Rate This Document

Core8051s Embedded Processor Hardware Development Tutorial

for Fusion Mixed-Signal FPGAs

Actel®
POWER MATTERS

Actel Corporation, Mountain View, CA 94043

© 2009 Actel Corporation. All rights reserved.

Printed in the United States of America

Part Number: 50200155-1

Release: March 2009

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Actel.

Actel makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability or fitness for a particular purpose. Information in this document is subject to change without notice. Actel assumes no responsibility for any errors that may appear in this document.

This document contains confidential proprietary information that is not to be disclosed to any unauthorized person without prior written consent of Actel Corporation.

Trademarks

Actel and the Actel logo are registered trademarks of Actel Corporation.

Adobe and Acrobat Reader are registered trademarks of Adobe Systems, Inc.

All other products or brand names mentioned are trademarks or registered trademarks of their respective holders.

Table of Contents

Introduction

This tutorial shows how to develop a simple 8051-based embedded processor system using Actel design tools. This design is suitable as a starting point for developing an embedded system.

It is assumed that the reader is familiar with the FPGA design flow using Actel tools. Actel provides tutorials for both Libero® Integrated Design Environment (IDE) and SmartDesign on the Actel website in addition to training classes.

After completing this tutorial you will be familiar with the hardware design flow for creating an 8051-based embedded system using SmartDesign. This includes the following steps:

- Instantiating and configuring the processor, memory, and peripherals
- Connecting peripherals and defining an address map
- Generating RTL and the FPGA programming image ready for software development

This tutorial is designed to support the following three development board designs:

- Cortex™-M1–Enabled Fusion Development Kit (M1AFS-DEV-KIT-SCS) board
- Fusion Embedded Development Kit (M1AFS-EMBEDDED-KIT) board
- Fusion Advanced Development Kit (M1AFS-ADV-DEV-KIT) board

In general, most tutorial steps apply to all three target boards. Steps which are specific to a given target board will be clearly indicated.

Requirements for Tutorial

This tutorial requires that the Actel Libero IDE is installed. Libero IDE allows you to create the design, perform simulation, and prepare a file for programming of the FPGA fabric.

This tutorial is based on Libero IDE version 8.5.

This tutorial includes a set of files needed for completing and programming this design. These files can be found at http://www.actel.com/documents/Core8051s_Fusion_DesignTutorial_DF.zip. Unzip this file in a root directory (e.g., c:/ or d:/) of your hard drive. If you unzip the file to a directory path that is too long, you will be asked for a password and you will not be able to extract the files.

Inside the unzipped folder you will find a *Core8051s_Fusion_FPGA_Design_Tutorial_Files* folder (Figure 1). Inside this folder will be a folder for each of three target boards. Inside the board folder will be a folder which contains a completed design and a folder called *Tutorial_Files*, which contains support files needed for completing this design.

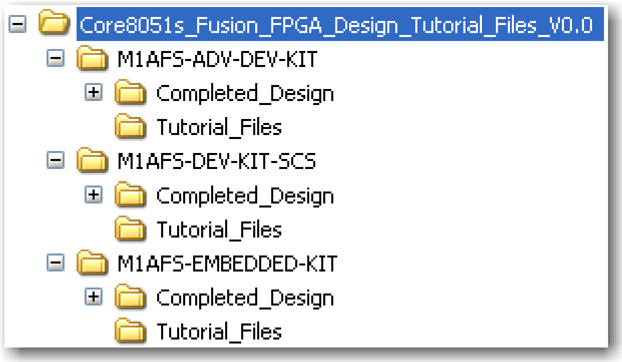


Figure 1 · File Structure

The tutorial folder includes the following files:

- An *.ihx file. This is a hex file with the application code for the NVM used by the microcontroller. This is one of the following, depending on the target board.

  M1AFS_SCS_51S_TUT.ihx (for the M1AFS-DEV-KIT-SCS board)

  M1AFS_EMB_51S_TUT.ihx (for the M1AFS-EMBEDDED-KIT board)

  M1AFS_ADV_51S_TUT.ihx (for the M1AFS-ADV-DEV-KIT board)
- The file **MY_NVM.vhd**. This is a wrapper for the NVM used for code memory. This file is used for all target boards.
- The file **memdecode.vhd**. This is VHDL code for the address decoding and control logic. This file is used for all target boards.
- A *.pdc file containing pin assignments for the board. The exact file name is specific to the target board.

Design Overview

This design targets the Actel Fusion® mixed-signal FPGA. Key features of Fusion are its integrated analog-to-digital converter (ADC) and analog I/O, embedded flash memory and SRAM, and support for advanced I/O standards. The analog inputs can provide voltage, current, and temperature measurements. The analog outputs can provide gate drive for external analog switches. The Fusion flash memory is used to create NVM storage for the application code.

