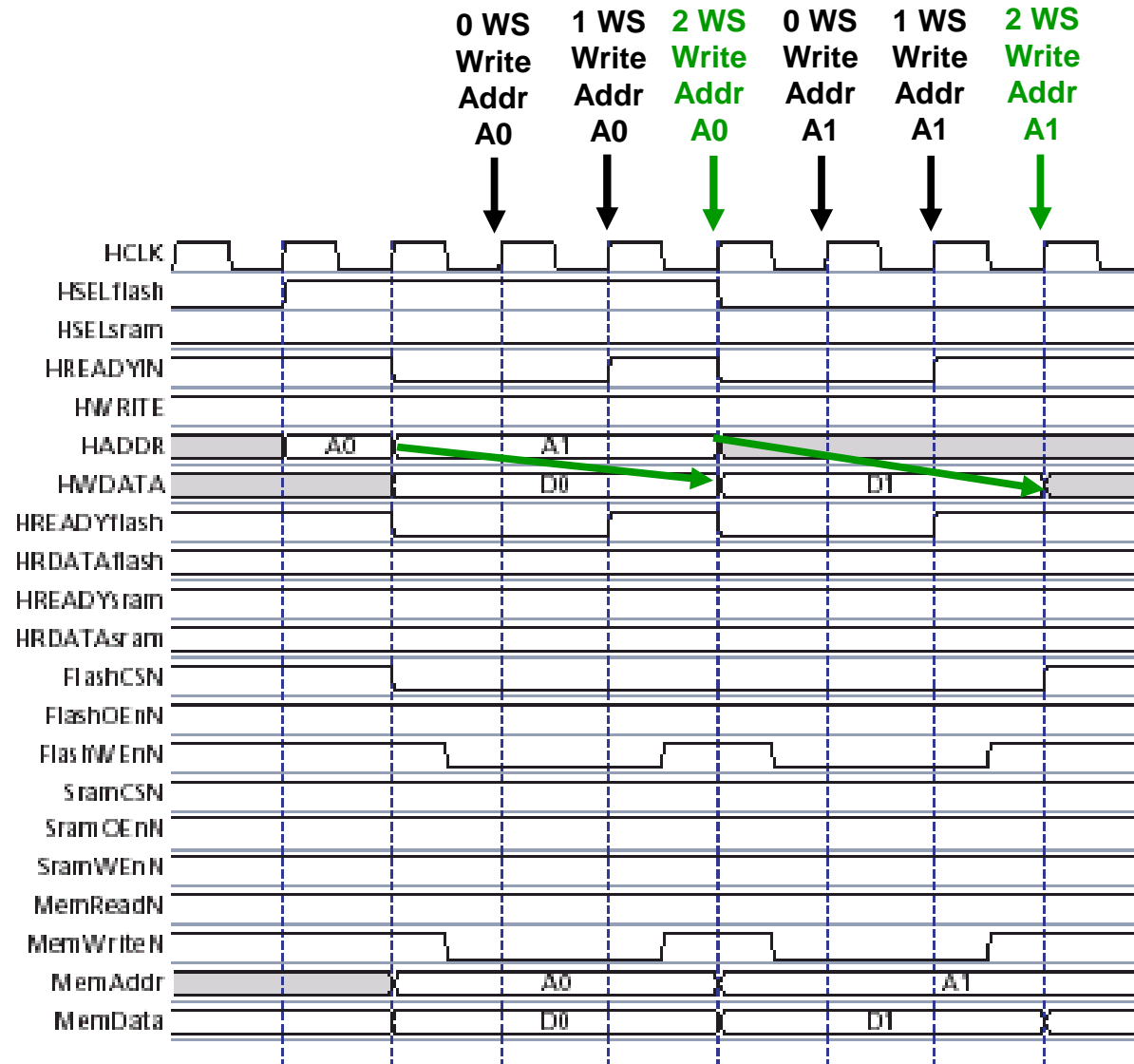
CoreMemCtl

Interfacing

- Flash
 - Read and Write Wait States Can be Inserted
- Asynchronous SRAMs
 - Read and Write Wait States Can be Inserted
- Synchronous SRAMs
 - CoreMemCtl Designed to Connect to either Pipelined or Flow-through Devices
 - Pipeline Mode – Output Data Assumed Valid just after the SRAM Clock Edge that Follows the Edge on which the Read Address Is Clocked into the SRAM
 - Flow-Through Mode, Additional Register Is Implemented FPGA Logic to Accomplish the Same Thing

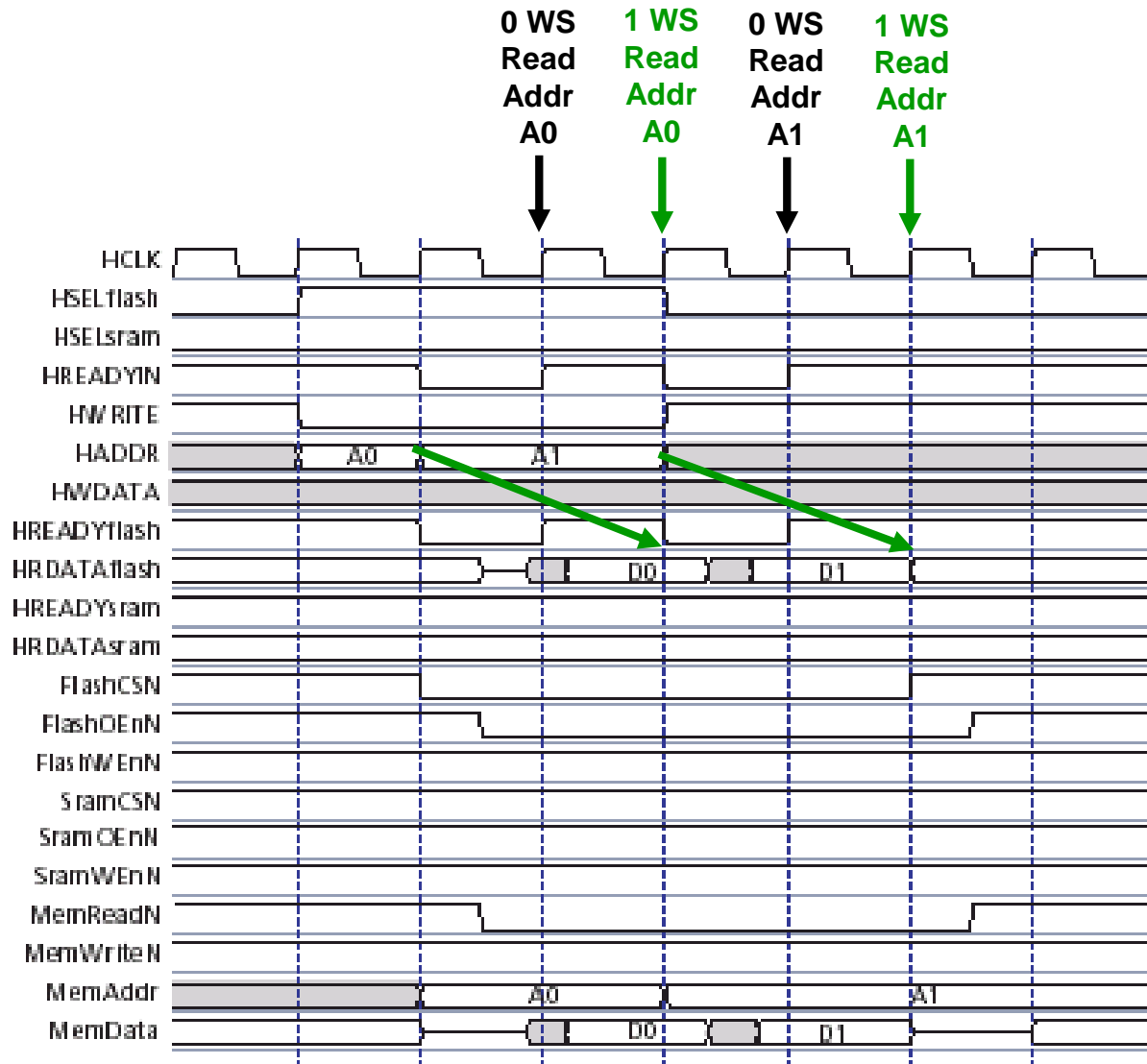
Flash Interfacing

Write Wait States – 2-Wait-State Example



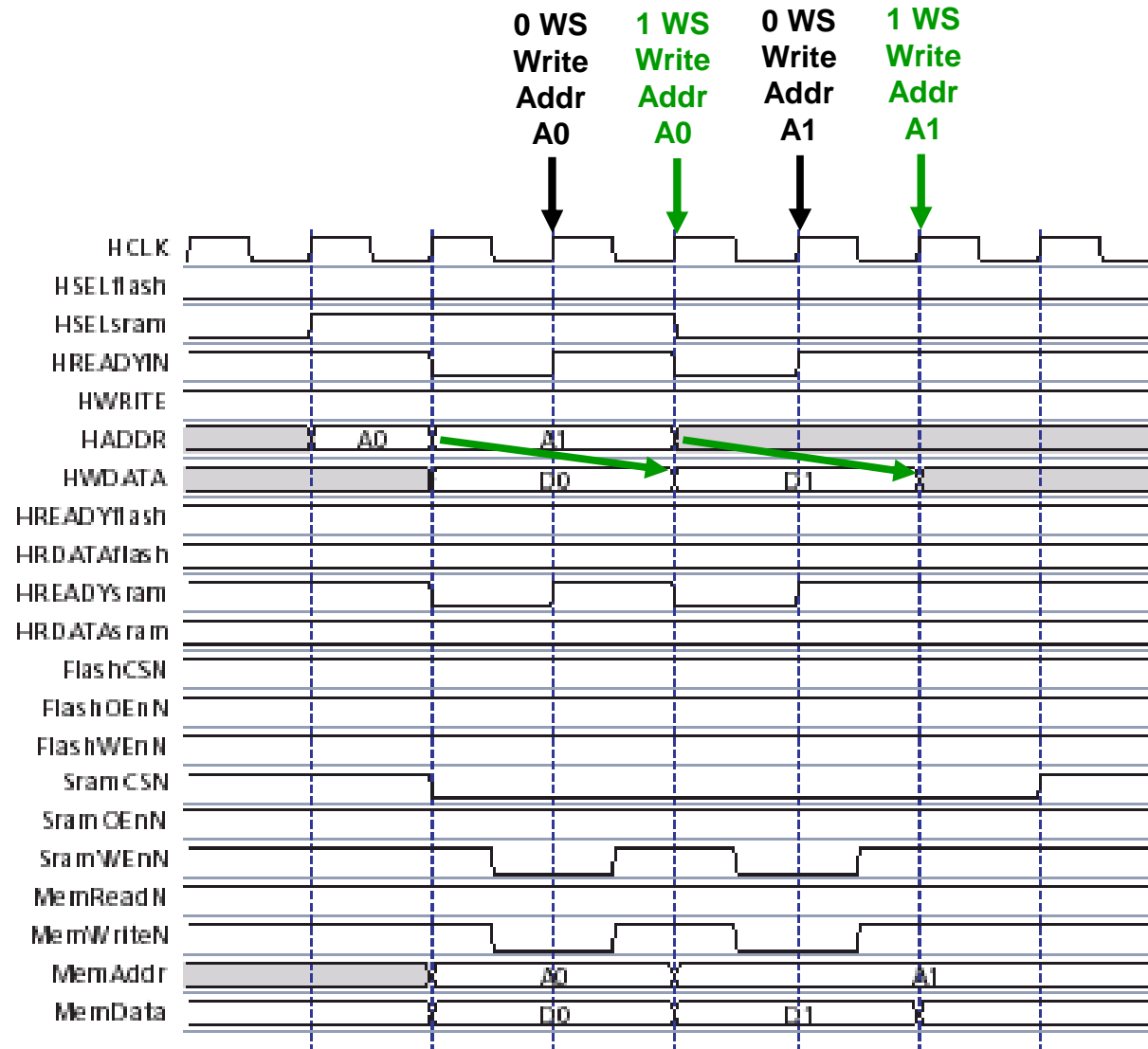
Flash Interfacing

Read Wait States – 1-Wait-State Example



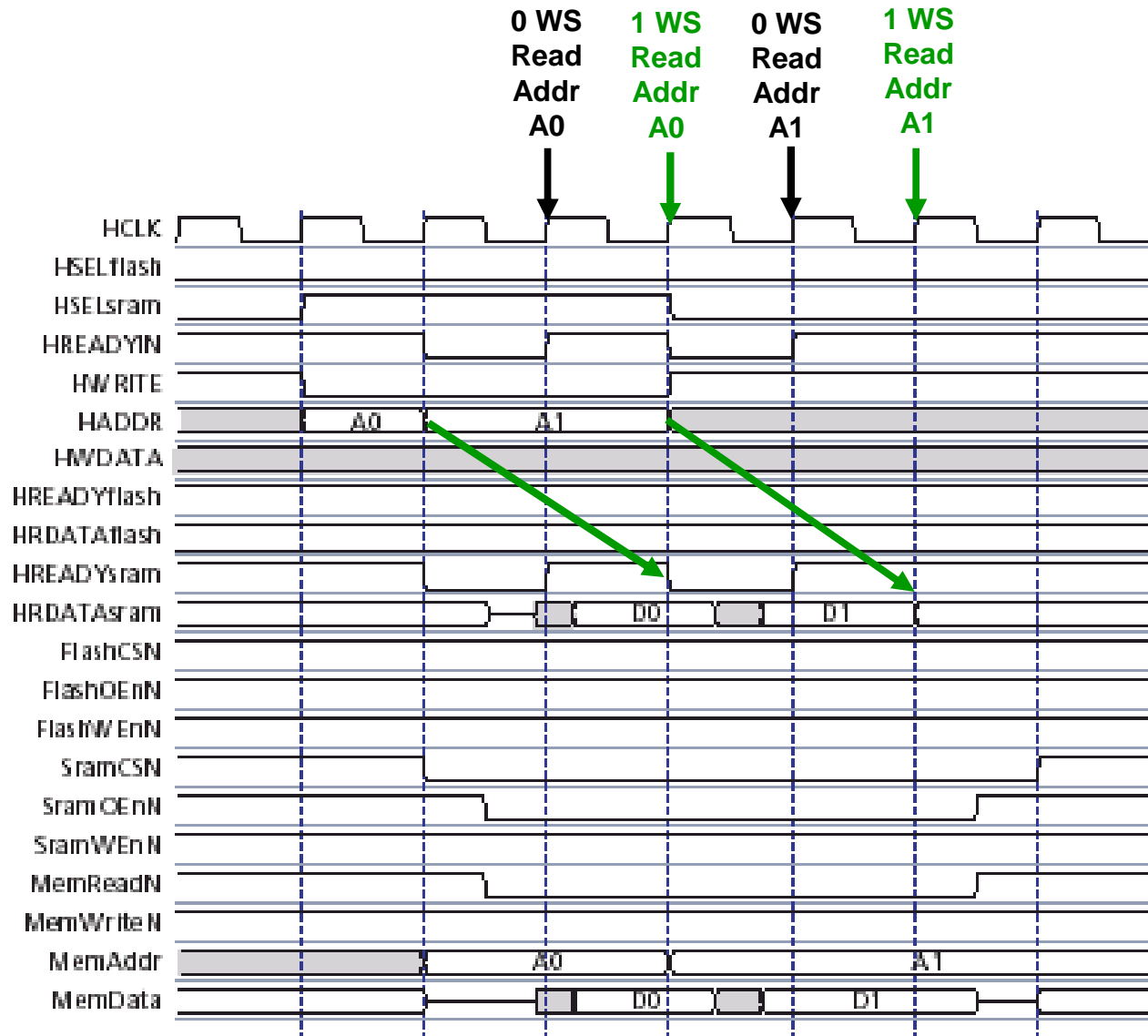
Asynchronous SRAM Interfacing

Write Wait States – 1-Wait-State Example



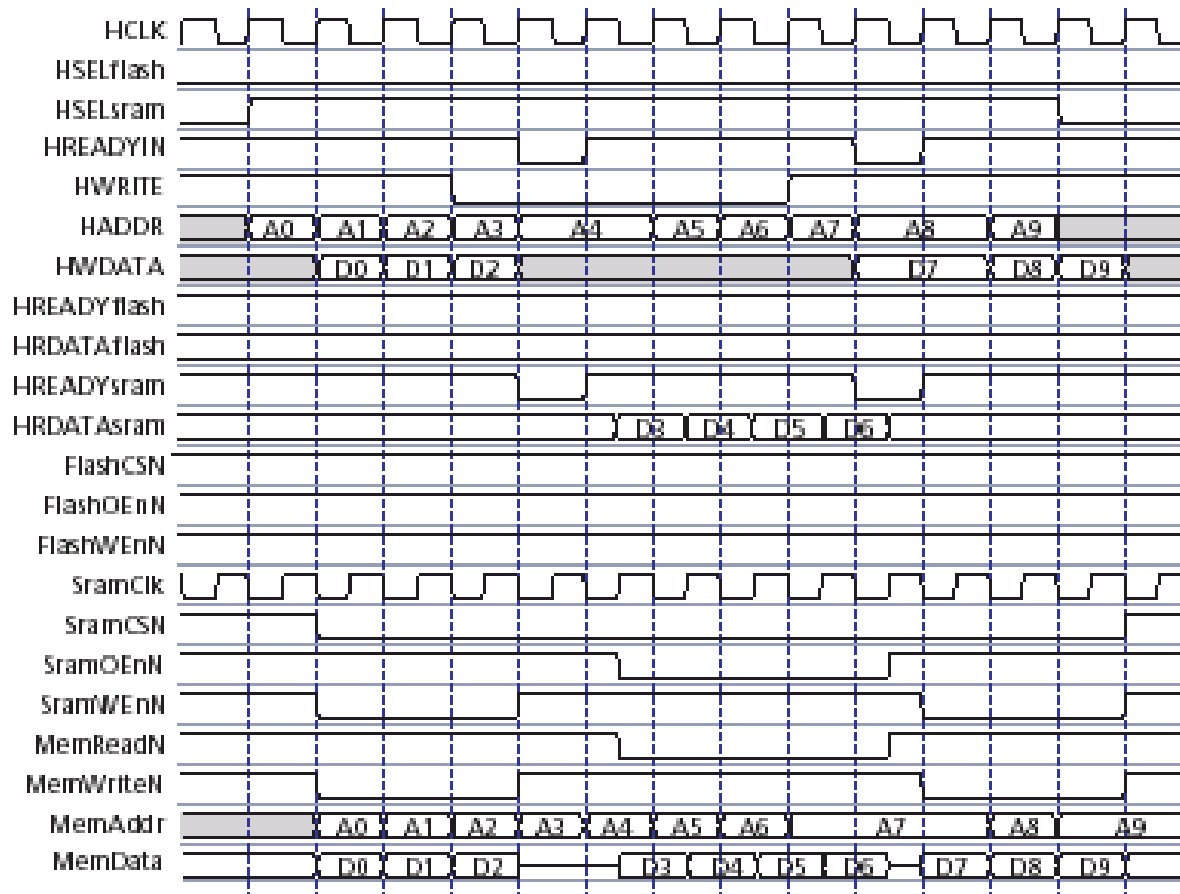
Asynchronous SRAM Interfacing

Read Wait States – 1-Wait-State Example



Synchronous SRAM Interfacing

Multiple Reads and Writes



CoreTimer

- APB Slave
- Dual 32-bit Timers
 - One Interrupt per Timer
- Can Be Extended to Include More Timers
- Free-running or Periodic Modes (One-shot Operation also Possible)