This design implements an 8051-based microcontroller system. In addition to the Core8051s microcontroller core, this system consists of the following peripherals:

- CoreUARTapb
- CoreTimer (two instances)
- CoreWatchdog
- CoreGPIO
- CoreInterrupt
- CoreAI (interface to the Fusion Analog Block)

These peripherals are interfaced using the APB3 bus. The CoreAPB3 bus component provides an APB interface that supports up to 16 APB slaves. There is one APB master, which sends out a PSEL signal to CoreAPB3. This is used by CoreAPB3, along with appropriate bits from the PADDR bus, to decode the appropriate PSELS signal. All 16 APB3 slots occupy 256 memory locations. In Core8051s, the APB3 interface uses the upper 4 KB of the memory address space from 0xF000 through 0xFFFF.

The memory system consists of the following elements:

- 64 KB of NVM code memory. This memory is internal to the FPGA and occupies addresses 0x0000 through 0xFFFF of the code memory space.
- 64 KB of SRAM code memory. This memory is external to the FPGA and occupies addresses 0x0000 through 0xFFFF of the code memory space. This SRAM is intended for use with the debugger for debugging purposes.
- 8 KB of SRAM data memory. This memory is internal to the FPGA; however, it occupies addresses 0x0000 through 0x0FFF of the 8051's external data memory space.
- 52 KB of SRAM data memory. This memory is external to the FPGA and occupies addresses 0x1000 through 0xFEFF of the 8051's external data memory space. The APB3 interface used to connect peripherals to Core8051s occupies the upper 4 KB of external data memory.
- Control and interface logic

Note: Both the M1AFS-DEV-KIT-SCS and M1AFS-ADV-DEV-KIT boards contain flash memory devices external to the FPGA. These memory devices are not used in the design and are biased by the respective designs to be inactive.

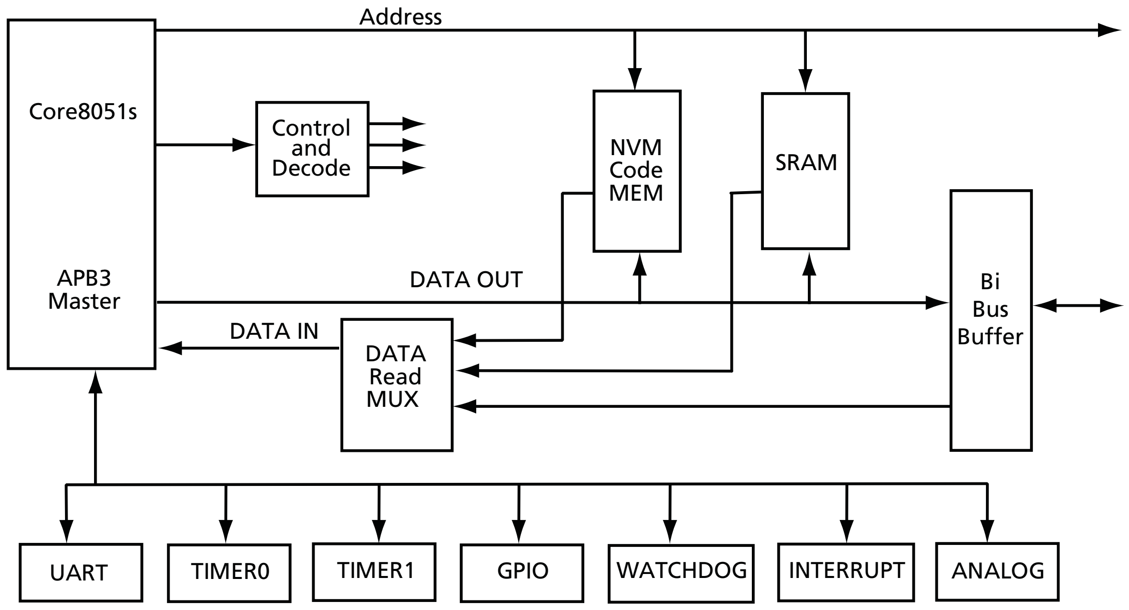


Figure 1-1 · Block Diagram

The software used in the tutorial uses the analog block of the Fusion device to read the voltages and currents of the 3.3 V and 1.5 V supplies. In addition, ambient room temperature is also measured using the analog block and an NPN transistor connected as a temperature sensor.

This hardware design is very flexible and mimics the features set of a standard 8051. In addition, this design adds an exceptionally flexible analog frontend that is not found on most 8051 family devices.

Building the Design

It is assumed that the reader is familiar with the FPGA design flow using Actel tools. Actel provides tutorials for both Libero IDE and SmartDesign on the Actel website in addition to training classes.

The microcontroller and its peripherals are grouped in this design into a component called Core8051S_sub. Similarly, the memories and associated control logic are grouped into a component called Memory. These two components are then instantiated onto a top-level entity along with other system level components (Figure 2-1).

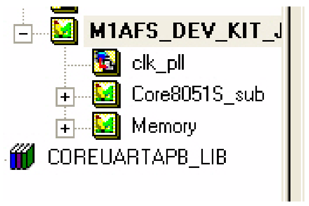


Figure 2-1 · M1AFS_DEV_KIT Components

Step 1 – Create the Libero IDE Project

In this step we will create the top-level entity and the two component entities.

1. Create a Libero IDE project for this design having the following characteristics, based on the target hardware board.

   For the M1AFS-DEV-KIT-SCS:
   - Project name: M1AFS-SCS_8051S_TUT
   - Family: Fusion
   - Die: M1AFS600
   - Package: 484 FBGA

   For the M1AFS-EMBEDDED-KIT:
   - M1AFS_EMB_8051S_TUT
   - Family: Fusion
   - Die: M1AFS1500
   - Package: 484 FBGA

   For the M1AFS-ADV-DEV-KIT:
   - Project name: M1AFS_ADV_8051S_TUT
   - Family: Fusion
   - Die: M1AFS1500
   - Package: 484 FBGA

2. Select **File** > **Import Files** from the main Libero IDE tabs. In the Import Files dialog box, select **Physical Design Constraint Files (*.pdc)** for the type of files. Browse to the **Tutorial_Files** folder for your board and click on the *.pdc file. Click **Import**.

Step 2 – Create the Microcontroller Subsystem

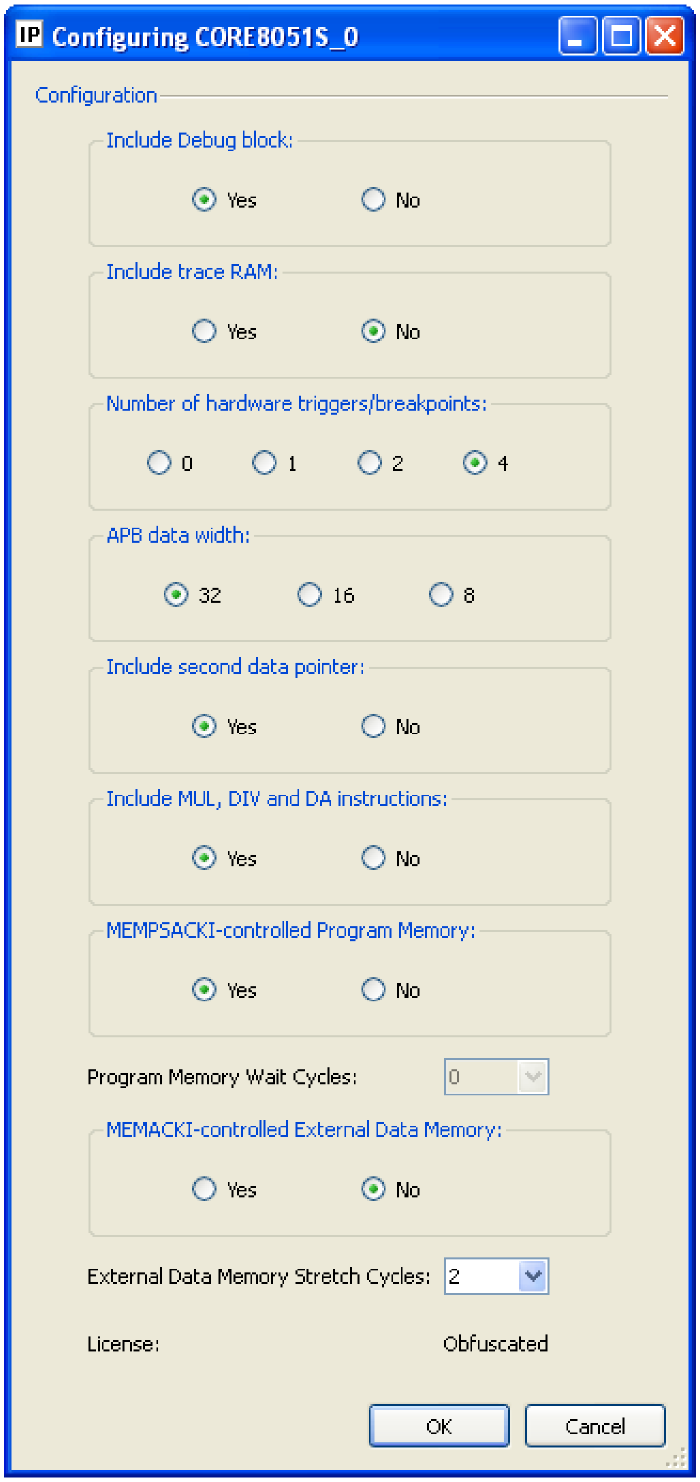
This consists of the microcontroller core, its peripherals, and interconnections.