- Tile Count: 310 – 535

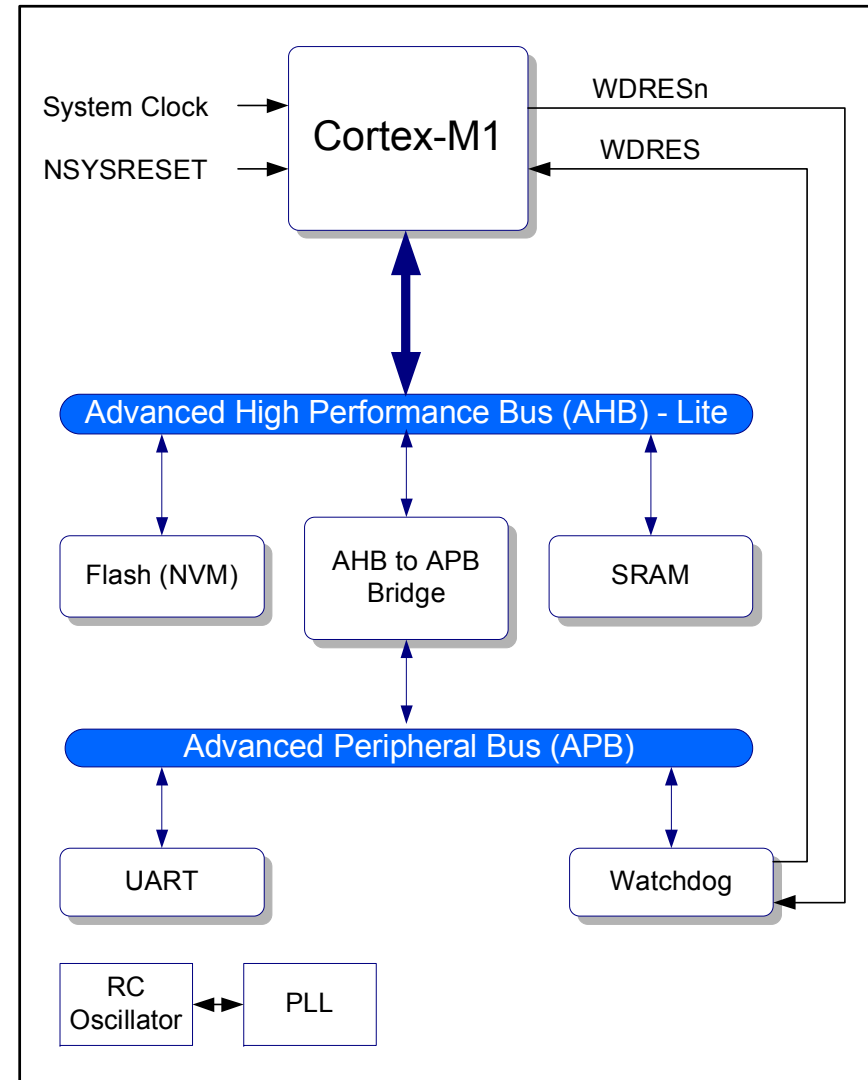
CoreGPIO

- APB Slave
 - 32-bit Output Register (Write Only)
 - 32-bit Input Register (Clear on Read)
-
- Tile Count: 100

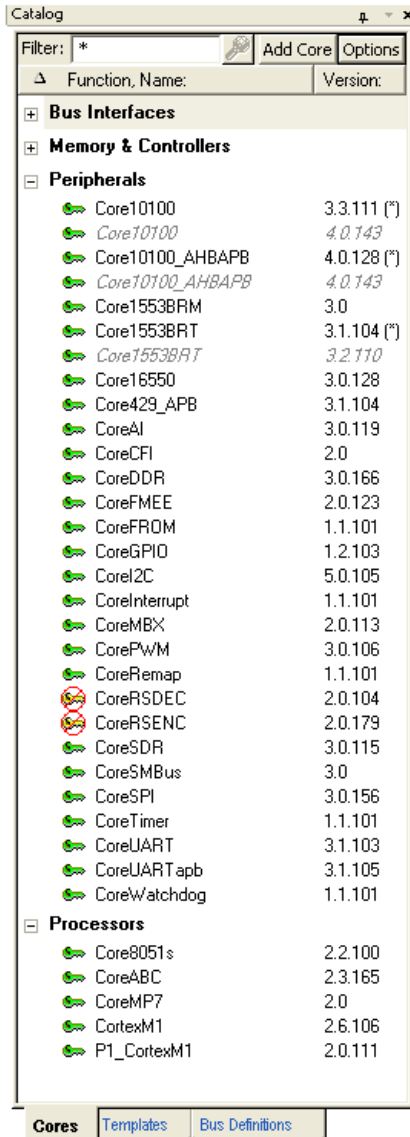
CoreWatchdog

- APB Slave
- Generates Interrupt at Programmed Interval
- If Interrupt Is Not Serviced, Watchdog Generates System Reset Signal
- Watchdog Can Be Enabled and Disabled
- If Synchronized Reset Is Not Available from System, Watchdog Should Be Used with MP7Bridge

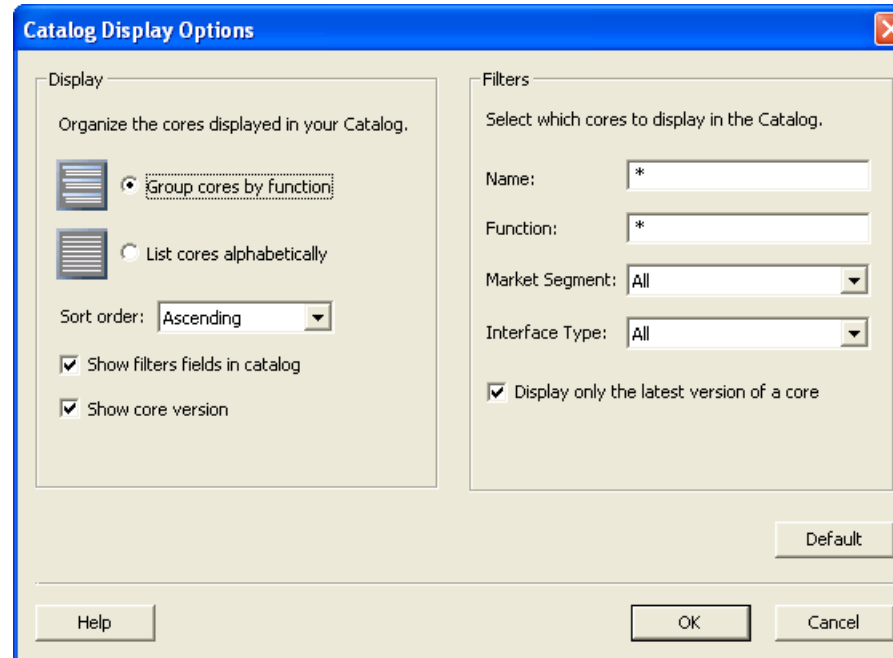
- Tile Count: 280 – 490



Component Versioning



- Where Available, Different Versions of Component May Be Selected
- Select Show Core Version from IP Catalog Options
 - Select Version You Want to use in Design
- Only One Version of Component Can Be in Project
 - DON'T MIX VERSIONS!



Component Help

- Libero Information Window Provides Essential Designer Information About Core Selected in Catalog
- Includes a Link to Documentation for Core

The screenshot displays two windows from the Libero IDE. The top window, titled 'Catalog', shows a list of cores with 'CoreAI' selected. The bottom window, titled 'Information Window', provides detailed information for the selected 'CoreAI' component.

Function, Name:	Version:
Core16550	3.0.128
Core429_APB	3.1.104
CoreAhbNvm	1.0
CoreAhbSram	1.0
CoreAI	2.1
CoreCFI	2.0
CoreFMEE	2.0.123
CoreFPU	1.1.104

Information Window

Project Manager New Features

Properties

Vendor: Actel
Library: DirectCore
Name: COREAI
Version: 2.1

Description:

CoreAI (Analog Interface) allows for simple control of the analog peripherals within the Fusion family of Actel devices. Control may be accomplished by using an internal or external microprocessor or microcontroller (such as Core8051 or CoreMP7), or it may be accomplished by user-created custom logic within the FPGA fabric. The industry standard AMBA APB slave interface is used as the primary control mechanism within CoreAI.

Version History:

2.1 Fixes a simulation issue that manifests itself when ADC FIFO is used with an obfuscated license.

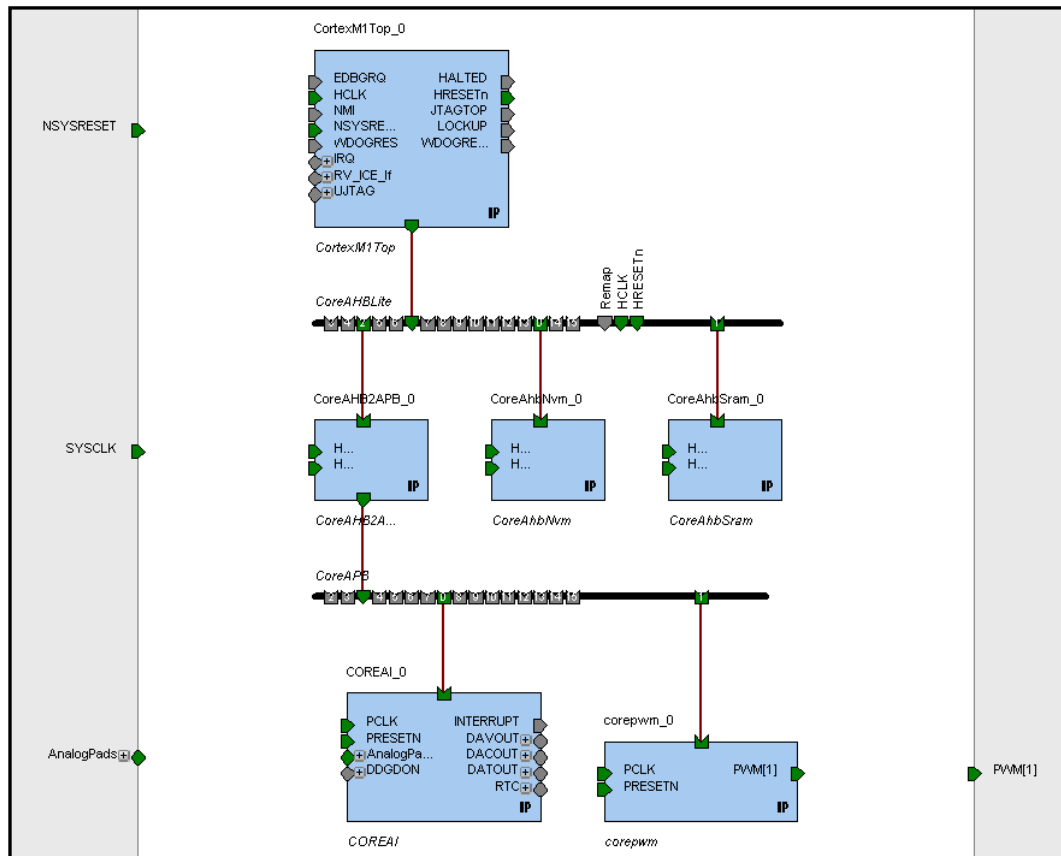
[Open documentation](#)



IP for ARM in Fusion

Building Fusion Systems

Overview

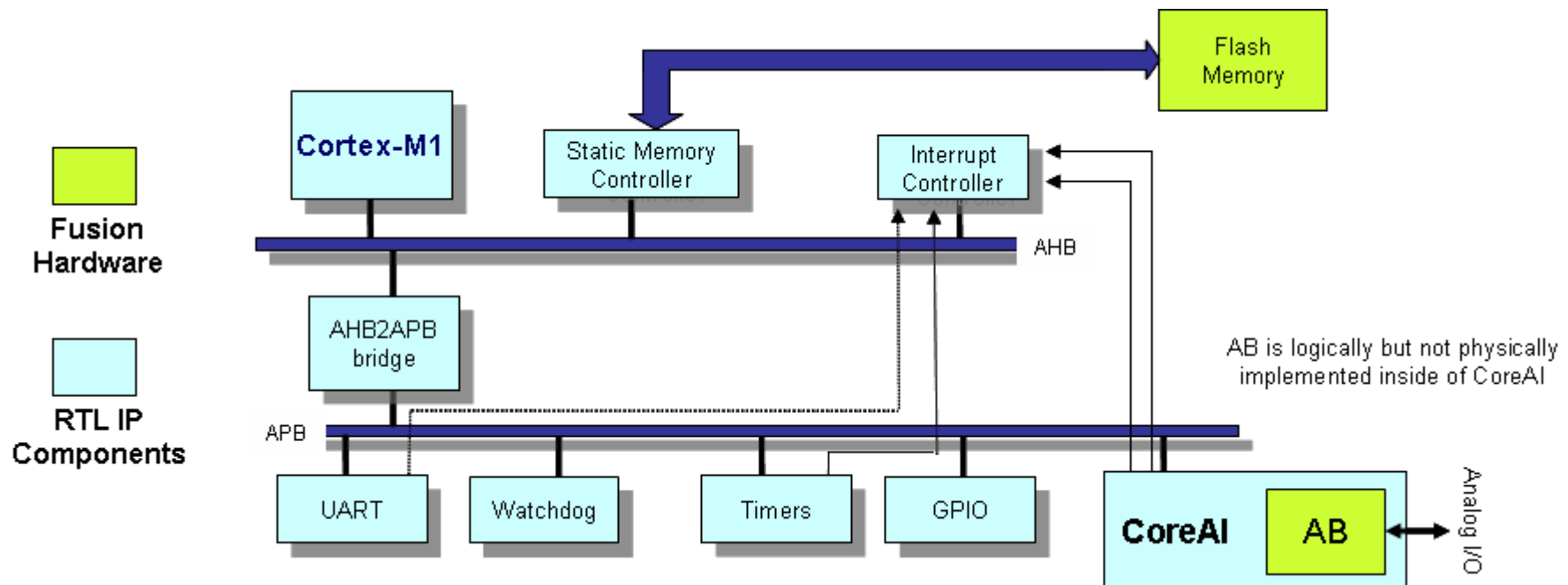


- CoreAI is Main Fusion-enabling Component
- Analog Signals Are Auto-stitched
- Analog Interface (AI) is Instantiated and Can Be Fully Configured inside SmartDesign
- ACM Configuration File is Output to SoftwareExport for Application Use

CoreAI

Analog Interface

- CoreAI Brings Cortex-M1 and Fusion Together
 - Available FREE as Part of SysBASIC IP
 - Configurable IP Block (150-460 tiles)
 - Simplifies Fusion Analog Interface to Cortex-M1
- Advantages for Designers
 - Parameterizable Control of Fusion Analog Block (AB) and I/O
 - Controlled by Cortex-M1 via APB Bus



CoreAI

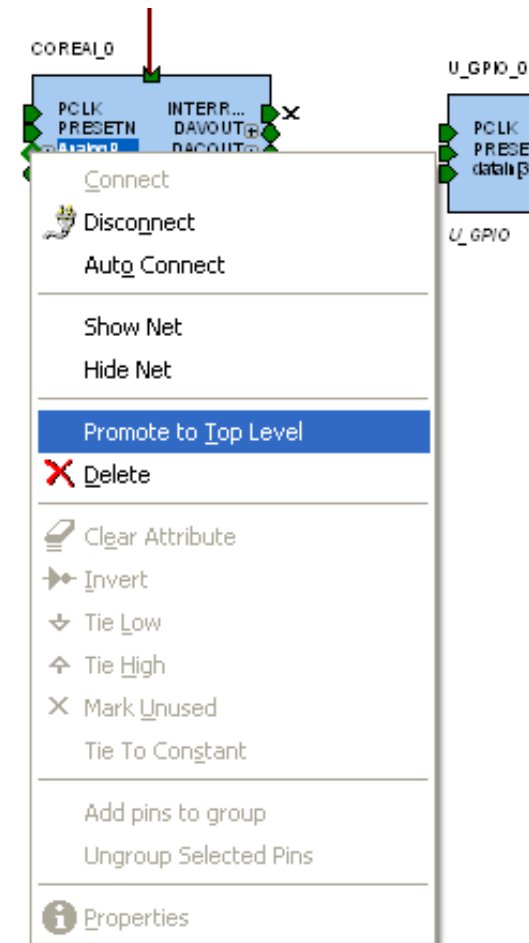
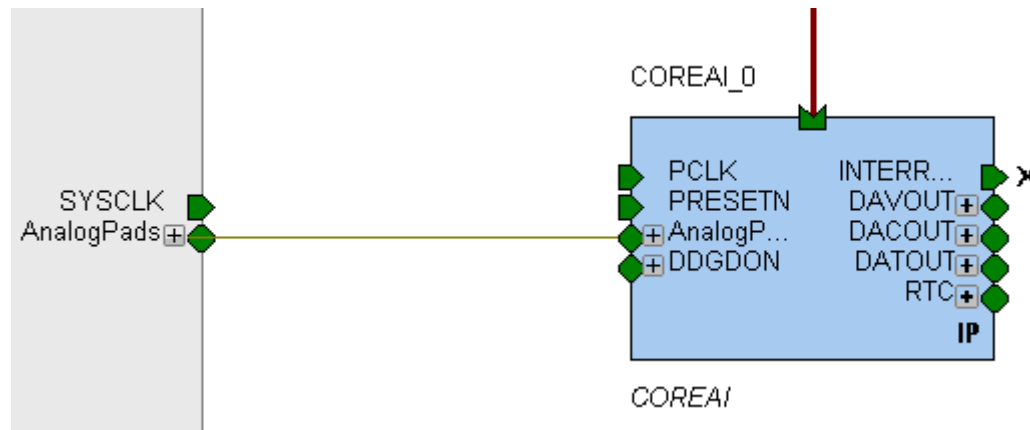
Features

- APB Slave
 - Connect to CoreAPB
- Instantiates Analog Block (AB)
- Configure via Dialog Box
 - Define How Analog Pins Are Used (0V to 4V Input, Current Monitoring, Temperature Monitoring, ...)
 - Has Implications for Required Connections to Top-level Analog Pads
- “AnalogPads” Port on CoreAI Groups All Connections to Analog Pads

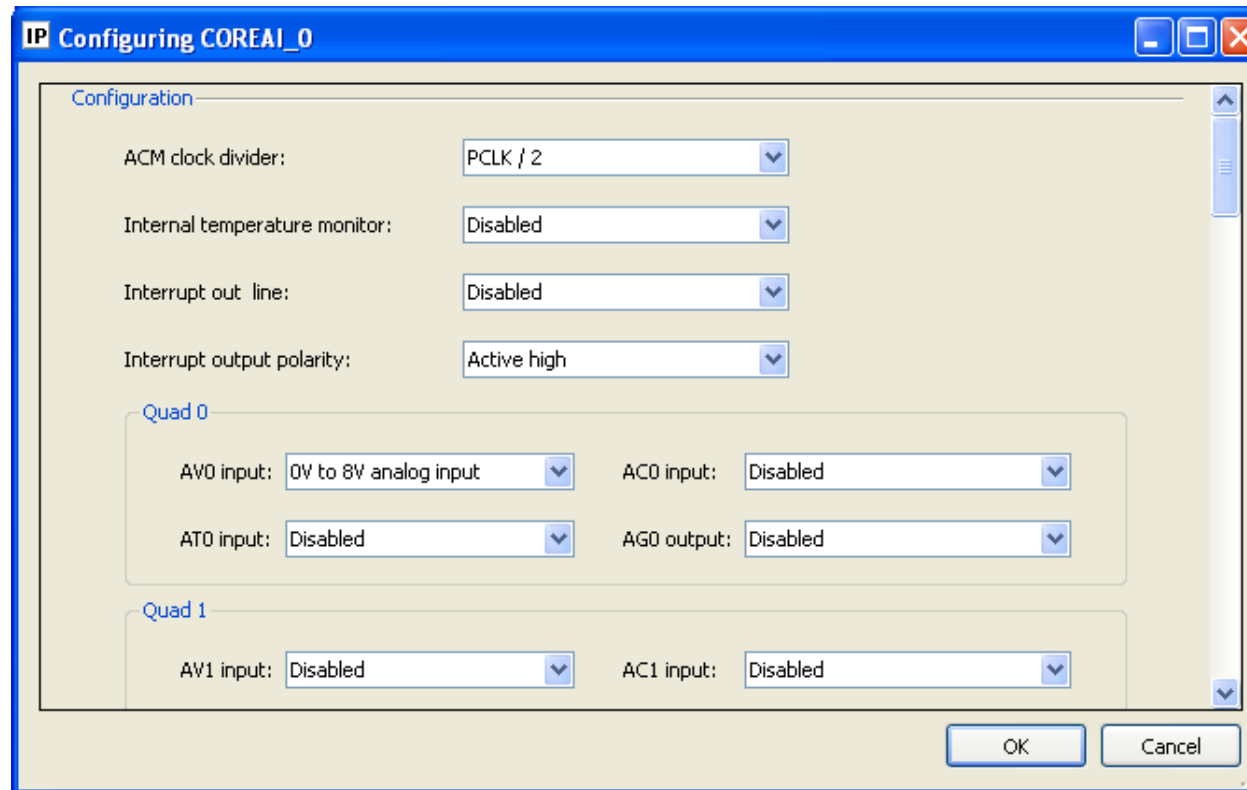
CoreAI Connections

AnalogPads Bus

- AnalogPads Contain 47 Individual Signals...
 - Route Connection to AnalogPads to Top Level of Your Design
 - Bus Connection to AnalogPads Labelled “AnalogPads” in Example Below
 - Possible to Connect Signals Individually – Not Advised!