1. Use SmartDesign to create the 8051 core subsystem file. Give the SmartDesign component the name Core8051S_sub.
2. Create an instance of Core8051s:

   Core8051s is a microcontroller core that is instruction set compatible with the 8051. It contains the main 8051 core logic but no peripheral logic. Core8051s has an APB bus interface that can be used to easily expand the functionality of the core by connecting it to existing APB IP peripherals. This allows users to configure the core with the peripheral functions (timers, UARTs, I/O ports, etc.) that they need for their application.

   In the Libero IDE catalog, under the heading Processors, find and instantiate an instance of Core8051s. Configure this core as indicated below, shown in Figure 2-2 on page 11:

   - Include debug block: Yes
   - Include trace RAM: No
   - Number of hardware triggers/breakpoints: 4
   - APB data width: 32
   - Include second data pointer: Yes
   - Include MUL, DIV, and DA instructions: Yes
   - MEMPSACKI-controlled Program Memory: Yes
   - MEMACKI-controlled External Data Memory: No
   - External Data Memory Stretch Cycles: 2


Configuring CORE8051S_0

Configuration

Include Debug block: [✓ Yes] No

Include trace RAM: Yes [✓ No]

Number of hardware triggers/breakpoints: 0 1 2 [✓ 4]

APB data width: [✓ 32] 16 8

Include second data pointer: [✓ Yes] No

Include MUL, DIV and DA instructions: [✓ Yes] No

MEMPSACKI-controlled Program Memory: [✓ Yes] No

Program Memory Wait Cycles: 0

MEMACKI-controlled External Data Memory: Yes [✓ No]

External Data Memory Stretch Cycles: 2

License: Obfuscated

Figure 2-2 · Configure Core8051s

The debug block is necessary if you want to use the JTAG interface for debugging your design.

Trace RAM is not supported in the FlashPro3 debugger, except as an additional cost option. There is no need to include it in the design unless your Flashpro3 is licensed for this feature. For additional information, visit the Core8051 web page from http://www.actel.com/products/ip/ by selecting DirectCore then Core8051s. Scroll down the page to find the 8051 Trace Debugger Upgrade.

The number of hardware triggers / breakpoints is set at four.

The APB databus width is selectable at 8, 16, or 32 bits wide. It must be wide enough to accommodate the width of the widest peripheral in the design. Core GPIO currently needs to have a 32 inputs and outputs in order to simulate properly. This implies that APB3 bus width should be set to 32-bits.

The second DPTR is not currently supported by C compilers. Assembly language programs may make use of it. The second DPTR is useful for memory block moves.

Core8051s has an option to include or omit multiply and divide and the decimal adjust (DA) instructions in order to reduce the number of tiles used. Caution should be exercised in omitting these instructions when using a C compiler.

MEMPSACKI is an input that asserts wait states when reading code memory. The system designer has an option to use the MEMPSACKI signal or use a fixed number of wait states. The internal Flash memory block in the Fusion FPGA (NVM) used in this design provides a busy signal that can be inverted and connected to MEMPSACKI.

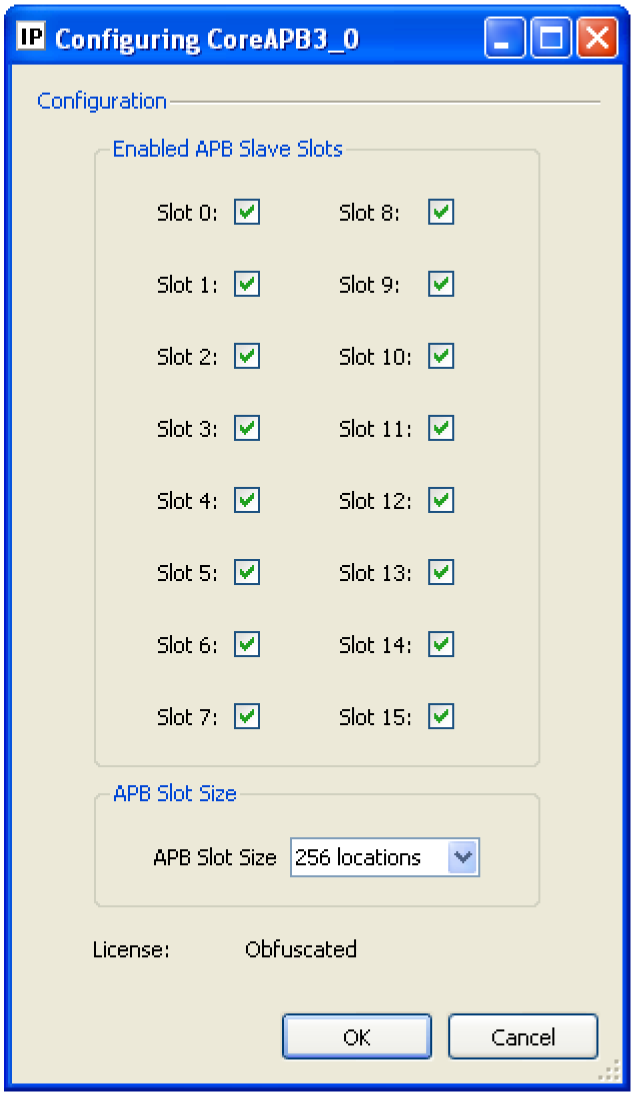
MEMACKI is an input that asserts wait states when reading data memory. The system designer has an option to use the MEMACKI signal or use a fixed number of wait states. The SRAM used in this design does not have a busy/non-busy signal available, thus two fixed wait states (External Data Memory Stretch Cycles) are used.