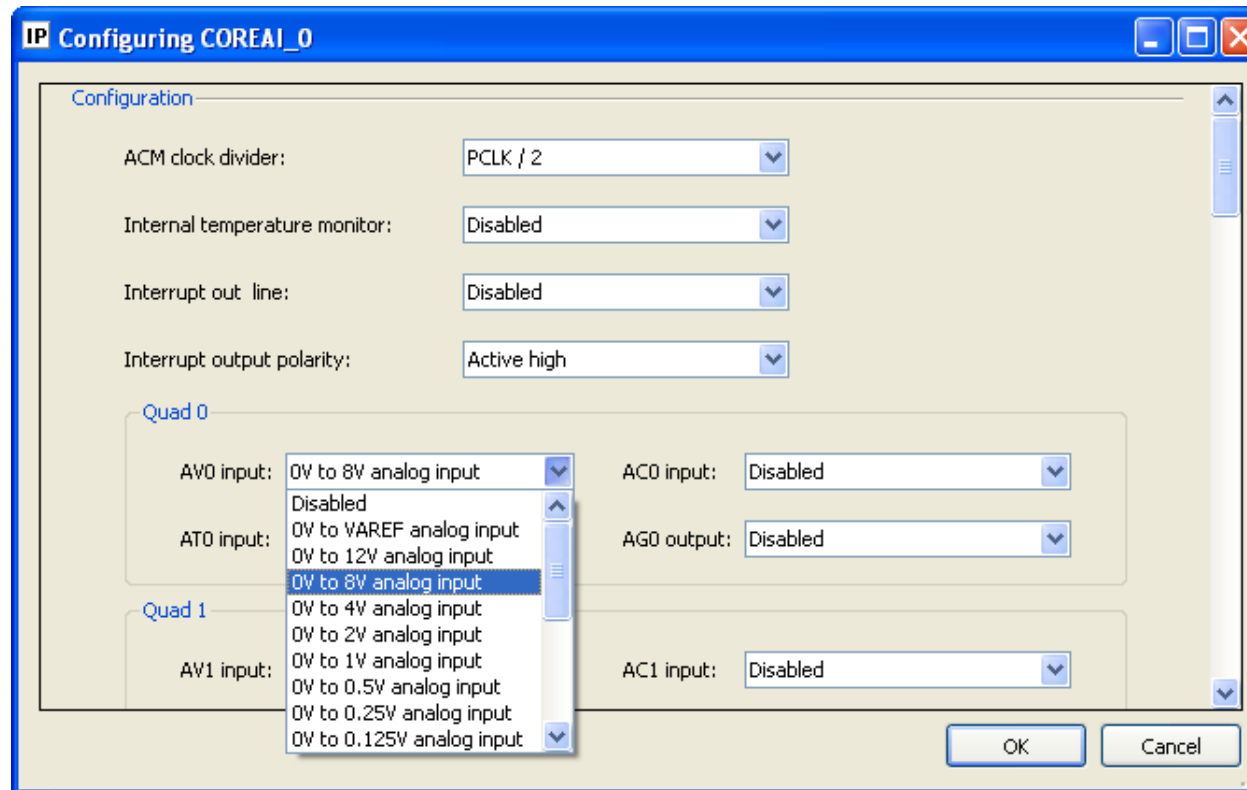
CoreAI Configuration Dialog



- ACM Clock Divider – PCLK Divided by 2, 4, 8 or 16
 - ACM Clock Frequency Must Not Exceed 10 MHz
- Internal Temperature Monitor – Enable/Disable
- Interrupt – Enable/Disable
- Interrupt Polarity – Active-high/Active-low

CoreAI Configuration

Quad Settings



- 10 Analog Quads (Quad 0 to Quad 9)
 - Analog Quad => 4-channel System for Pre-conditioning Analog Signals before Passing to ADC for Conversion into Digital Signal
 - Quads Can Be Configured via CoreAI Configuration Dialog

CoreAI Configuration

Additional Options

Real-Time Clock Usage
Select 'Yes' or 'No'

VAREF Selection
Internal 2.56v Reference
Voltage brought out on
VAREF Pin

ADC Resolution Control
Fixed (8-bit, 10-bit, or 12-bit)
or selectable via register

TVC Control
Clock divider fixed or
selectable via register

STC Control
Sample time fixed or
selectable via register

ADC FIFO Control
Select 'Yes' or 'No' and
almost full/empty values

APB Interface Width
16 bits (Fixed)

The screenshot shows the 'Configuring COREAI_0' dialog box with the following settings:

- Real Time Clock:** Use Real Time Clock: No
- ADC:**
 - VAREFSEL input control: Fixed
 - Fixed VAREFSEL value: Output 2.56V int ref on VAREF
 - ADC MODE control: Register controlled
 - ADC MODE fixed value: 10-bit
 - TVC[7:0] pins control: Register controlled
 - TVC[7:0] fixed value: 0
 - STC[7:0] pins control: Register controlled
 - STC[7:0] constant: 0
 - Use ADC conversions FIFO: No
 - ADC FIFO almost empty value: 0
 - ADC FIFO almost full value: 0
- APB:** APB interface width: 16 bits

Buttons: OK, Cancel

CoreAhbNvm, CoreAhbSram, CorePWM

- CoreAhbNvm (Fusion Only)
 - AHB Slave Providing Access to Nonvolatile (Flash) Internal Memory on Fusion Device
 - Configure Size (256KB, 512KB or 1MB) in SmartDesign
- CoreAhbSram (Fusion and ProASIC3/E)
 - AHB Slave which Allows AHB Master (Cortex-M1) to Access Internal SRAM
 - Size (2KB, 10KB, 14KB or 28KB) Configured in SmartDesign
 - (2KB of Internal SRAM Reserved for Cortex-M1)
- CorePWM (Any Family)
 - Not Fusion-specific but Has “Analog Flavor”
 - APB Slave Providing up to 8 Pulse-Width-Modulation Outputs (Motor Control, Tone Generation)

SmartDesign Files in Libero

Libero Project Complete - 1

The screenshot displays the Libero Project Manager interface for a project named "Cortex-M1_AFS_SOC_lab". The main window shows the "Project Flow" diagram, which illustrates the design process from "Design Entry Tools" (I/O Attribute Editor, HDL Editor, SmartDesign, ViewDraw) through "Source Files" to "Synthesis" (Synplify) and "Post-Synthesis Files". The flow then branches into "Simulation" (ModelSim) and "Stimulus" (WaveForm HDL Editor). The "Design Explorer" on the left shows the project hierarchy, with "M1BASIC_SOC" highlighted and circled, and a handwritten label "SmartDesign Component" pointing to it. The "Catalog" on the right lists various components like "Actel Macros", "Basic Blocks", and "Peripherals". The "Information Window" at the bottom shows the project name and version (8.5.0.34). The status bar at the bottom indicates the project configuration: "VHDL FAM: Fusion DIE: M1AF5600 PKG: 484 FBGA".

SmartDesign Files in Libero

Libero Project Complete - 2

The screenshot displays the Libero Project Manager interface for a project named "Cortex-M1_AFS_SOC_lab". The main window shows a "Project Flow" diagram with the following components:

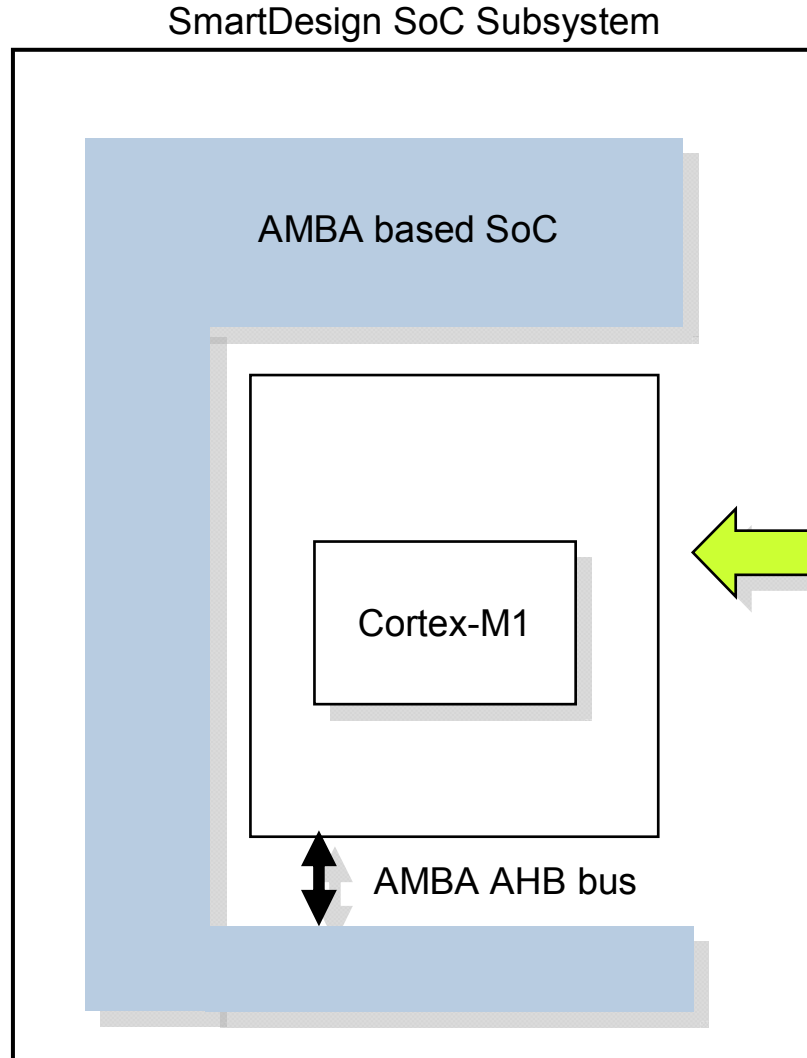
- Design Entry Tools:** I/O Attribute Editor, HDL Editor, SmartDesign, and ViewDraw.
- Source Files:** These feed into the Synthesis step.
- Synthesis:** Utilizes Synplify to generate Post-Synthesis Files.
- Simulation:** Utilizes ModelSim for simulation, with Waveform HDL files.
- Post-Synthesis Files:** These feed into the Simulation step.

The Design Explorer on the left shows the project hierarchy. A circle highlights the "Simulation Files" folder, which contains various scriptlet files (e.g., CoreAhbNvm_scriptlet.bfm) and memory files (e.g., nvm0.mem). An arrow points from this circle to the text "SmartDesign Output".

The Project Manager interface also includes a Catalog on the right with categories like Actel Macros, Basic Blocks, and Bus Interfaces. The Information Window at the bottom displays the version (8.5.0.34) and release (v8.5) information.

Bus-Functional Models

SoC Testing with BFM



Everything Is Generated by SmartDesign !

Test Script
Human
Readable

```
memmap      uart      0xC3000000;  
write       b uart    0x0C 0x06;  
read        b uart    0x08;  
readcheck   b uart    0x0C 0x06;
```

Editable by User to Add His Own Test Functionality

BFM Features

- Allows User to Test Integration of Subsystem
 - Test that All Peripherals Can Be Accessed
 - Test Address Decoding and Bus Connections

- Pin-compatible with Cortex-M1 Processor Core
 - Primarily AMBA Interface

- Models Big-/Little-endian Operation

- BFM Includes Timing Shell
 - Facilitates Back-annotation of Delay Information during Post-synthesis Simulation

- Support for Interrupts

BFM Test Script

- SmartDesign Generates 'Connectivity Testing' BFM Test Script
 - HDL- and Simulator-independent
- User Can Extend Test Script
 - Write Specific Values to Control Registers of a Peripheral
 - Read Status Registers and Verify against Expected Data
- Uses User-specified Resource Names for Ease of Understanding

```
# write to ssram resource (offset 20)
write w ssram 0x20 0x55555555

# read back from same location
# (expect same data)
read w ssram 0x20 0x55555555

# write to flash resource (offset 100)
write h flash 0x100 0xaaaa

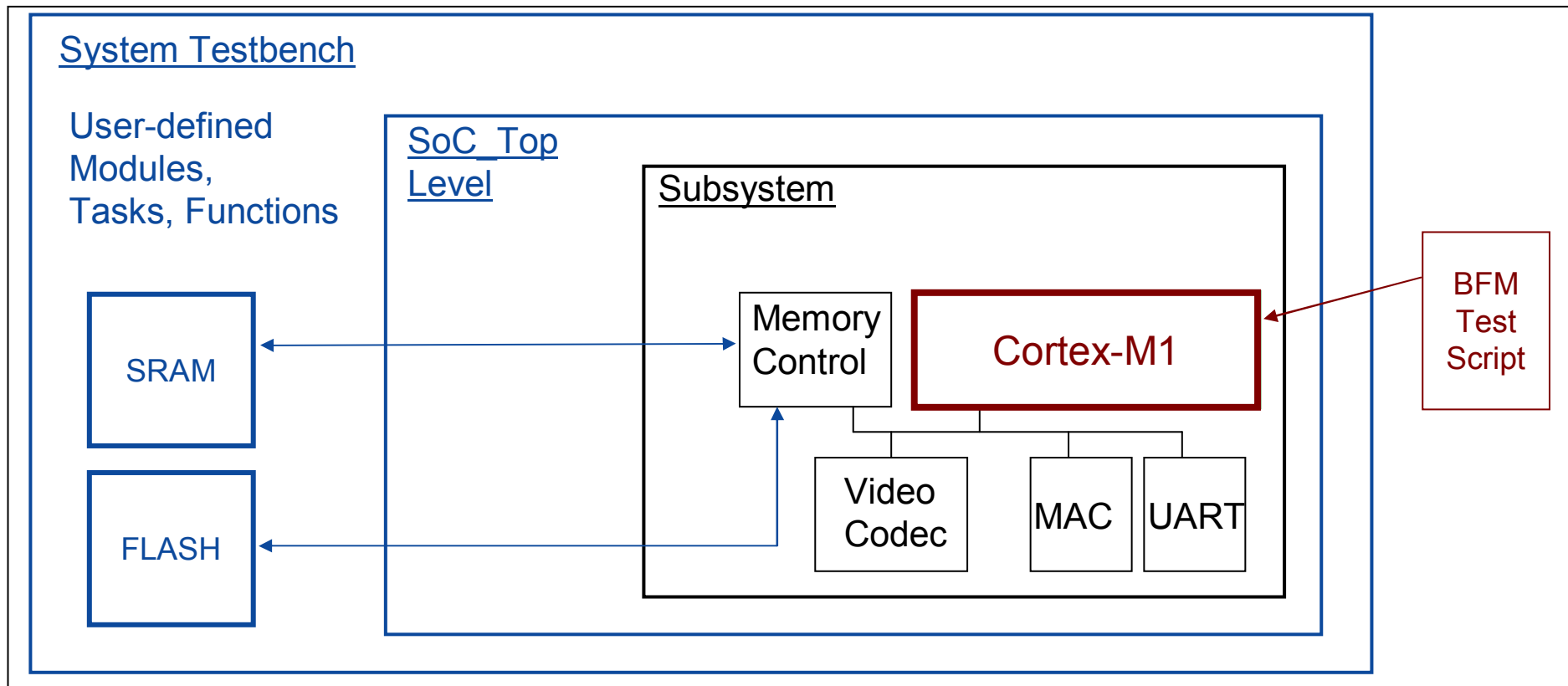
# read back from same location
# (expect same data)
read h flash 0x100 0xaaaa
```

Cortex-M1 Signals Modeled by BFM

- General
 - NSYSRESET
 - SYSCLK
 - RESETn
- AMBA AHB Bus
 - HADDR
 - HSIZE
 - HTRANS
 - HWRITE
 - HWDATA
 - HRDATA
 - HBURST
 - HMASTLOCK
 - HPROT
 - HREADY
 - HRESP
- TCM Interface
- Embedded Trace Interfaces Are NOT Modeled by BFM
 - Often These Are Not Used Anyway!