3. Create an instance of CoreAPB3.

CoreAPB3 is an AMBA bus interface that is used to connect subsystem cores to Actel's soft processors, such as Core8051s. The bus interface is easy to use and fully compatible with the APB v3.0 protocol. The core is constructed to allow easy connection of IP cores in systems built around the Core8051s processor.

In the Libero IDE catalog, under the heading Bus Interfaces, find and instantiate an instance of CoreAPB3. Configure this core as indicated below (Figure 2-3 on page 13):

- All slots selected
- APB Slot size: 256 locations



Configuring CoreAPB3_0

Configuration

Enabled APB Slave Slots

Slot 0:	✓	Slot 8:	✓
Slot 1:	✓	Slot 9:	✓
Slot 2:	✓	Slot 10:	✓
Slot 3:	✓	Slot 11:	✓
Slot 4:	✓	Slot 12:	✓
Slot 5:	✓	Slot 13:	✓
Slot 6:	✓	Slot 14:	✓
Slot 7:	✓	Slot 15:	✓

APB Slot Size

APB Slot Size: 256 locations

License: Obfuscated

Figure 2-3 · Configuring CoreAPB3

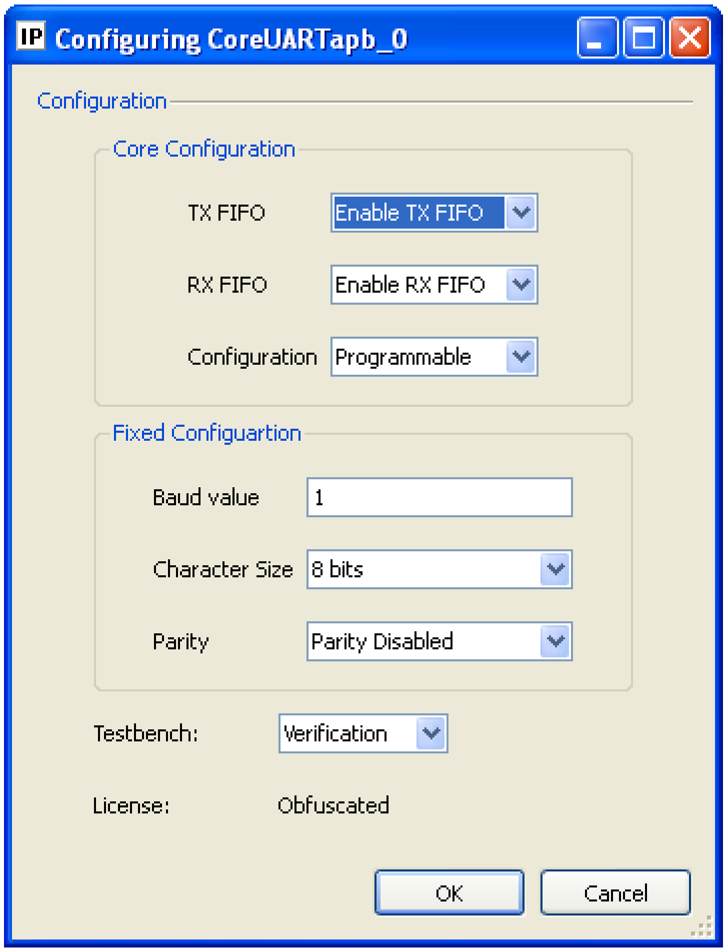
4. Create an instance of CoreUARTapb.

   CoreUARTapb is a serial communications interface that is intended primarily for use in embedded systems. The controller can operate in either an asynchronous (UART) or synchronous mode. In asynchronous mode, CoreUARTapb can be used to interface directly to industry standard UARTs. CoreUARTapb has an APB-wrapper that adds an APB interface allowing the core to be connected to the APB bus and controlled by an APB bus master.

   Unlike a standard 8051 UART, CoreUARTapb includes a baudrate generator and thus does not need a separate timer for the baudrate.