SoC System Testbench

- SmartDesign Output (Black)
- User Adds Modules, Tasks (Blue)
- BFM Is Embedded within SoC (Red)



BFM Log Output

- BFM Outputs Log Messages to Simulator (ModelSim) Console

```
# write to ssram resource (offset 20)
write 55555555 to offset 20 in ssram
# read back from same location
read data = 55555555, as expected
# write to flash resource (offset 100)
Write aaaa to offset 100 in flash
# read back from same location
read data = aaaa, as expected
# Wait for ARM interrupt
Interrupt detected
Write 22 to offset 4 in videoCodec
# read back from same location
Error in read of videoCodec
Expected 22, actual value 20
```



BFM Scripts

Top-Level BFM Script

subsystem.bfm

```
# memmap resource_name base_address;
#
# write width resource_name byte_offset data;
# read width resource_name byte_offset;
# readcheck width resource_name byte_offset data;
#
#-----
# Memory Map
# Define name and base address of each resource.
#-----

memmap CoreMemCtrl_00 0x0;
memmap CoreMemCtrl_00 0x10000000;
memmap CoreAhbSram_00 0x30000000;
memmap CoreGPIO_00 0x40000000;
memmap CoreUARTapb_00 0x41000000;

#-----
# Include resource scriptlets
#-----

include CoreMemCtrl CoreMemCtrl_00;
include CoreMemCtrl CoreMemCtrl_00;
include CoreAhbSram CoreAhbSram_00;
include CoreGPIO CoreGPIO_00;
include CoreUARTapb CoreUARTapb_00;
```

Template

- Actual Read and Write Commands Are in Scriptlets Referenced by include Statements
- include References File with Added Suffix “_scriptlet.bfm”

System memory map

References peripheral listed in memory map

References File

CoreGPIO_scriptlet.bfm

BFM Scriptlet Example

CoreAhbSram scriptlet.bfm

```
#-----  
# Resource: CoreAhbSram  
# Instance: VAR_resource  
#-----#  
# BFM scriptlet for CoreAhbSram  
# Test byte/halfword/word writes and reads  
write  w VAR_resource 0x0 0x12345678;  
readcheck w VAR_resource 0x0 0x12345678;  
write  b VAR_resource 0x0 0x9a;  
readcheck w VAR_resource 0x0 0x1234569a;  
readcheck b VAR_resource 0x0 0x9a;  
readcheck b VAR_resource 0x1 0x56;  
readcheck b VAR_resource 0x2 0x34;  
readcheck b VAR_resource 0x3 0x12;  
readcheck h VAR_resource 0x0 0x569a;  
readcheck h VAR_resource 0x2 0x1234;  
write  b VAR_resource 0x1 0xbc;  
write  b VAR_resource 0x2 0xde;  
write  b VAR_resource 0x3 0xf0;  
readcheck w VAR_resource 0x0 0xf0debc9a;  
write  w VAR_resource 0x4 0xaabbccdd;  
readcheck h VAR_resource 0x6 0xaabb;
```

- VAR_resource Is Dummy Instance Name
- Actual Name Is Referenced in subsystem.bfm

BFM Script Command

`write`

■ Function

- BFM Performs Write to Specified Offset within Memory Map of Specified Resource

■ Syntax

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

where

width – W (Word), H (Halfword), B (Byte)

resource_name – User-defined Resource Name

byte_offset – Hex Offset from Resource Address Base

data – Hex Value of Data to Be Written

■ Example

```
write W VideoCodec 20 11223344;
```

BFM Script Command

read

■ Function

- BFM Performs Read from Specified Offset within Memory Map of Specified Resource

■ Syntax

```
read width resource_name byte_offset;
```

where

width – W (Word), H (Halfword), B (Byte)

resource_name – User-defined Resource Name

byte_offset – Hex Offset from Resource Address Base

■ Example

```
read W VideoCodec 20;
```

BFM Script Command

readcheck

■ Function

- BFM Performs Read from Specified Offset within Memory Map of Specified Resource

■ Syntax

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

where

width – W (Word), H (Halfword), B (Byte)

resource_name – User-defined Resource Name

byte_offset – Hex Offset from Resource Address Base

data – Hex Value of Expected Data

■ Example

```
readcheck W VideoCodec 20 11223344;
```

BFM Script Command

`wait`

■ Function

- BFM Stalls (Doesn't Initiate Any Bus Transactions) for Specified Number of Clock Periods

■ Syntax

```
wait num_clock_ticks;
```

where

`num_clock_ticks` – Number of Clock Cycles that BFM Stalls

■ Example

```
wait 20;
```

BFM Script Command

poll

- Function

- Continuously Reads Specified Location until Requested Value Is Obtained

- Syntax

```
poll width resource_name byte_offset data_bitmask;
```

where

width – W (Word), H (Halfword), B (Byte)

resource_name – User-defined Resource Name

byte_offset – Hex Offset from Resource Address Base

data_bitmask – Bitmask Is ANDed with Read Data and Result Is Compared with Bitmask Itself

- If Equal, Poll Command Is Complete
- If Not Equal, Polling Continues

- Example

```
poll W VideoCodec 40 567890AB;
```

Accessing Peripheral Locations

- Writes / Reads to Memory Locations Are Valid Accesses
 - Example – CoreAHBSram
- Writes / Reads to Meaningful Register Addresses Have the Desired Effect on a Peripheral
 - Register Locations Have Specific Meanings in Core
 - Example – CoreTimer Used in Lab
- Reads to Unused Register Addresses Still Return Data Written ...
 - ... even if No Actual Register Is Present
 - NO Checking for Invalid Addresses



Modifying Test Scripts

User Modifications

- Users Can Modify `subsystem.bfm` or Scriptlets to Add New Tasks or Test Additional Functionality
- BFM Scripts Are Overwritten When Core is Re-generated in SmartDesign

User Modifications

Scriptlets

- Open File in Libero HDL Editor and Add New Tasks
 - Perform “Save As” on Modified Scriptlet
 - Name Convention: <peripheral_name>_scriptlet.bfm

Modified Scriptlet Saved as CoreTimer_user_scriptlet.bfm

```
#-----  
# Resource: CoreTimer  
# Instance: VAR_resource  
#-----  
write    W    VAR_resource    0x00 0x00000100;  
readcheck W VAR_resource    0x00 0x00000100;  
readcheck W VAR_resource    0x04 0x00000100;  
write    h    VAR_resource    0x0C 0x0001;  
readcheck h VAR_resource    0x0C 0x0001;  
write    h    VAR_resource    0x08 0x0003;  
readcheck h VAR_resource    0x08 0x0003;  
poll     b    VAR_resource    0x014 0x01;  
read    h    VAR_resource    0x04;  
write    h    VAR_resource    0x10 0x1234;
```

User Modifications

subsystem.bfm

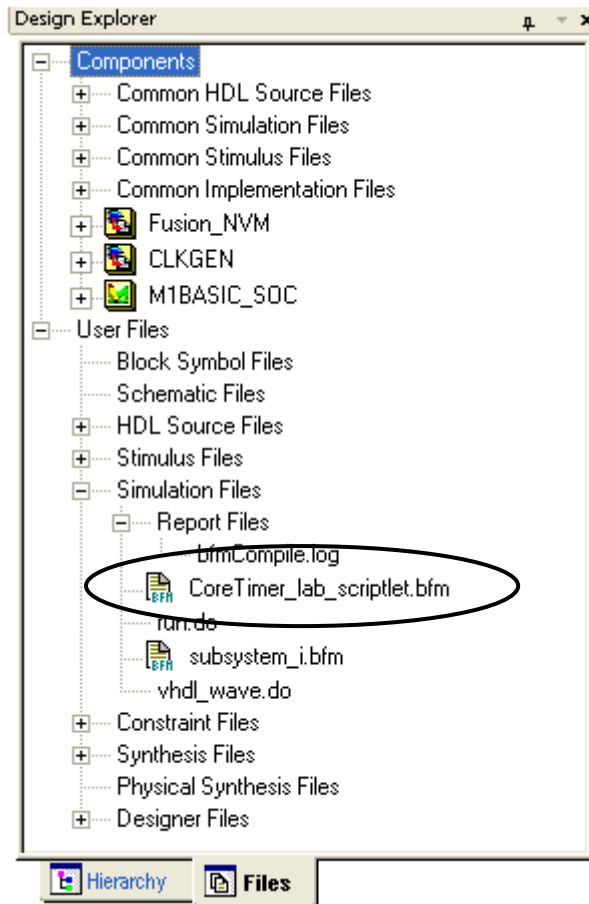
- Modify **subsystem.bfm** to use Modified Scriptlets
 - Perform “Save As” on Modified File

```
#-----  
# Memory Map  
# Define name and base address of each resource.  
#-----  
  
memmap CoreAhbSram_00 0x00000000;  
memmap CoreAhbSram_01 0x10000000;  
memmap CoreGPIO_00 0xc2000000;  
memmap CoreUARTapb_00 0xc3000000;  
memmap CoreTimer_00 0xc4000000;  
  
#-----  
# Include resource scriptlets  
#-----  
  
include CoreAhbSram CoreAhbSram_00;  
include CoreAhbSram CoreAhbSram_01;  
include CoreGPIO CoreGPIO_00;  
include CoreUARTapb CoreUARTapb_00;  
include CoreTimer_user CoreTimer_00;
```

Use modified scriptlet

Modified BFM Files in Libero

- Files Appear Under User Files After “Save As”




Modified Files appear
under User Files in Libero



Simulation

BFM Log Output

- BFM Outputs Log Messages to ModelSim Transcript Window

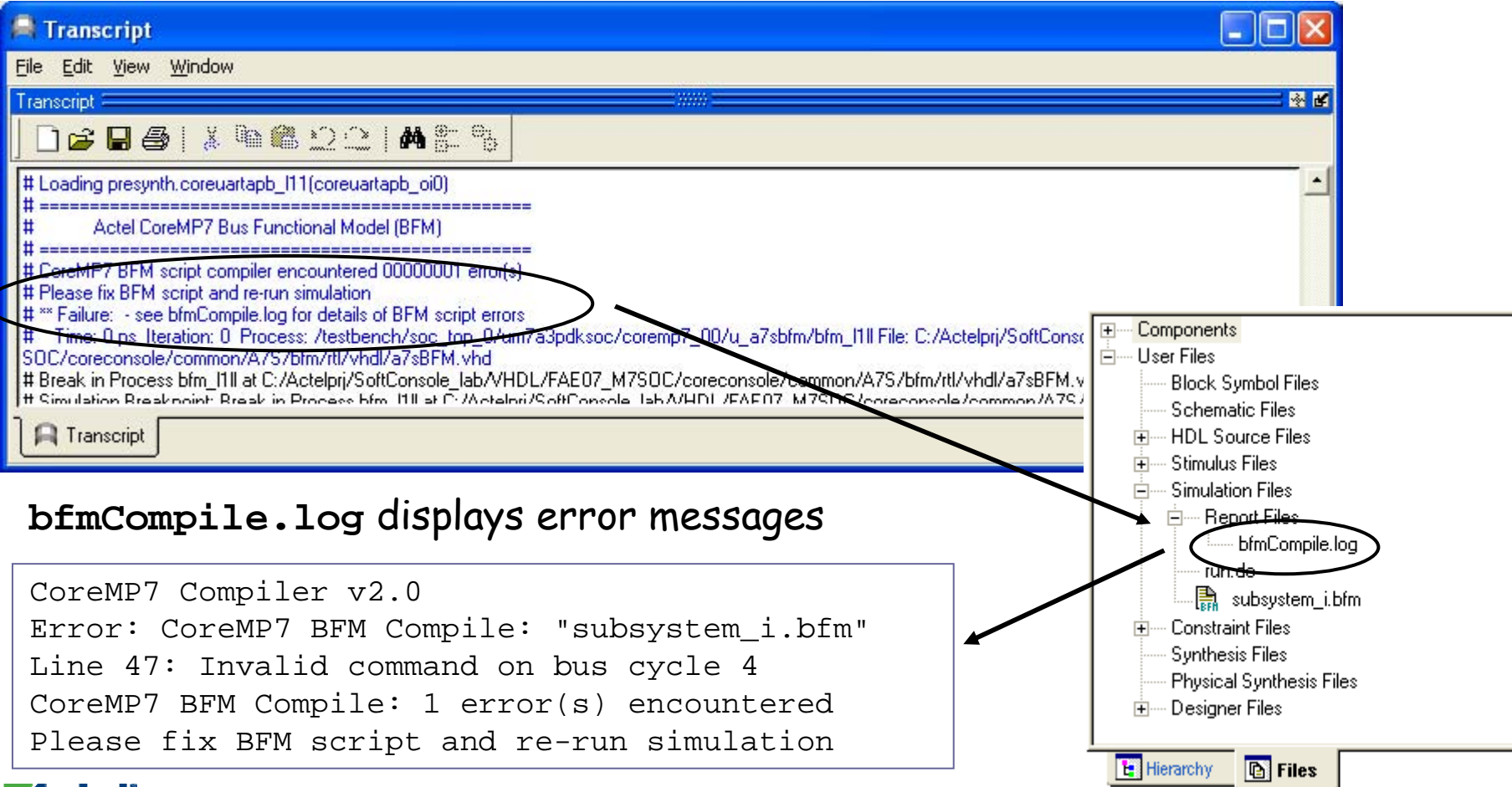


The screenshot shows the ModelSim Transcript window with the following text:

```
# =====  
# Actel CortexM1 Bus Functional Model (BFM)  
# =====  
# ** Warning: BUSY SIGNAL DE-ASSERTION IN SIMULATION MAY NOT MATCH WITH SILICON BEHAVIOR.  
# Time: 0 ps Iteration: 0 Instance: /testbench/soc_top_0/um1basic_soc/coreahbnvm_0/cahbnvml00  
# ** Warning: USER LOGIC SHOULD POLL FOR BUSY TO ENSURE BEHAVIOR CONSISTENT WITH SILICON.  
# Time: 0 ps Iteration: 0 Instance: /testbench/soc_top_0/um1basic_soc/coreahbnvm_0/cahbnvml00  
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).  
# Time: 0 ps Iteration: 0 Instance: /testbench/soc_top_0/um1basic_soc/coreuartapb_0  
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).  
# Time: 0 ps Iteration: 0 Instance: /testbench/soc_top_0/um1basic_soc/coreuartapb_0  
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).  
# Time: 0 ps Iteration: 0 Instance: /testbench/soc_top_0/um1basic_soc/coreuartapb_0  
# ** Warning: ****NVM: RESET went unknown ****  
# Time: 0 ps Iteration: 2 Instance: /testbench/soc_top_0/um1basic_soc/coreahbnvm_0/cahbnvml00  
# ** Warning: ****NVM: RESET went unknown ****  
# Time: 0 ps Iteration: 3 Instance: /testbench/soc_top_0/um1basic_soc/coreahbnvm_0/cahbnvml00  
# ----- Execution of BFM Script Started -----  
# N-cycle: Write 0000022C to address C4000000  
# N-cycle: Read 0000022C from address C4000000  
# N-cycle: Read 0000022C from address C4000004  
# N-cycle: Write 0001 to address C400000C  
# N-cycle: Read 0001 from address C400000C  
# N-cycle: Write 0003 to address C4000008  
# N-cycle: Read 0003 from address C4000008  
# Polling address C4000014 for bitmask 01  
# Poll completed - read 01 from address C4000014  
# N-cycle: Read 022B from address C4000004  
# N-cycle: Write 1234 to address C4000010  
# Polling address C4000014 for bitmask 01  
# Poll completed - read 01 from address C4000014  
# N-cycle: Read 022C from address C4000004  
# N-cycle: Write 1234 to address C4000010  
# Polling address C4000014 for bitmask 01  
# Poll completed - read 01 from address C4000014  
# N-cycle: Read 022B from address C4000004  
# N-cycle: Write 1234 to address C4000010  
# ----- Successful Execution of BFM Script Complete -----  
# ** Failure: Breakpoint encountered - normal completion of BFM-driven simulation  
# Time: 457892825 ns Iteration: 1 Process: /testbench/soc_top_0/um1basic_soc/cortexm1_top_0/cortexm1_top_0/bfm_core_line_1430 File: C:\Actel\src\Cortex-M1_AES_SOC_lab\Soluti
```

BFM Simulation Errors and Error Messages

- BFM Simulation Error Messages Reference
bfmCompile.log



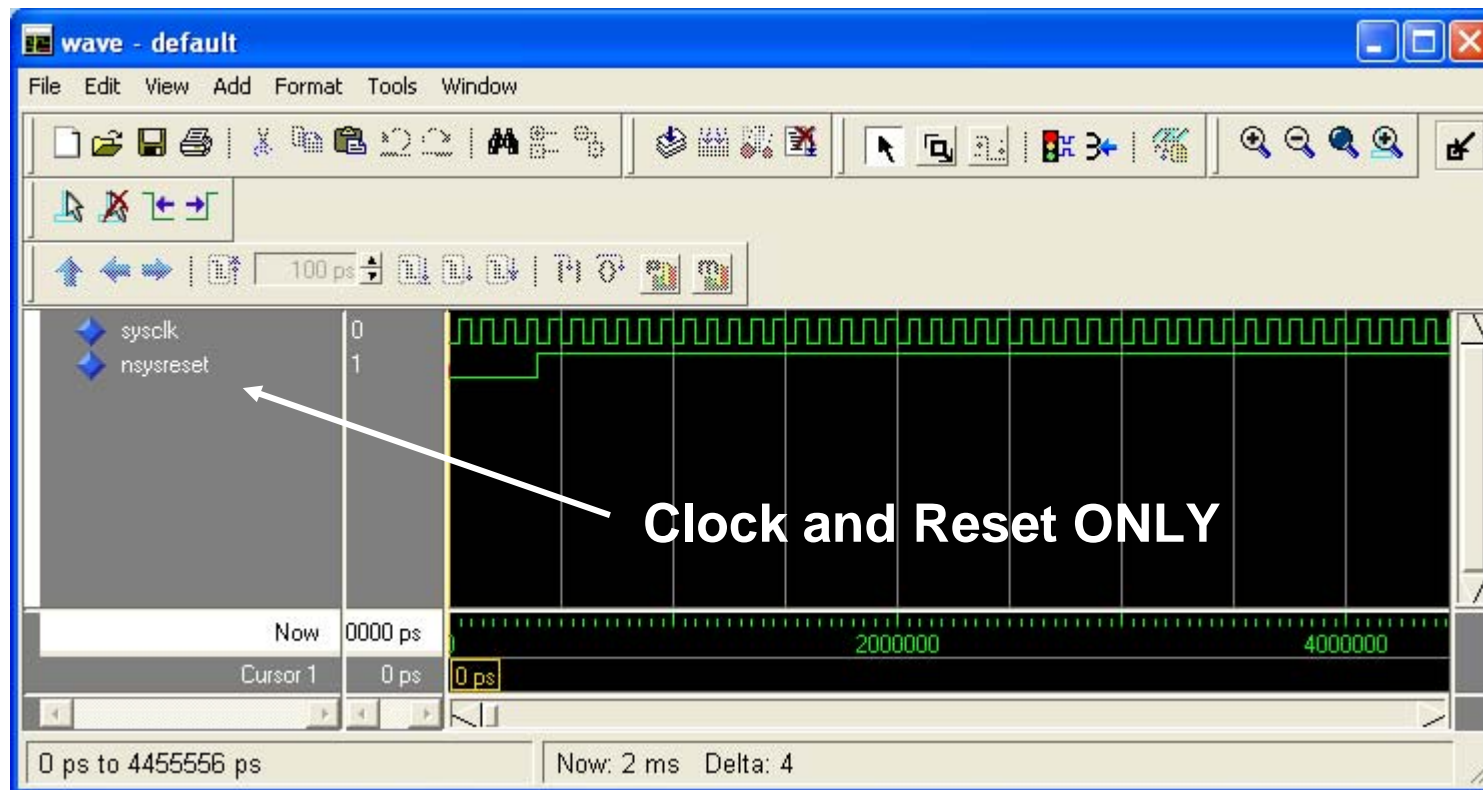
The image shows a screenshot of a software interface. On the left is a 'Transcript' window with a menu bar (File, Edit, View, Window) and a toolbar. The transcript text includes: '# Loading presynth.coreuartpb_i11(coreuartpb_oio)', '# =====', '# Actel CoreMP7 Bus Functional Model (BFM)', '# =====', '# CoreMP7 BFM script compiler encountered 00000001 error(s)', '# Please fix BFM script and re-run simulation', '# ** Failure: - see bfmCompile.log for details of BFM script errors', '# Time: 0 ps Iteration: 0 Process: /testbench/soc_top_0/am7a3pdk/soc/coremp7_00/u_a7sbfm/bfm_i11 File: C:/Actelprj/SoftConst/SOC/coreconsole/common/A7S/bfm/rtl/vhdl/a7sBFM.vhd', '# Break in Process bfm_i11 at C:/Actelprj/SoftConsole_lab/VHDL/FAE07_M7SOC/coreconsole/common/A7S/bfm/rtl/vhdl/a7sBFM.v', and '# Simulation Breakpoint: Break in Process bfm_i11 at C:/Actelprj/SoftConsole_lab/VHDL/FAE07_M7SOC/coreconsole/common/A7S/bfm/rtl/vhdl/a7sBFM.v'. A red circle highlights the error message lines. On the right is a file browser window showing a tree structure: Components, User Files, Block Symbol Files, Schematic Files, HDL Source Files, Stimulus Files, Simulation Files, Report Files (with 'bfmCompile.log' circled in red), run.do, subsystem_i.bfm, Constraint Files, Synthesis Files, Physical Synthesis Files, and Designer Files. Arrows point from the circled error message in the transcript to the circled 'bfmCompile.log' file in the file browser.