In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreUARTapb. Configure this core as indicated below (Figure 2-4):

- TX FIFO: Enabled
- RX FIFO: Enabled
- Configuration: Programmable
- Baud value: 1
- Character size: 8 bits
- Parity: Parity Disabled
- Testbench: Verification



TX FIFO: Enable TX FIFO
RX FIFO: Enable RX FIFO
Configuration: Programmable
Baud value: 1
Character Size: 8 bits
Parity: Parity Disabled
Testbench: Verification
License: Obfuscated

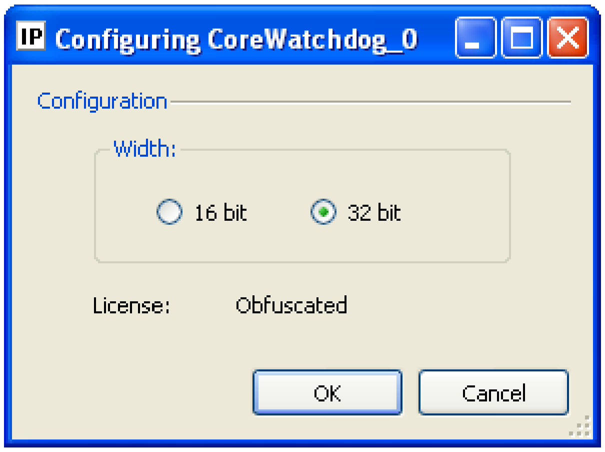
Figure 2-4 · Configuring CoreUARTapb

5. Create an instance of CoreWatchdog.

   CoreWatchdog is an APB slave that provides a means of recovering from software crashes. When enabled, the core will generate a soft reset if the microprocessor fails to refresh it on a regular basis. CoreWatchdog is based on a decrementing counter which can assert a reset signal if it is allowed to time out. The width of the decrementing counter can be configured as either 16 or 32 bits. Processor-accessible registers in CoreWatchdog provide a means to control and monitor the operation of the core.

   In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreWatchdog. Configure this core as indicated below (Figure 2-5):

   - Configuration: 32-bit


Configuring CoreWatchdog_0

Configuration

Width: ( ) 16 bit [✓ 32 bit]

License: Obfuscated

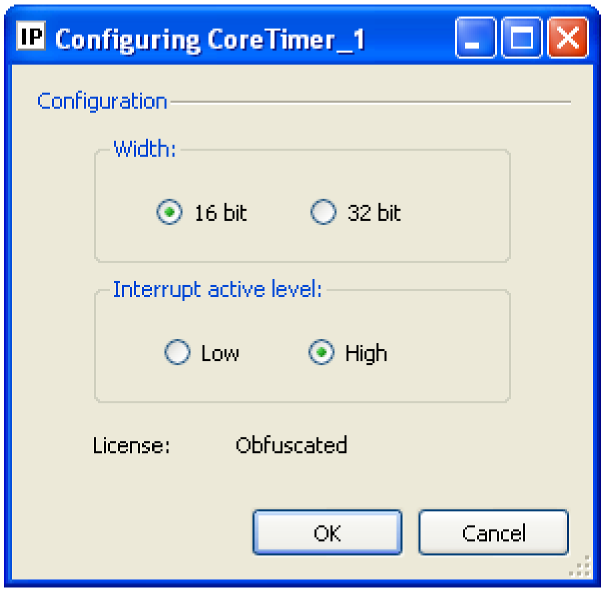
Figure 2-5 · Configuring CoreWatchdog

6. Create the first instance of CoreTimer.

   CoreTimer is an APB slave that provides the functionality for an interrupt-generating, programmable decrementing counter. CoreTimer is configurable and programmable, and can be used in either continuous or one-shot modes. This core is an essential element in many designs because it supports accurate generation of timing for precise application control.

In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreTimer. Configure this core as indicated below (Figure 2-6):

- Width: 16-bit
- Interrupt active level: high



Width: [✓ 16 bit] 32 bit

Interrupt active level: Low [✓ High]

License: Obfuscated

Figure 2-6 · Configuring CoreTimer

7. Create a second instance of CoreTimer.

   This second CoreTimer instance is included in the design simply to provide more timer flexibility in the system.

   In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreTimer. Configure this core as indicated below:

   - Width: 16-bit
   - Interrupt active level: High

8. Create an instance of CoreGPIO.

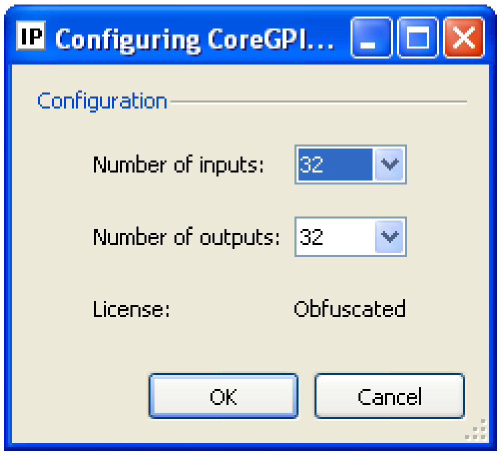
   CoreGPIO is an APB bus peripheral that provides up to 32 inputs and 32 outputs for general purpose use. There is a single register at offset 0x00 and aliased throughout the slot. Writing to this register writes 32 bits to the outputs. Reading from the register reads the state of the inputs.