bfmCompile.log displays error messages

```
CoreMP7 Compiler v2.0
Error: CoreMP7 BFM Compile: "subsystem_i.bfm"
Line 47: Invalid command on bus cycle 4
CoreMP7 BFM Compile: 1 error(s) encountered
Please fix BFM script and re-run simulation
```

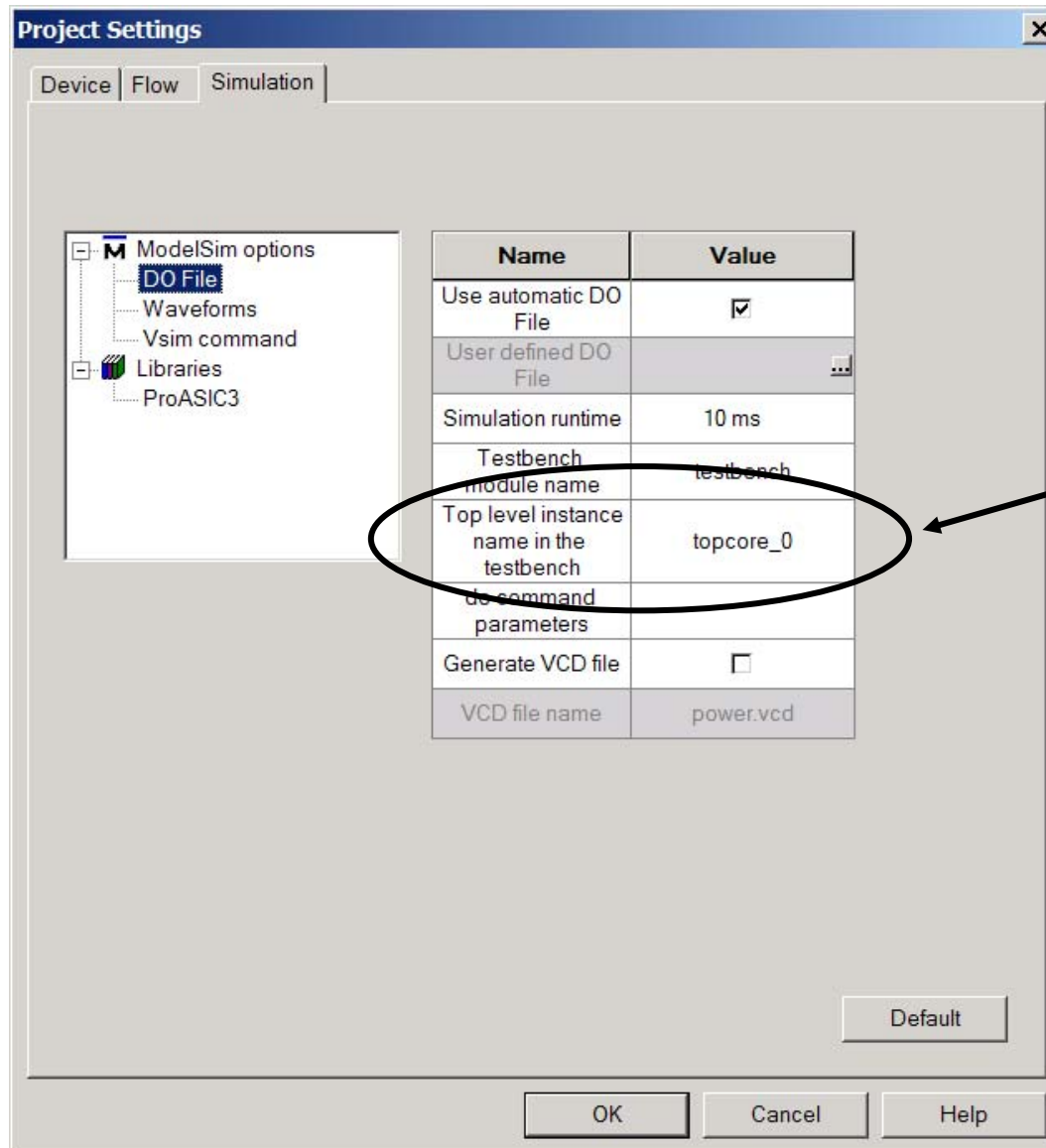
Initial BFM Simulation

- By Default Only `nsysreset` and `sysclk` Appear in the Wave Window



Viewing All Cortex-M1 Signals

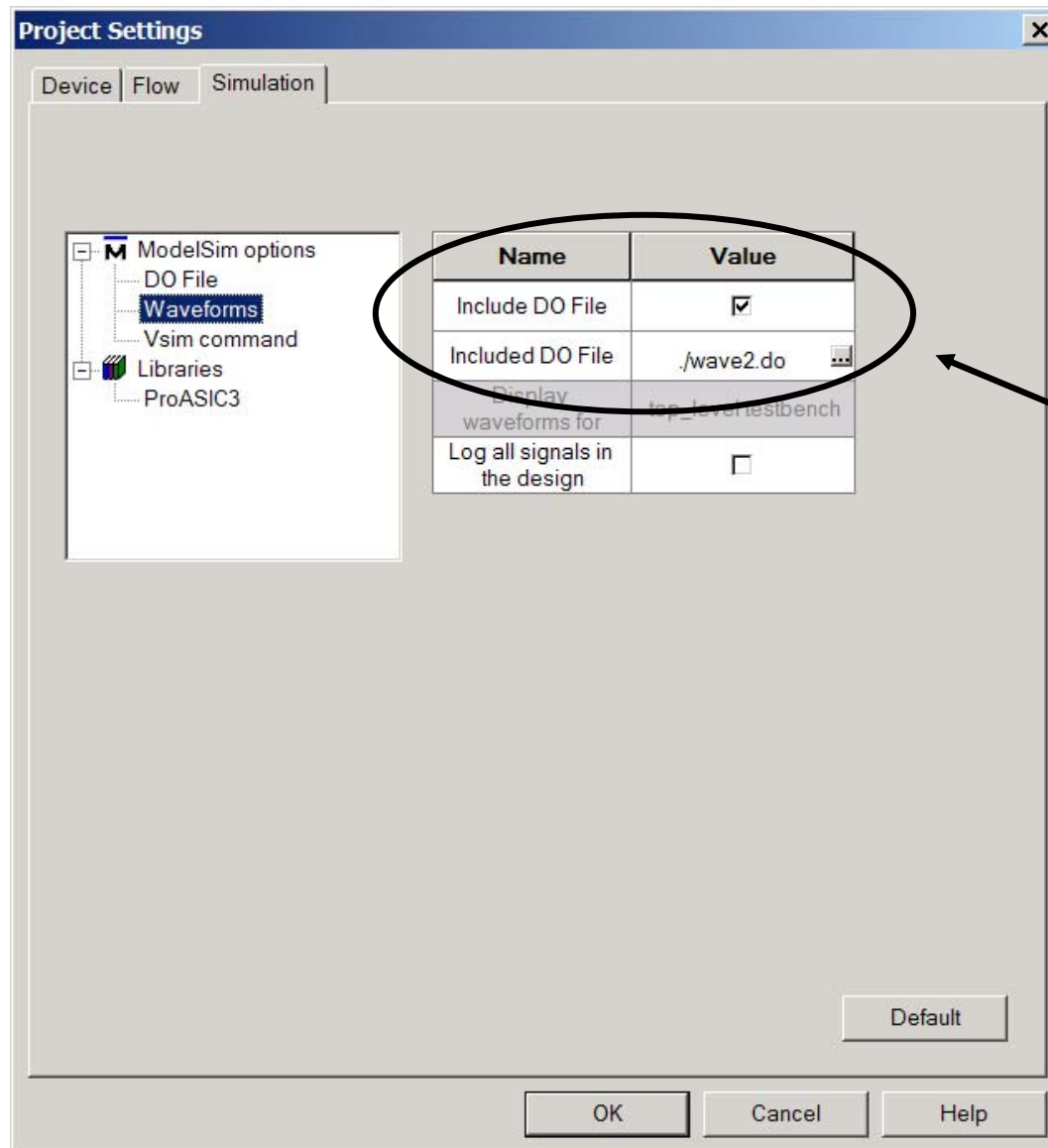
Option 1 – Change Top-Level Instance



Use DUT Instance

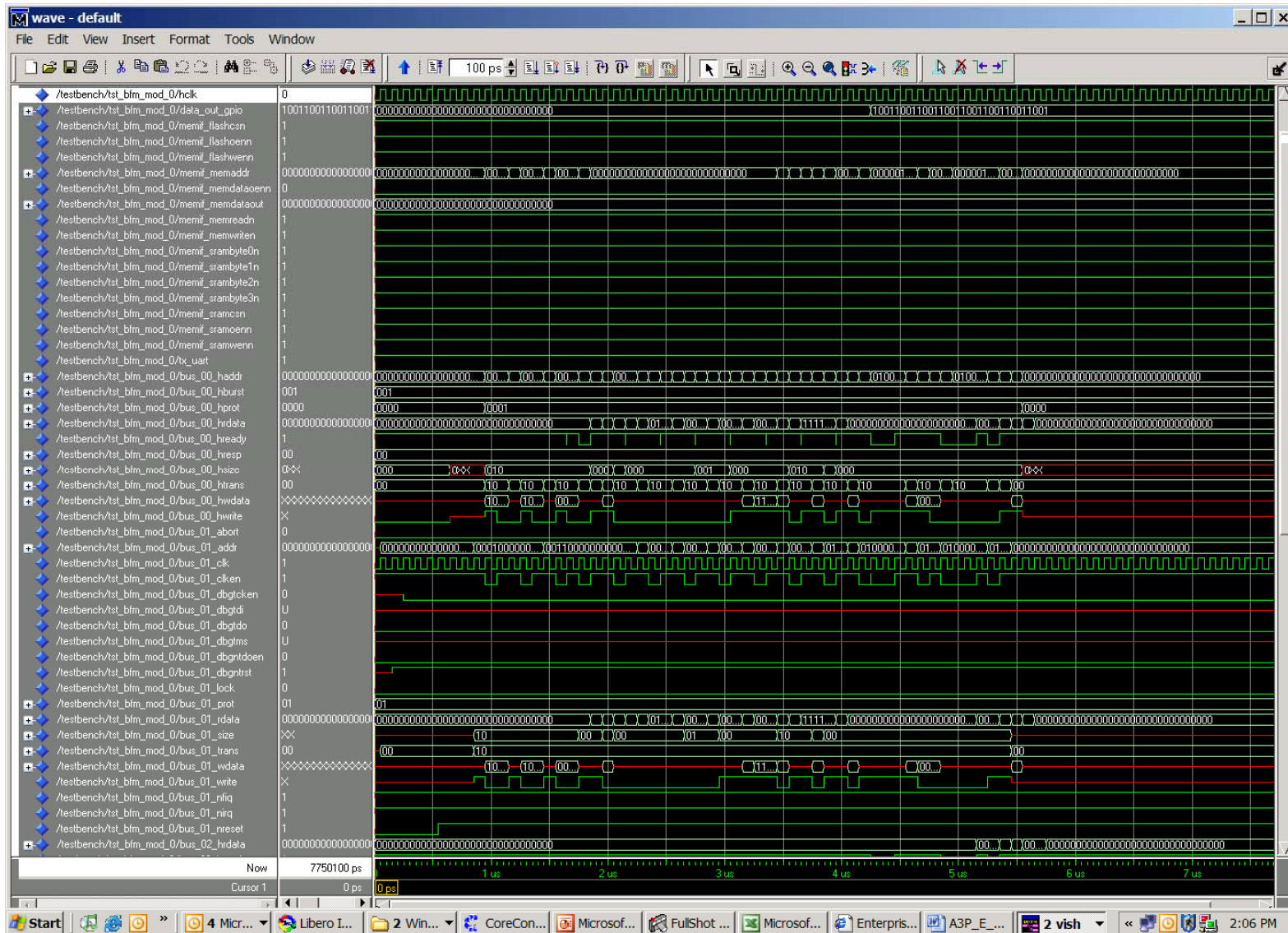
Viewing All Cortex-M1 Signals

Option 2 – Include Wave File



**Add Custom
.do File**

BFM Simulation System



BFM Summary

- Allows User to Test Integration of Subsystem
 - Tests that All Peripherals Can Be Accessed
 - Tests Address Decoding and Bus Connections
- Includes ARM Cortex-M1 Processor Core
 - AMBA Interface
 - Memory (TCM) Interface
- Includes Timing Shell
 - Facilitates Back-annotation of Delay Information during Post-synthesis Simulation



Cortex-M1 Development Tools

Cortex-M1 Ecosystem Support

■ Actel Tools

- Libero Integrated Design Environment
- SoftConsole Program Development Environment



■ Compilers and Debuggers

- ARM RealView Developer Suite
 - C Compiler, Debugger, Instruction Set Simulator
- Keil –Compiler, Debugger
- IAR – Compiler, Debugger
- CodeSourcery - GNU/GDB