   It is not required that all inputs and outputs be used. The user need only connect to those inputs or outputs which are actually used.

   Currently CoreGPIO needs to be configured for 32 inputs and 32 outputs in order to simulate correctly. Unused inputs can be tied low and unused outputs marked as unused.

In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreGPIO. Configure this core as indicated below (Figure 2-7):

- Number of inputs: 32
- Number of outputs: 32



Number of inputs: 32

Number of outputs: 32

License: Obfuscated

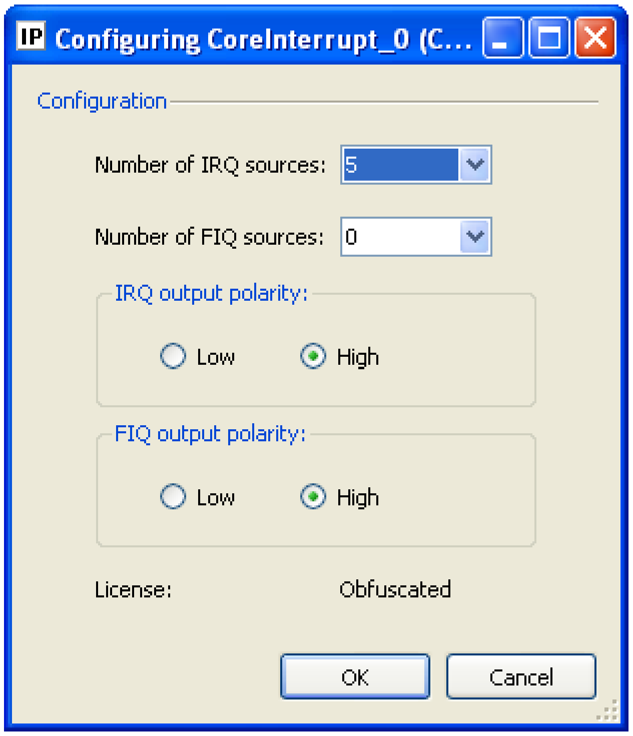
Figure 2-7 · Configuring CoreGPIO

9. Create an instance of CoreInterrupt.

   CoreInterrupt is an APB Slave component that provides configurable interrupt processing. It supports 0 to 32 IRQ sources and 0 to 8 FIQ sources. The IRQ and FIQ source inputs are level-sensitive, active high ports. These interrupt sources are processed to produce two output interrupt lines: IRQ and FIQ.

   In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreInterrupt. This core provides two interrupt outputs, IRQ and FIQ, which have significance in an ARM7 or Cortex-M1 design. In a Core8051s design there is no difference and these two outputs can be used as two channels of interrupt control. Configure this core as indicated below (Figure 2-8 on page 18):

   - Number of IRQ sources: 5
   - Number of FIQ sources: 0
   - IRQ output polarity: High
   - FIQ output polarity: High



Number of IRQ sources: 5

Number of FIQ sources: 0

IRQ output polarity: Low [✓ High]

FIQ output polarity: Low [✓ High]

License: Obfuscated

Figure 2-8 · Configuring CoreInterrupt

10. Create an instance of CoreAI.

CoreAI allows for easy control of the on-chip analog hardware peripherals within the Fusion family of Actel devices. The industry-standard APB slave interface is used as the primary control mechanism within CoreAI. Like all DirectCores, CoreAI is designed, verified, and fully supported to make it easy to use, enabling designers to get their products to market faster.

CoreAI provides flexible capabilities for the analog inputs. In this tutorial some analog inputs will be configured to read voltages, some to read the differential voltage between two nodes (current monitor), and others to read temperature.

In the Libero IDE catalog, under the heading Peripherals, find and instantiate an instance of CoreAI. Configure this core as indicated below (Figure 2-9 on page 20 and Figure 2-10 on page 21):