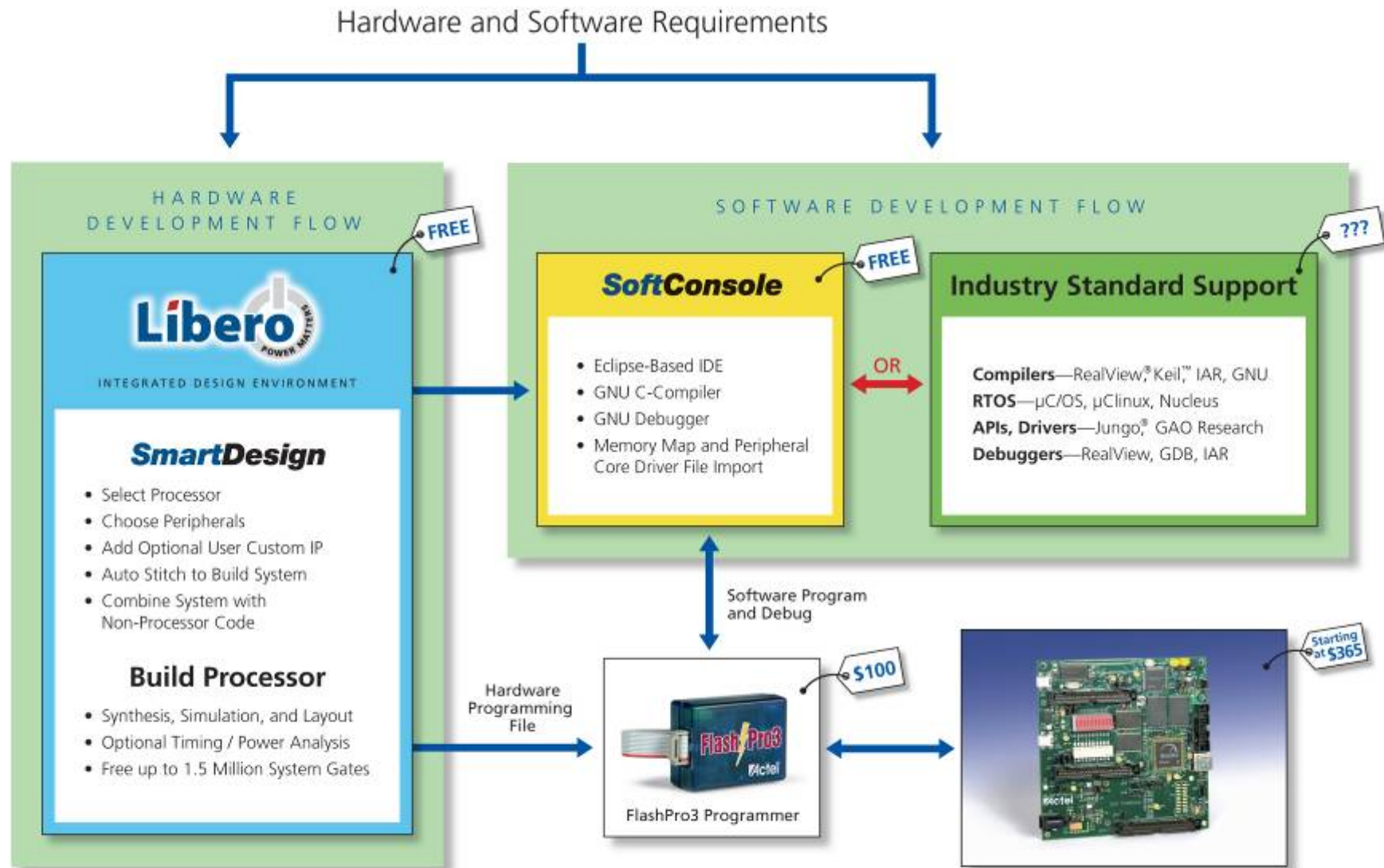


■ RTOS

- μ C/OS-II - Micrium

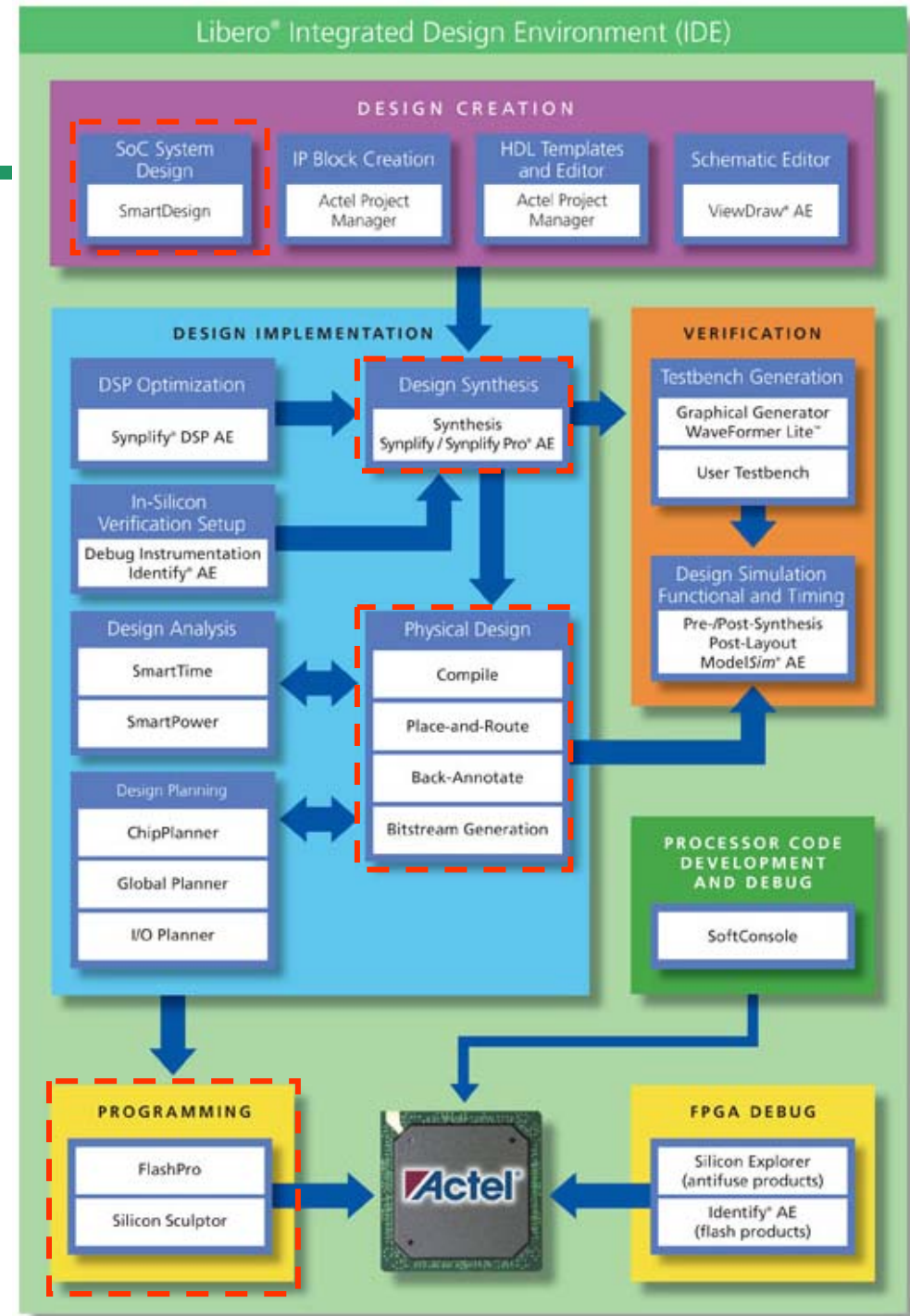


Actel Processor Design Flow

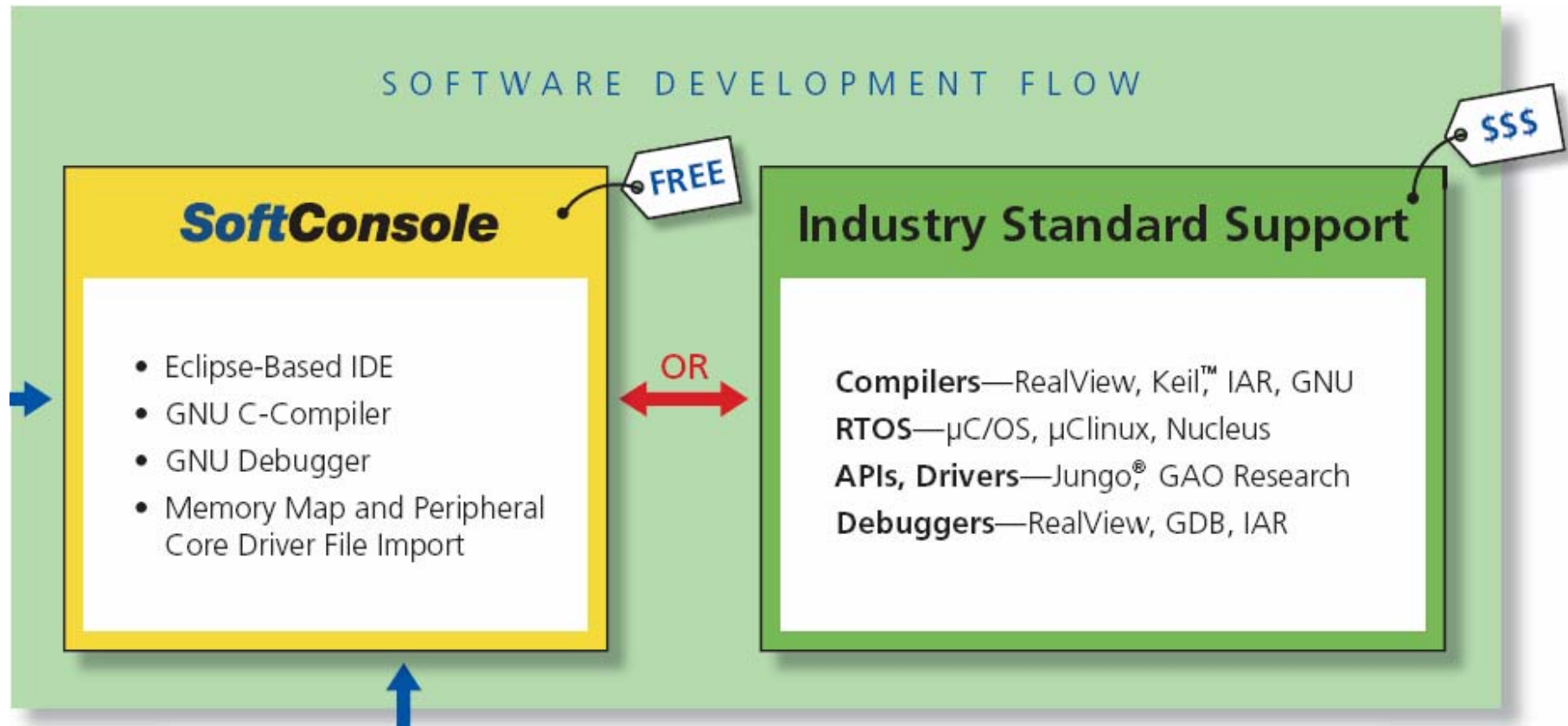


Hardware Design

- Design Creation
 - SmartDesign
 - Free Processor IP
 - Add FREE Peripherals
 - Add non-Processor IP
- Design Implementation
 - Synplify Synthesis
 - Modelsim Simulation
 - Designer FPGA Layout
- Design Analysis
 - Timing Analysis
 - Power Analysis



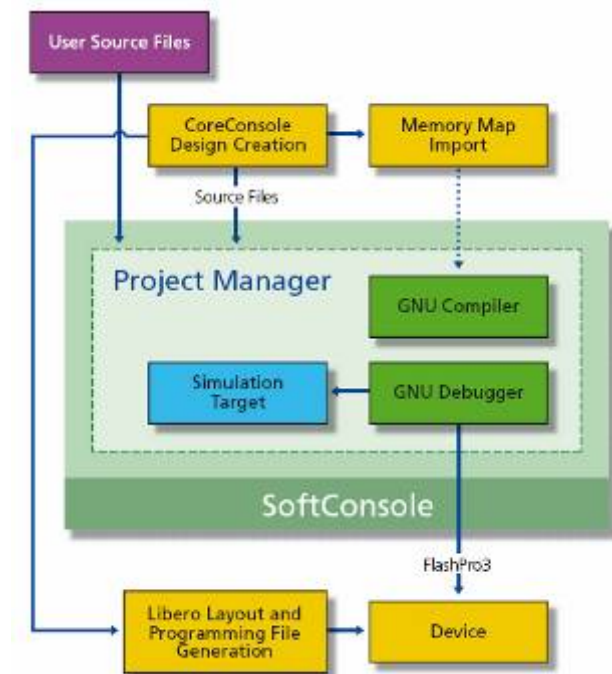
Software Development



SoftConsole v2.2

Processor SW Development

- FREE Software development environment
 - Eclipse-based IDE - easy user interface
 - Supports Cortex-M1, CoreMP7, Core8051/s
 - Can be downloaded from www.actel.com
- C/C++ programming and debug
 - CodeSourcery G++ ARM tools
 - SDCC 8051 compiler
 - Programming and debug with Actel's FlashPro3
- Can import existing code
 - Open platform for application development
- Support for RTOS and stacks
 - uC/OS-II
 - TCP/IP, USB, IPMI



SoftConsole

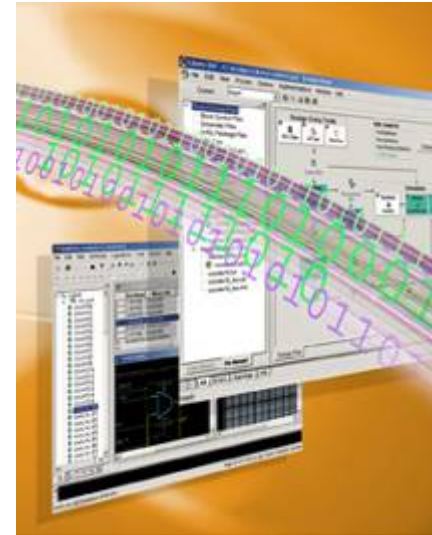
SoftConsole Features

■ Tools

- C/C++ Programming
- Debugging
- Disassembly
- Signals
- Evaluation of Expressions at Runtime

■ Intelligent Software Analysis

- Citing Points of Declaration and Definition
- Source Code Outlining
- Syntax Highlighting and Comment Toggling
- Code History for Tracking Changes
- Code Assist Mode
 - Source Code Completion to Avoid Syntax Errors
 - Code Templates to Auto-insert Standard Code
 - Examples – `switch` Statement and `try/catch` Block



SoftConsole Eclipse IDE

- Provides Open Platform for Application Development Tools
 - Implemented in Java for Easy OS Migration
- Tight Integration of Tools
 - CodeSourcery Sourcery G++ ARM Tools
 - FS2 FlashPro3 In-Target System Analyzer
- Simple User Interface
 - Supports All Levels of Development
 - Coding, Debugging, Disassembly, Edit & Recompile
 - Builds Applications with Minimal User Effort
- Existing Code Can Be Imported and Built in New SoftConsole Project

SoftConsole GNU C/C++ Compiler

- Extensive Intelligent ARM Optimization
 - Built from CodeSourcery G++ GNU/GDB
- Includes Many Features Useful for Embedded Systems
 - Powerful inline assembly syntax
 - Comprehensive linker script language permitting exact placement of code and data
- Large Developer Base Results in Tool Stability
- ISO C and C++ Language Support
 - Complete runtime libraries
 - Aggressive code usage analysis and syntax warnings
 - Supports ARM EABI for better portability

SoftConsole GDB Debugger

- Support for Source- and Assembly-level Debugging
- Live Debugging of New Code
 - In an FPGA or in the GDB ARM simulator
- Breakpoints Can Occur When Conditions Are Met
- Intelligent Access to Hardware
 - Register banks and memory ranges
 - Hover over a variable to read its current value
 - Current stack frame displayed while debugging
- Evaluation of Expressions at Runtime

On-Chip Debugging via FlashPro3

- Download and Debug Executable Programs to Cortex-M1 Development Kits using FlashPro3
 - No RealView ICE – Micro Edition (RVI-ME) required
 - Reduces Pin Count
 - Utilizes Dedicated FPGA JTAG Pins via UJTAG versus GPIO RVI-ME Configuration (10 Pins)
- Full Debugging of Code on Remote Target
 - View Internal Registers, Memory Locations, Variables, etc.
- Uses Same Interface as Instruction Set Simulator
 - Only One Tool to Learn



SoftConsole Requirements

■ Licensing

- SoftConsole Is Open Source
- Free download from Actel's web site
- Individual Licenses for SoftConsole Elements Are Presented in Installation Agreement

■ Supported Platforms

- Microsoft Windows – US Version
 - Windows 2000 with SP4
 - Windows XP SP2 (Professional Version Only)

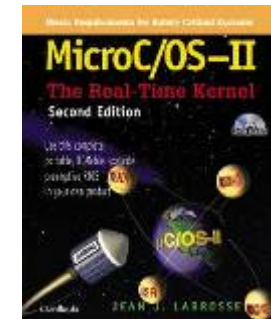
■ System Requirements

- 1.0 GHz Pentium-class Processor
- 400MB Available Disk Space
- 128 MB System RAM
- 1024x768 Video Resolution



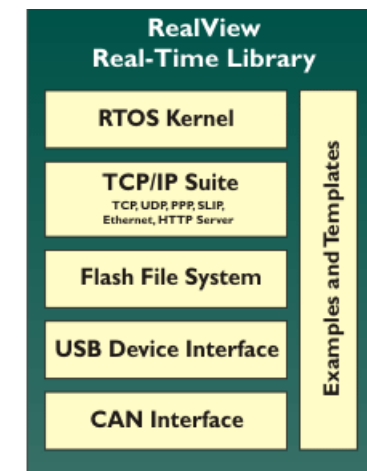
Micrium μ C/OS-II

- Cortex-M1 port can be downloaded for FREE
 - Open source model
 - To use for evaluation and development
 - License (\$4K) required to use in an application that is sold
 - License is Royalty-Free
- #2 Commercial RTOS
 - Used in 1000s of products all over the world
 - Real-time kernel
 - Scalable and ROMable
 - 5K to 20K bytes code, 1K to 3K bytes data (Cortex-M1)
 - Provides standard services
 - Semaphores, message mailbox, queues, task & time management
- Available Micrium Stacks
 - uC/TCP-IP (Ethernet)
 - uC-FS (File System for embedded applications)
 - uC/GUI (efficient GUI for any application with an LCD)
 - uC/USB (USB device or host controller)
 - uC/CAN (CAN protocol framework)
 - uC/FL (Flash loader for embedded update via PC)



RealView MDK (Keil)

- Now with full Support for Actel's Cortex-M1
 - Version 3.23a or later
 - Free eval download available
 - Full version costs \$4K
 - Features
 - [µVision](#) IDE, debugger and simulation environment
 - [RealView](#) industry-leading C/C++ compiler from ARM
 - [MicroLib](#) highly optimized run-time library
 - [Real-Time Trace](#) for Cortex-M3 processor based devices
 - Keil [RTX](#) – deterministic Real-Time Operating System
 - Keil has written an app note on using the MDK with Actel's M1A3P SOC board
 - Includes several examples
 - Requires Keil's uLink probe for download and debug
 - Plugs into 20-pin header on Actel boards



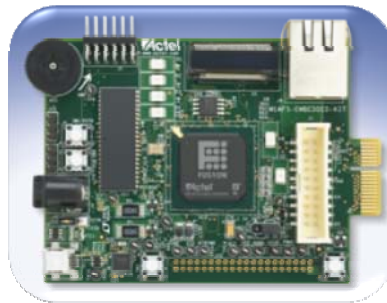
ARM Cortex-M1 Development Kits

Kit Name	Status	Ordering Code	Device Supported	Price
ARM Cortex-M1 IGLOO Development Kit	Production	M1AGL1000-DEV-KIT	M1AGL1000V2-FGG484	\$340
ARM Cortex-M1 ProASIC3L Development Kit*	First Article	M1A3PL-DEV-KIT	M1A3P1000L-FGG484	\$370
Fusion Embedded Development Kit	Production	M1AFS-EMBEDDED-KIT	M1AFS1500-FGG484	\$199
Fusion Advanced Development Kit *	First Article	M1AFS-ADV-DEV-KIT	M1AFS1500-FGG484	\$650

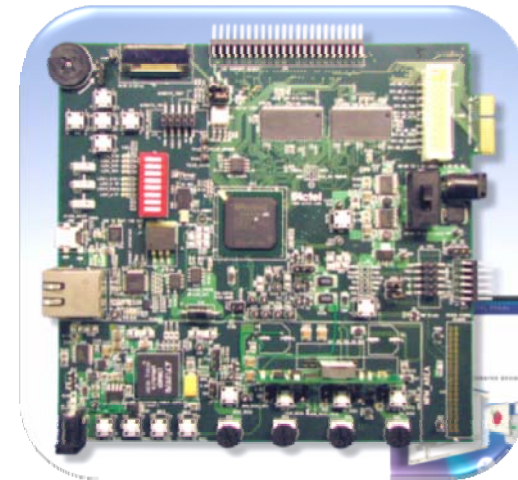
*First Customer Ship on track for April 09



M1AGL and M1A3PL
Development Kit



Fusion Embedded Development Kit



Fusion Advanced Development Kit

Fusion Embedded Development Kit

- Develop designs using
 - Fusion Mixed-Signal FPGA
 - M1AFS1500-FGG484
 - Cortex-M1 embedded processor
 - 8051s embedded processor
 - Ethernet applications

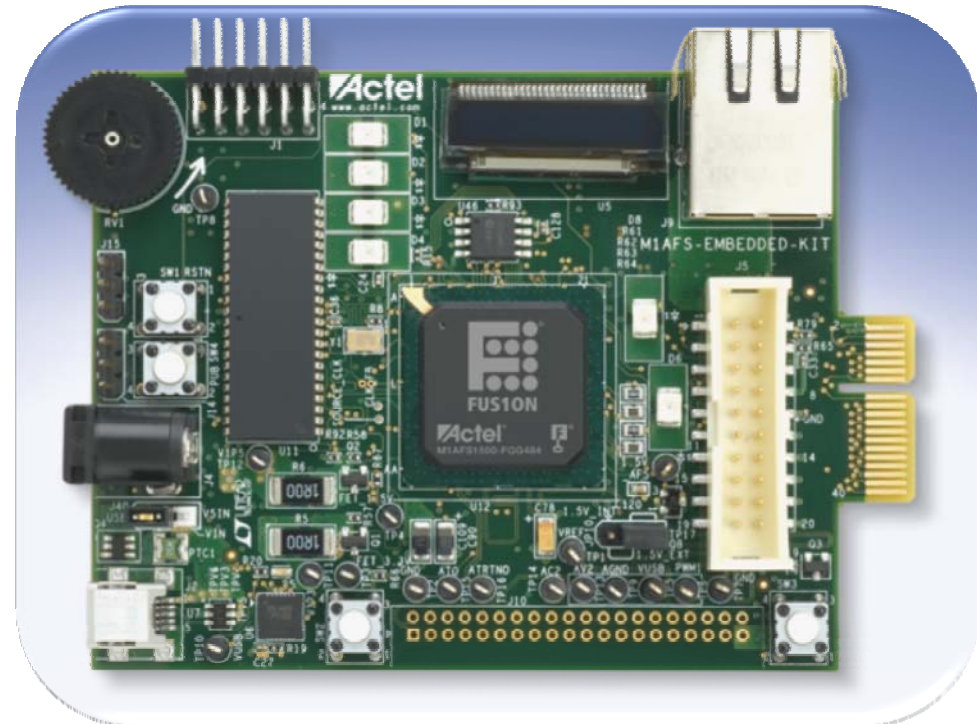
- The development kit includes
 - Low Cost Programming Stick
 - Libero IDE
 - Free Libero IDE Gold license
 - SoftConsole for Compile/Debug
 - On Chip Program and Debug
 - User's guide and tutorial
 - Example designs
 - PCB schematics and layout files

- Ordering code
 - M1AFS-EMBEDDED-KIT @ \$199



Fusion Embedded Development Board Features

- RoHS compliant
- 10/100 Ethernet interface
- USB-to-UART interface
- I2C interface
- Built-in temperature monitor
- Voltage potentiometer
- RealView debug header
- OLED 96x16 pixel display
- 4,000,000 SRAM



Cortex-M1 Summary

- Cortex-M1 Allows Designers to Benefit from Hassle-free, Industry-standard ARM Architecture
 - Optimized for Use in M1 Devices (ProASIC3/E, IGLOO/e & Fusion)
- Actel FPGA Tools Offer Seamless Development Flow
 - Libero, SmartDesign, Dev Kit Hardware Development Tools
 - SoftConsole with GNU Software Development Tools
 - RealView and Third-party Ecosystem Support
- Brings Flexibility and Fast Time to Market to System-level Designs

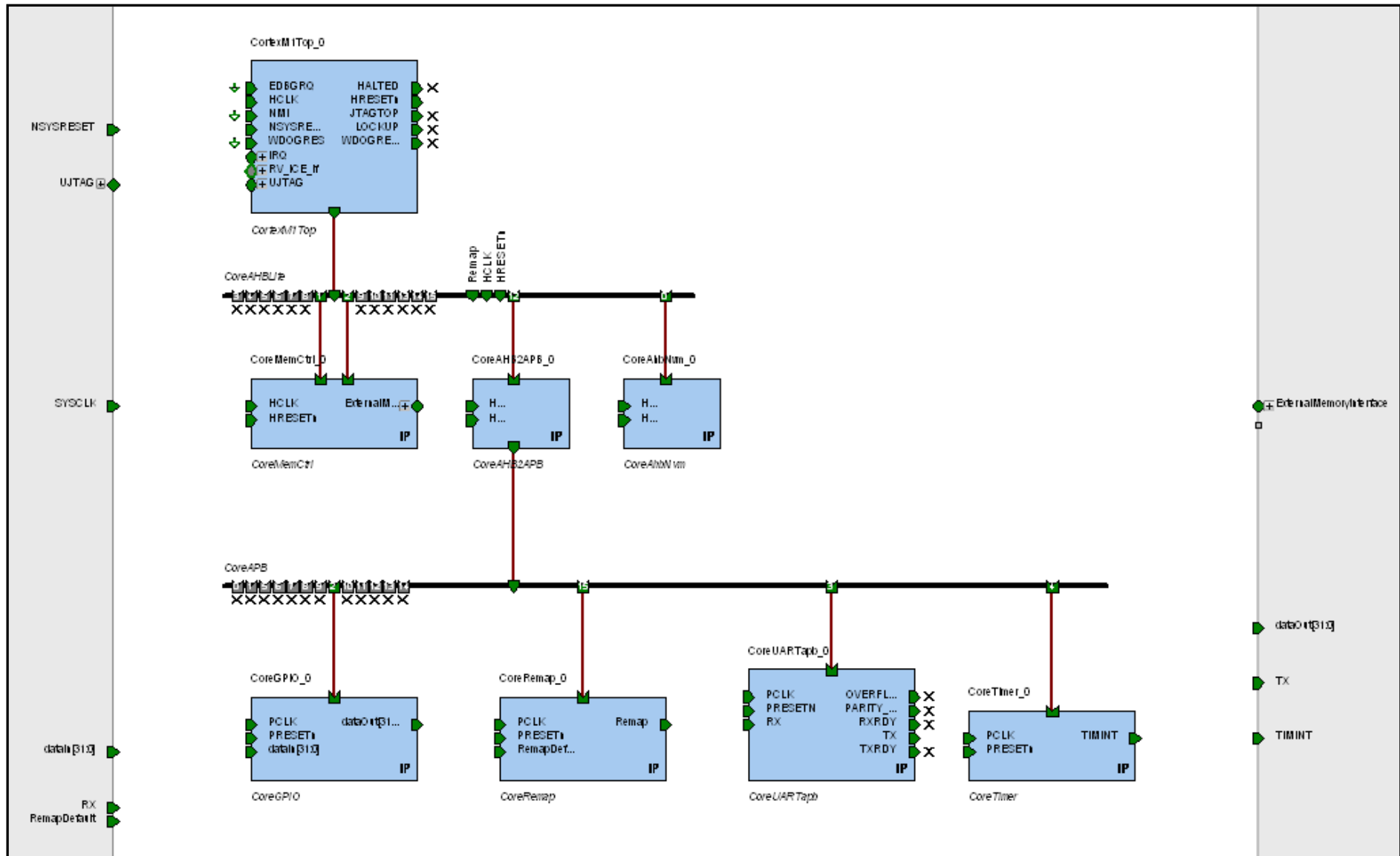
Reference

Lab Simulation Results



Lab Simulation Results

Cortex-M1 Subsystem



Cortex-M1 Subsystem

Memory Map

CortexM1Top_0 Subsystem

Master(s) on this bus:

- CortexM1Top_0

	Base Address	
	NoRemap	SwapSlots0and1
CoreAhbNvm_0 : NVM	0x00000000	0x10000000
CoreMemCtrl_0 : RAM	0x10000000	0x00000000
CoreMemCtrl_0 : FLASH	0x20000000	
CoreGPIO_0 : RegisterMap	0xc2000000	
CoreUARTapb_0 :	0xc3000000	
CoreTimer_0 : RegisterMap	0xc4000000	
CoreRemap_0 : RegisterMap	0xcf000000	

[subsystem list](#), [top of page](#)

CoreTimer

Register Memory Map

CoreTimer_0 : RegisterMap Memory Map

range: 0x01000000

Address	Type	Width	Reset Value	Name	Description
base address + 0x00	read-write	32	0x00000000	TimerLoad	
base address + 0x04	read-only	32	0xFFFFFFFF	TimerValue	
base address + 0x08	read-write	32	0x00	TimerControl	
base address + 0x0C	write-only	32	0x0	TimerPrescale	
base address + 0x10	write-only	32	0x0	TimerIntClr	
base address + 0x14	read-only	32	0x0	TimerRIS	
base address + 0x18	read-only	32	0x0	TimerMIS	

[back to CortexM1Top_0 Memory Map](#)

subsystem.bfm

Explanation

```
#-----  
# Memory Map  
# Define name and base address of each resource.  
#-----
```

```
memmap CoreMemCtrl_00 0x00000000;  
memmap CoreMemCtrl_00 0x10000000;  
memmap CoreGPIO_00 0xc2000000;  
memmap CoreTimer_00 0xc4000000;  
memmap CoreRemap_00 0xcf000000;
```

System memory map –
Syntax:
resource_name base_address

```
#-----  
# Include resource scriptlets  
#-----
```

```
#include CoreMemCtrl CoreMemCtrl_00;  
#include CoreMemCtrl CoreMemCtrl_00;  
#include CoreGPIO CoreGPIO_00;  
include CoreTimer_lab CoreTimer_00;  
#include CoreRemap CoreRemap_00;
```

Specifies which scriptlet to run and base address for peripheral (see above for address)
Here only CoreTimer_lab_scriptlet.bfm will be executed; all others are commented out
Note that “_scriptlet.bfm” is not required

Execute CoreTimer_lab_scriptlet.bfm with base address 0xc4000000 for CoreTimer

CoreTimer_lab_scriptlet.bfm

Explanation

```
write W VAR_resource 0x00 0x00000100;           #write 0x00000100 to TimerLoad register
                                                  #load a starting count in the timer
readcheck W VAR_resource 0x00 0x00000100;       #read TimerLoad register; expect 0x00000100
readcheck W VAR_resource 0x04 0x00000100;       #read TimerValue register; expect 0x00000100
write h VAR_resource 0x0C 0x0001;               #write 0x0001 to TimerPrescale register
                                                  #set counter clock rate (PCLK ÷ 4)
readcheck h VAR_resource 0x0C 0x0001;           #read TimerPrescale register; expect 0x0001
write h VAR_resource 0x08 0x0003;               #write 0x0003 to TimerControl register
readcheck h VAR_resource 0x08 0x0003;           #read TimerControl register; expect 0x0003
poll b VAR_resource 0x014 0x01;                 #read TimerRIS register; wait until bit 0 = 1
read h VAR_resource 0x04;                       #read TimerValue register
write h VAR_resource 0x10 0x1234;               #write TimerIntClr register to clear
interrupt

poll b VAR_resource 0x014 0x01;                 #any write value clears the interrupt
read h VAR_resource 0x04;                       #read TimerRIS register; wait until bit 0 = 1
write h VAR_resource 0x10 0x1234;               #read TimerValue register
interrupt                                         #write TimerIntClr register to clear

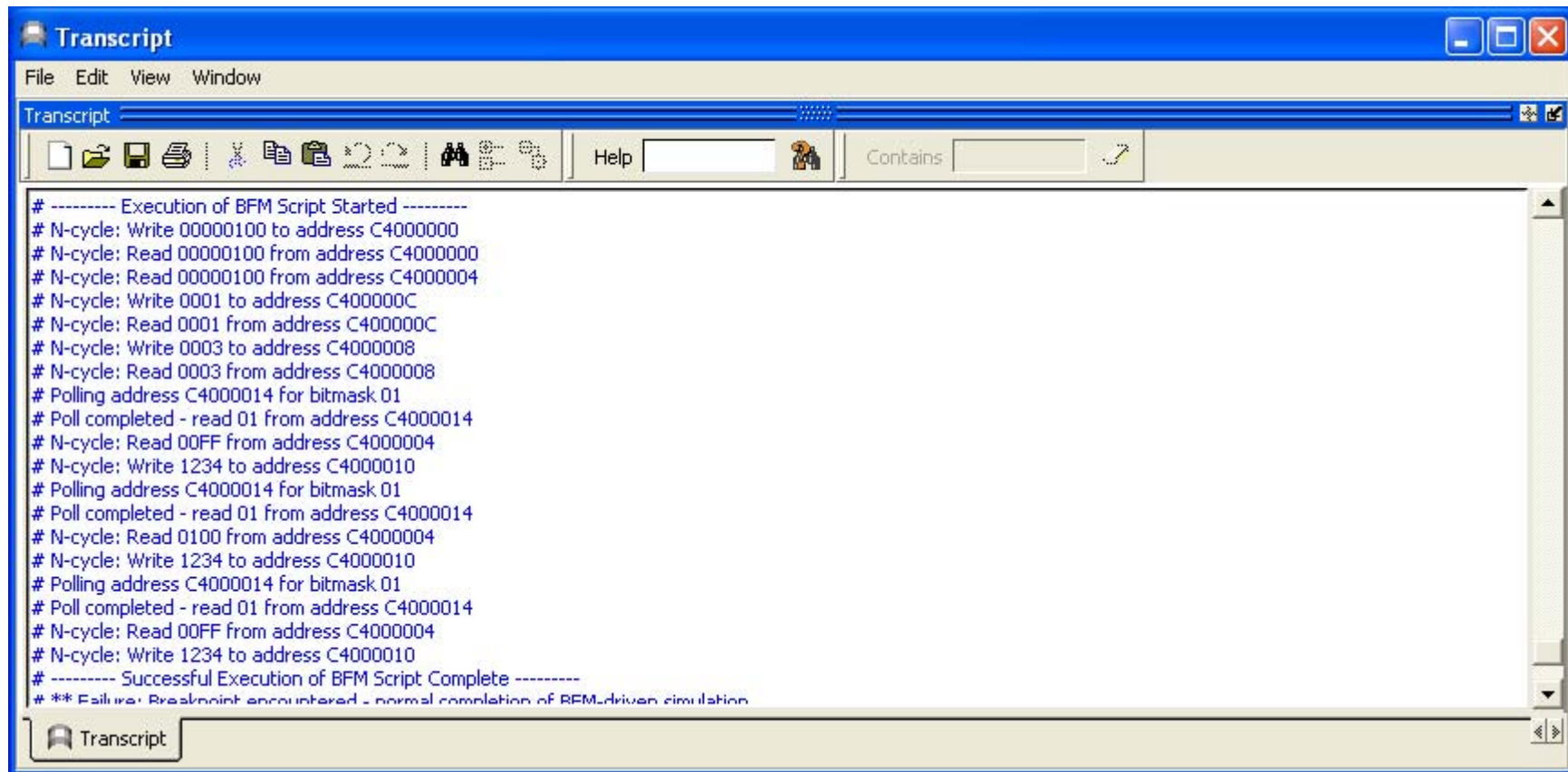
poll b VAR_resource 0x014 0x01;                 #poll TimerRIS register; wait until bit 0 = 1
read h VAR_resource 0x04;                       #read TimerValue register
write h VAR_resource 0x10 0x1234;               #write TimerIntClr register to clear
interrupt
```

Syntax:

```
write      width  resource_name  byte_offset  data;
read       width  resource_name  byte_offset;
readcheck  width  resource_name  byte_offset  data;
```

BFM Simulation

ModelSim Transcript Window

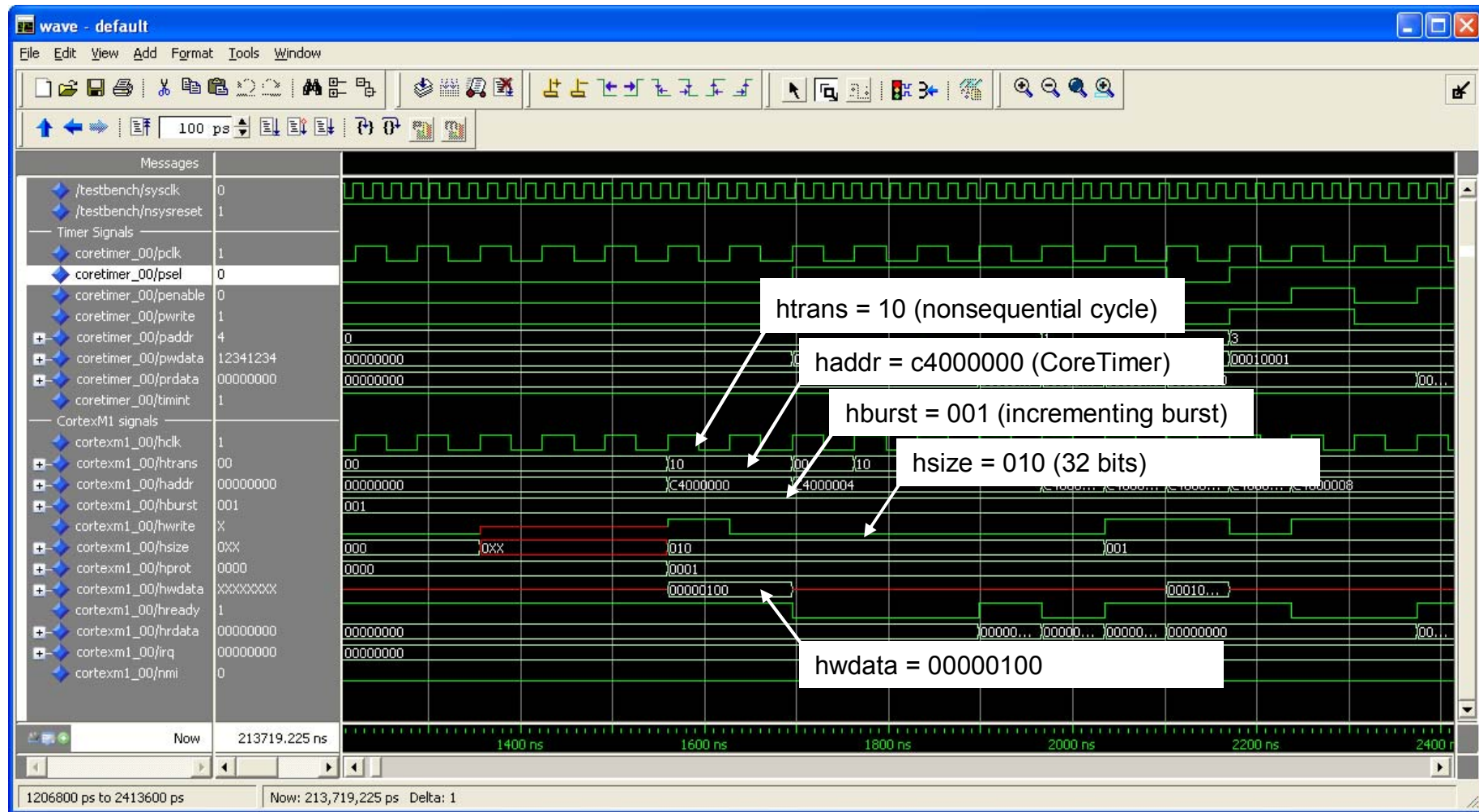


The screenshot shows a window titled "Transcript" with a menu bar (File, Edit, View, Window) and a toolbar. The main area contains the following text:

```
# ----- Execution of BFM Script Started -----  
# N-cycle: Write 00000100 to address C4000000  
# N-cycle: Read 00000100 from address C4000000  
# N-cycle: Read 00000100 from address C4000004  
# N-cycle: Write 0001 to address C400000C  
# N-cycle: Read 0001 from address C400000C  
# N-cycle: Write 0003 to address C4000008  
# N-cycle: Read 0003 from address C4000008  
# Polling address C4000014 for bitmask 01  
# Poll completed - read 01 from address C4000014  
# N-cycle: Read 00FF from address C4000004  
# N-cycle: Write 1234 to address C4000010  
# Polling address C4000014 for bitmask 01  
# Poll completed - read 01 from address C4000014  
# N-cycle: Read 0100 from address C4000004  
# N-cycle: Write 1234 to address C4000010  
# Polling address C4000014 for bitmask 01  
# Poll completed - read 01 from address C4000014  
# N-cycle: Read 00FF from address C4000004  
# N-cycle: Write 1234 to address C4000010  
# ----- Successful Execution of BFM Script Complete -----  
# ** Failure: Breakpoint encountered - normal completion of BFM-driven simulation
```

BFM Simulation

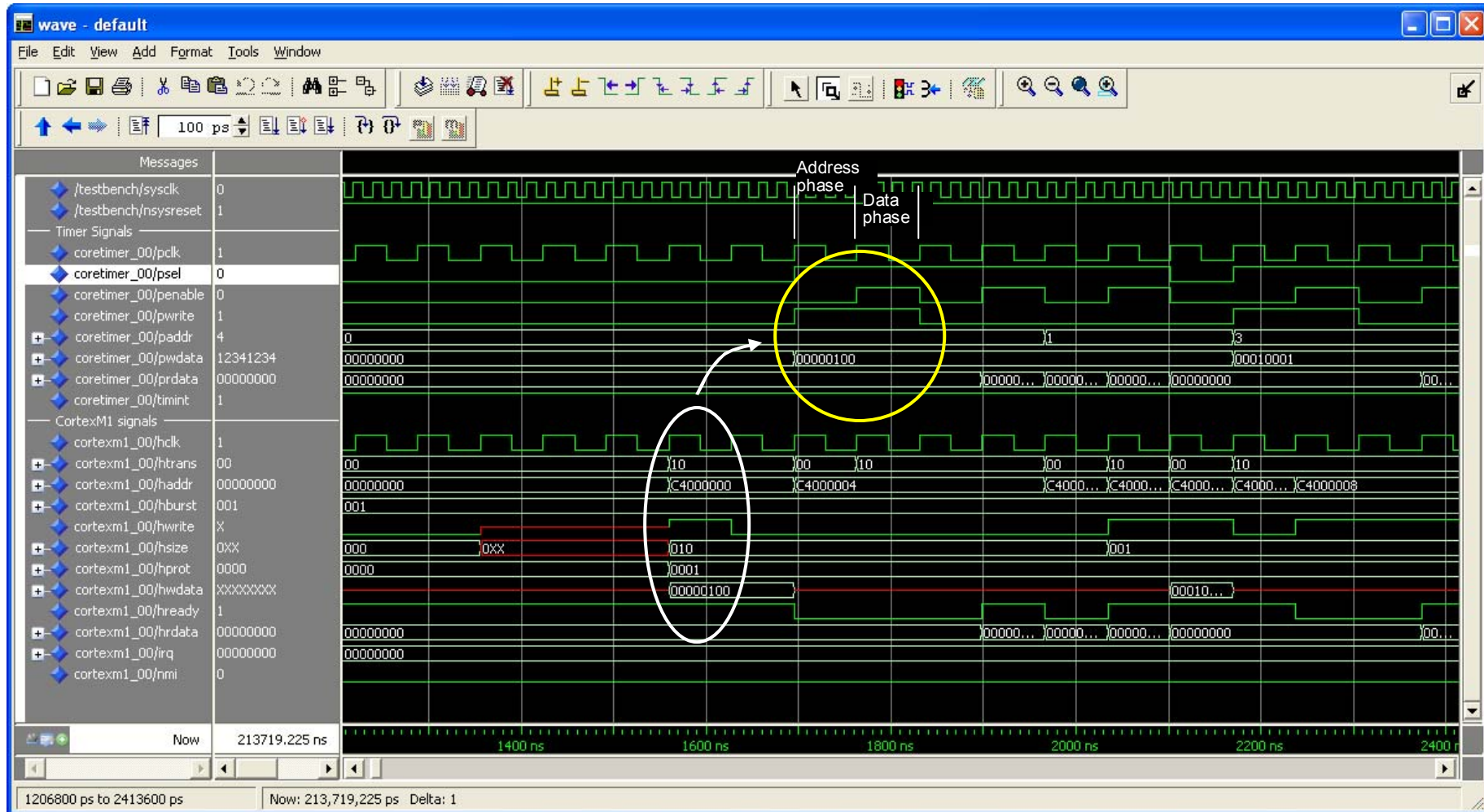
ModelSim Wave Window



AHB write cycle – write 0x00000100 to CoreTimer TimerLoad register (0xc400000)
 Initiated by readcheck W VAR_resource 0x00 0x00000100; in CoreTimer_lab_scriptlet.bfm

BFM Simulation

ModelSim Wave Window

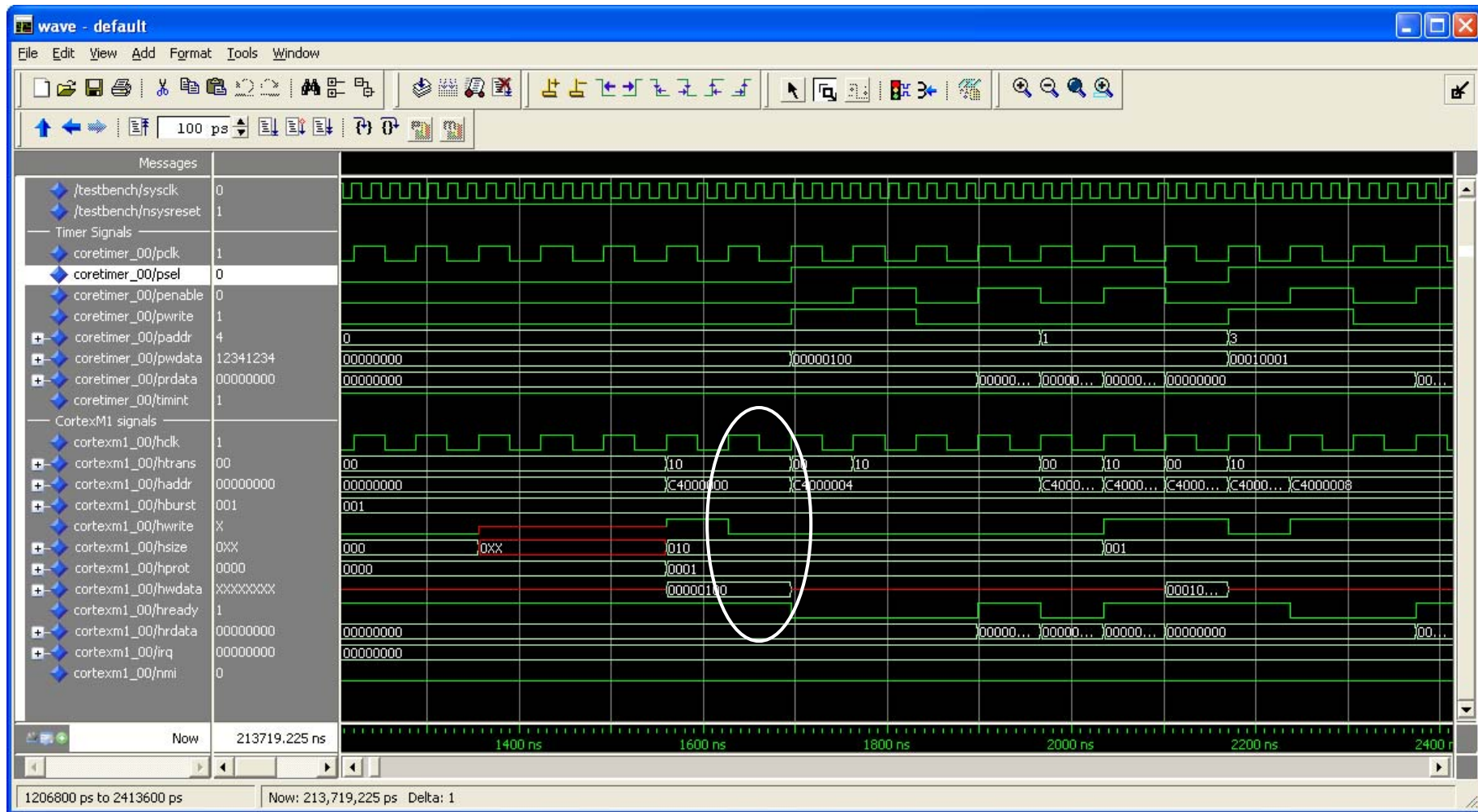


APB write cycle (yellow) – write 0x00000100 to CoreTimer TimerLoad register

Note that paddr = 0x000 – CoreAHB2APB strips off the upper 8 address bits and drives the appropriate PSEL

BFM Simulation

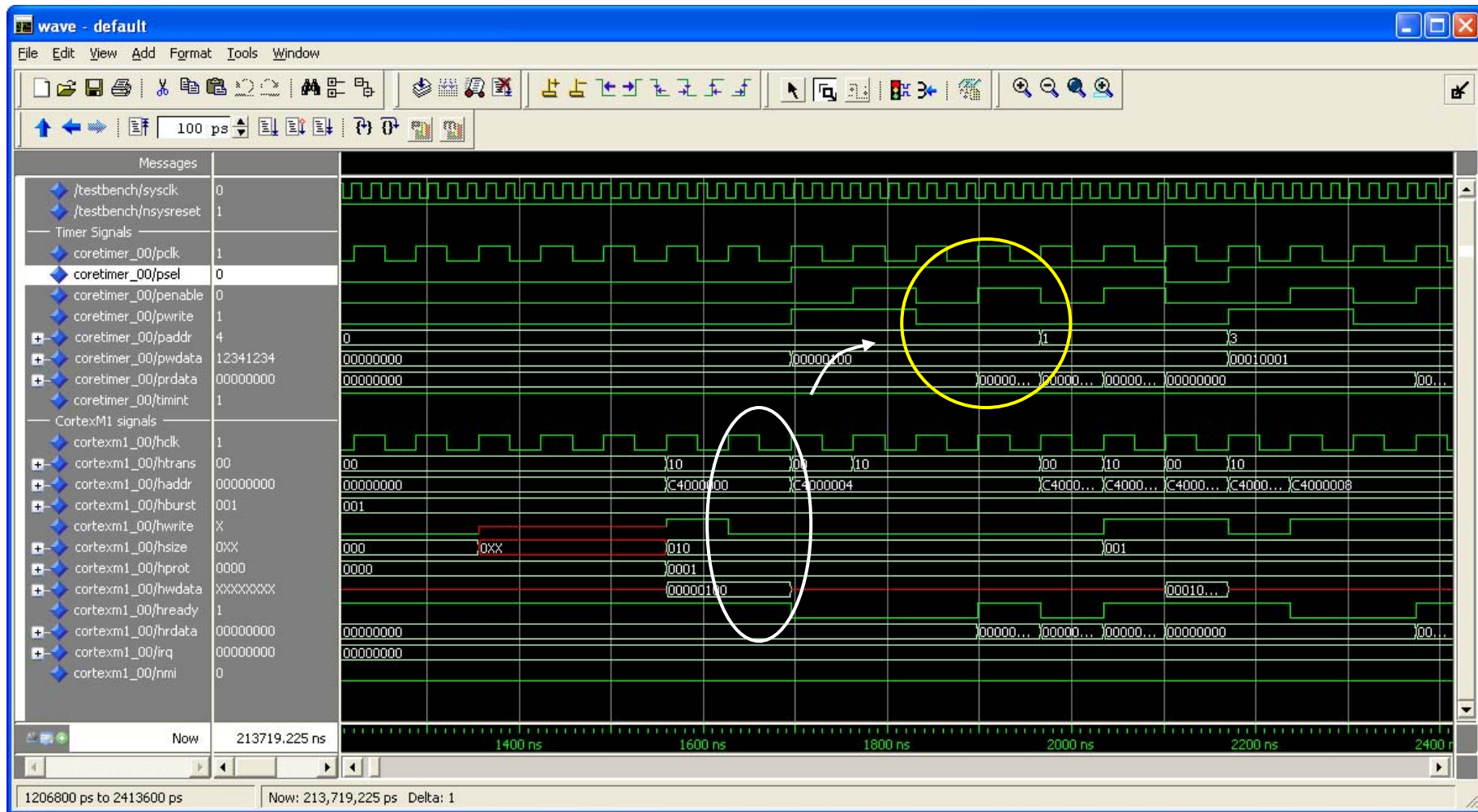
ModelSim Wave Window



AHB read cycle – read CoreTimer TimerLoad register
 Initiated by write W VAR_resource 0x00 0x00000100; in CoreTimer_lab_scriptlet.bfm

BFM Simulation

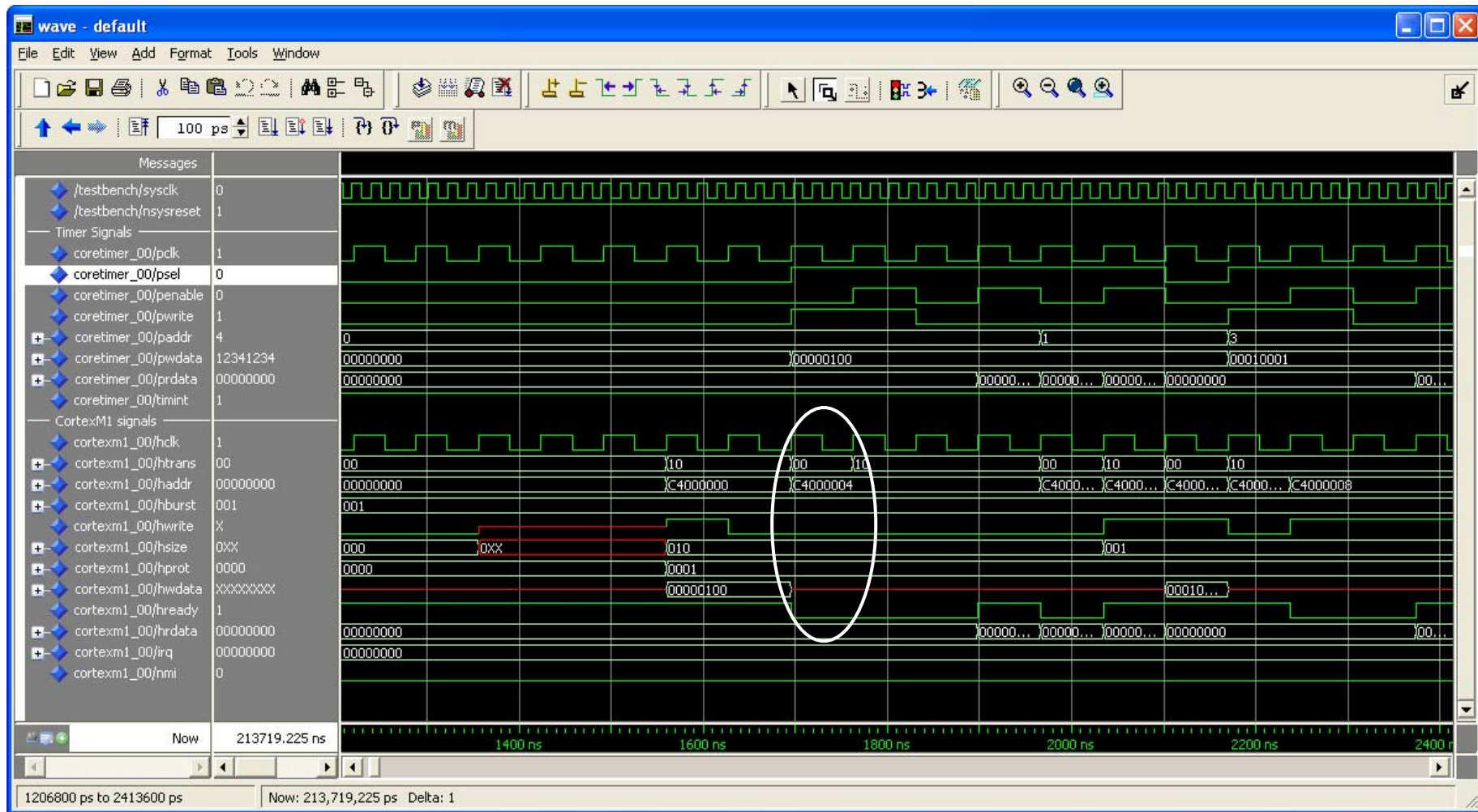
ModelSim Wave Window



APB read cycle (yellow) – write 0x00000100 to CoreTimer TimerLoad register
Note that paddr = 0; prdata = 0x00000100

BFM Simulation

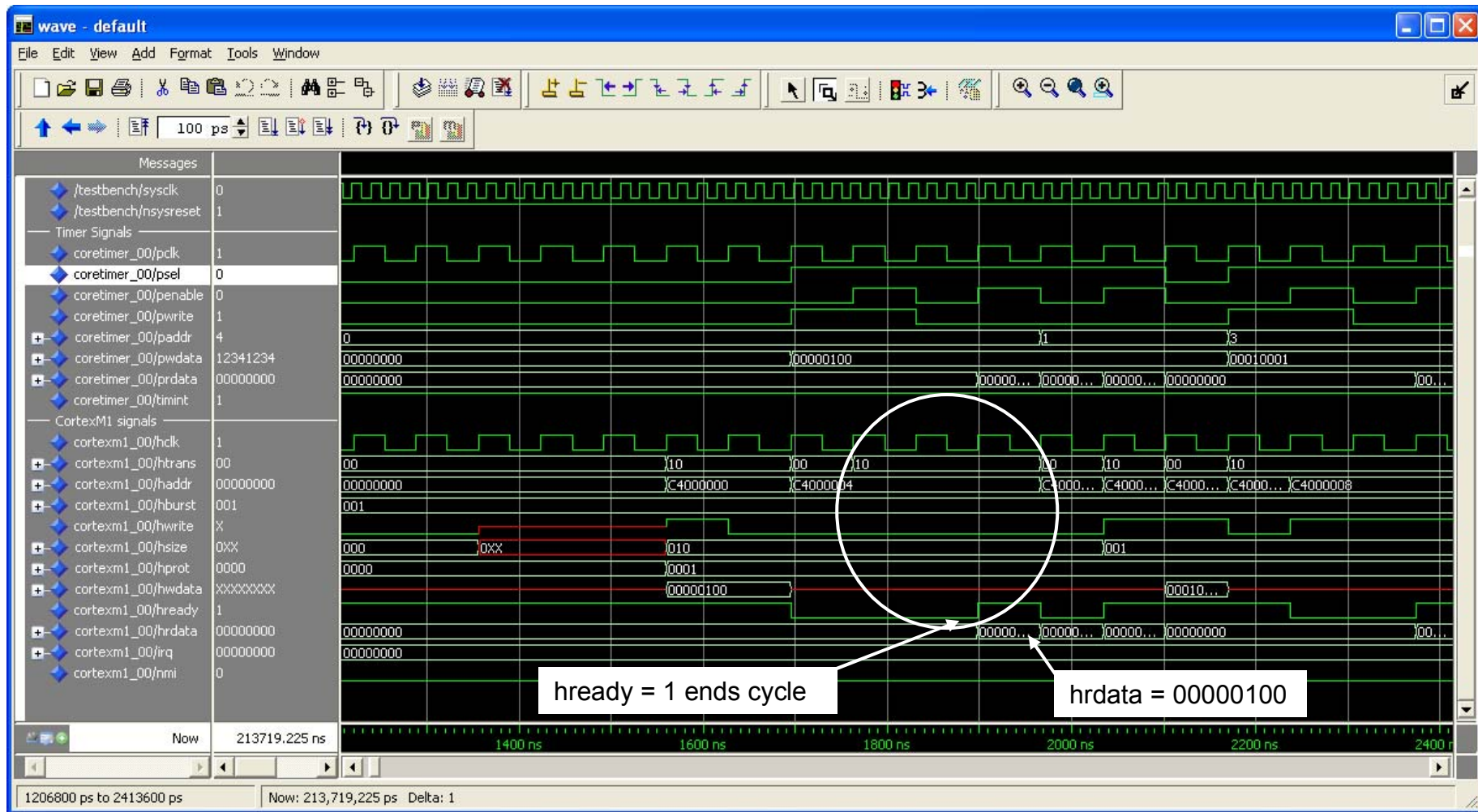
ModelSim Wave Window



AHB idle cycle

BFM Simulation

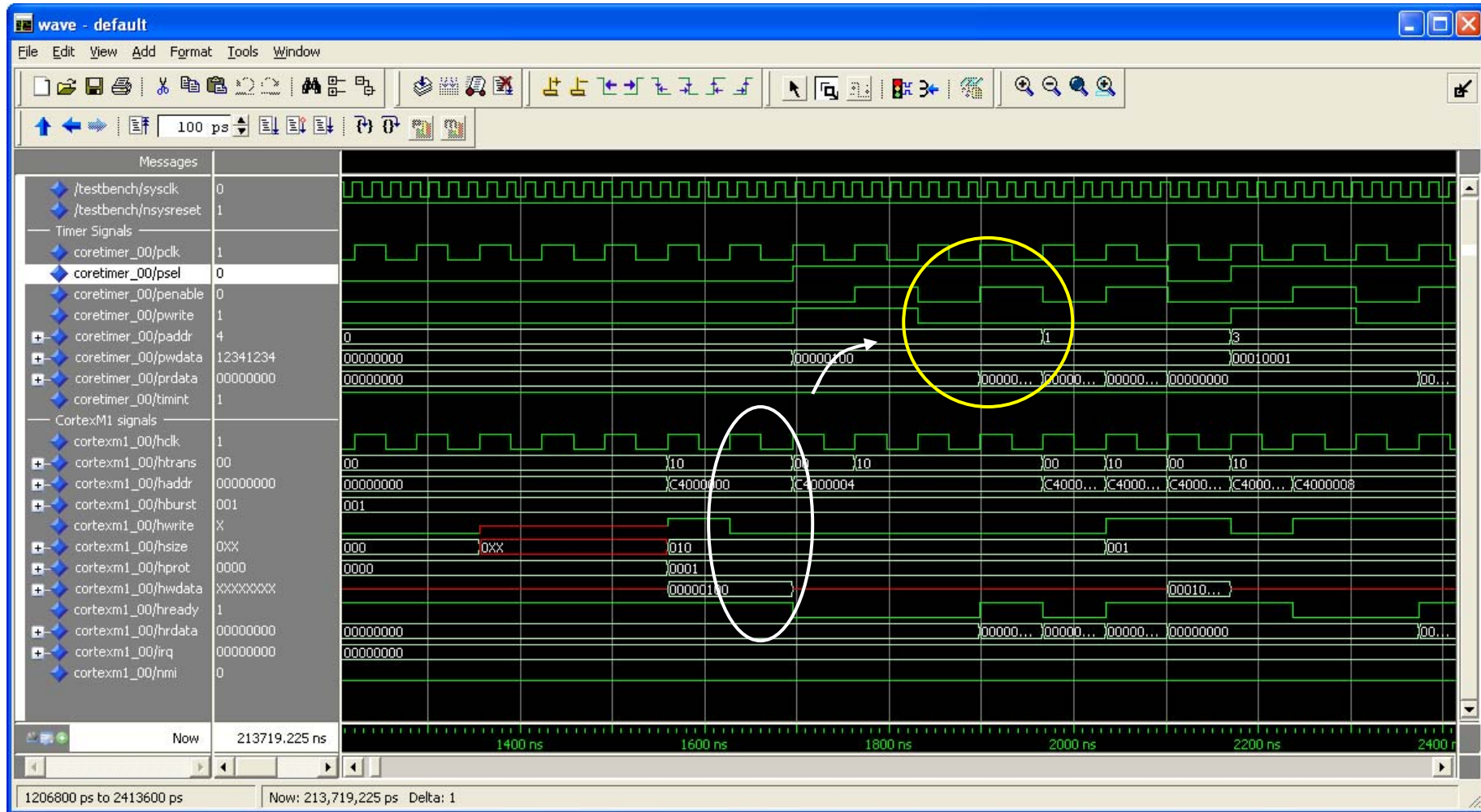
ModelSim Wave Window



AHB read cycle – read CoreTimer TimerValue register
 Initiated by write W VAR_resource 0x04 0x00000100; in CoreTimer_lab_scriptlet.bfm

BFM Simulation

ModelSim Wave Window



APB read cycle (yellow) – write 0x00000100 to CoreTimer TimerLoad register
 Note that paddr = 4; prdata = 0x00000100