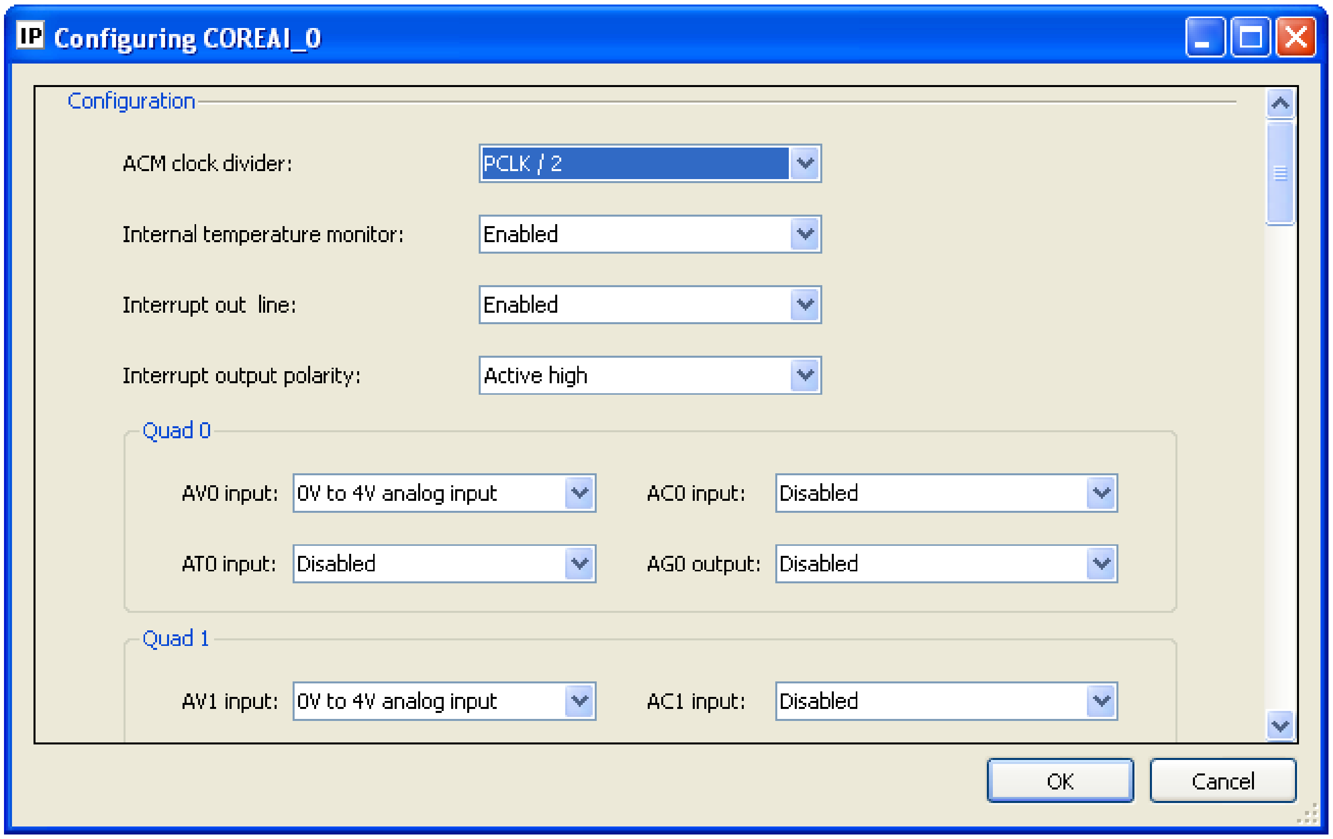
If your target is the M1AFS-DEV-KIT-SCS board, configure this core as indicated below:

- ACM clock divider: PCLK/2
- Internal temperature monitor: Enabled
- Interrupt out line: Enabled
- Interrupt output polarity: Active high
- Quad 6 AV6 input: 0 V to 4 V analog input
- Quad 6 AC6 input: Current Monitor
- Quad 7 AV7 input: 0 V to 2 V analog input
- Quad 7 AC7 input: Current Monitor
- Quad 8 AV8 input: 0 V to 8 V analog input
- Quad 8 AT8 input: Temperature Monitor
- Quad 9 AV9 input: 0 V to 4 V analog input
- Quad 9 AC9 input: Current Monitor
- Quad 9 AT9 input: Temperature Monitor
- Use Real Time Clock: No
- VAREFSEL input control: Fixed
- Fixed VAREFSEL value: Output 2.56 V
- ADC Mode control: Register controlled
- TVC[7:0] pins control: Register controlled
- STC[7:0] pins control: Register controlled
- Use ADC conversions FIFO: No
- APB interface width: 16 bits

If your target is the M1AFS-EMBEDDED-KIT or M1AFS-ADV-DEV-KIT board, configure this core as indicated below:

- ACM clock divider: PCLK/2
- Internal temperature monitor: Enabled
- Interrupt out line: Enabled
- Interrupt output polarity: Active high
- Quad 0 AV0 input: 0 V to 4 V analog input
- Quad 0 AC0 input: Current Monitor
- Quad 1 AV1 input: 0 V to 2 V analog input
- Quad 1 AC1 input: Current Monitor
- Quad 2 AT2 input: Temperature Monitor

- Quad 4 AC4 input: 0 V to 4 V
- Use Real Time Clock: No
- VAREFSEL input control: Fixed
- Fixed VAREFSEL value: Output 2.56 V
- ADC Mode control: Register controlled
- TVC[7:0] pins control: Register controlled
- STC[7:0] pins control: Register controlled
- Use ADC conversions FIFO: No
- APB interface width: 16 bits



ACM clock divider: PCLK / 2

Internal temperature monitor: Enabled

Interrupt out line: Enabled

Interrupt output polarity: Active high

Quad 0

AV0 input: 0V to 4V analog input

AC0 input: Disabled

AT0 input: Disabled

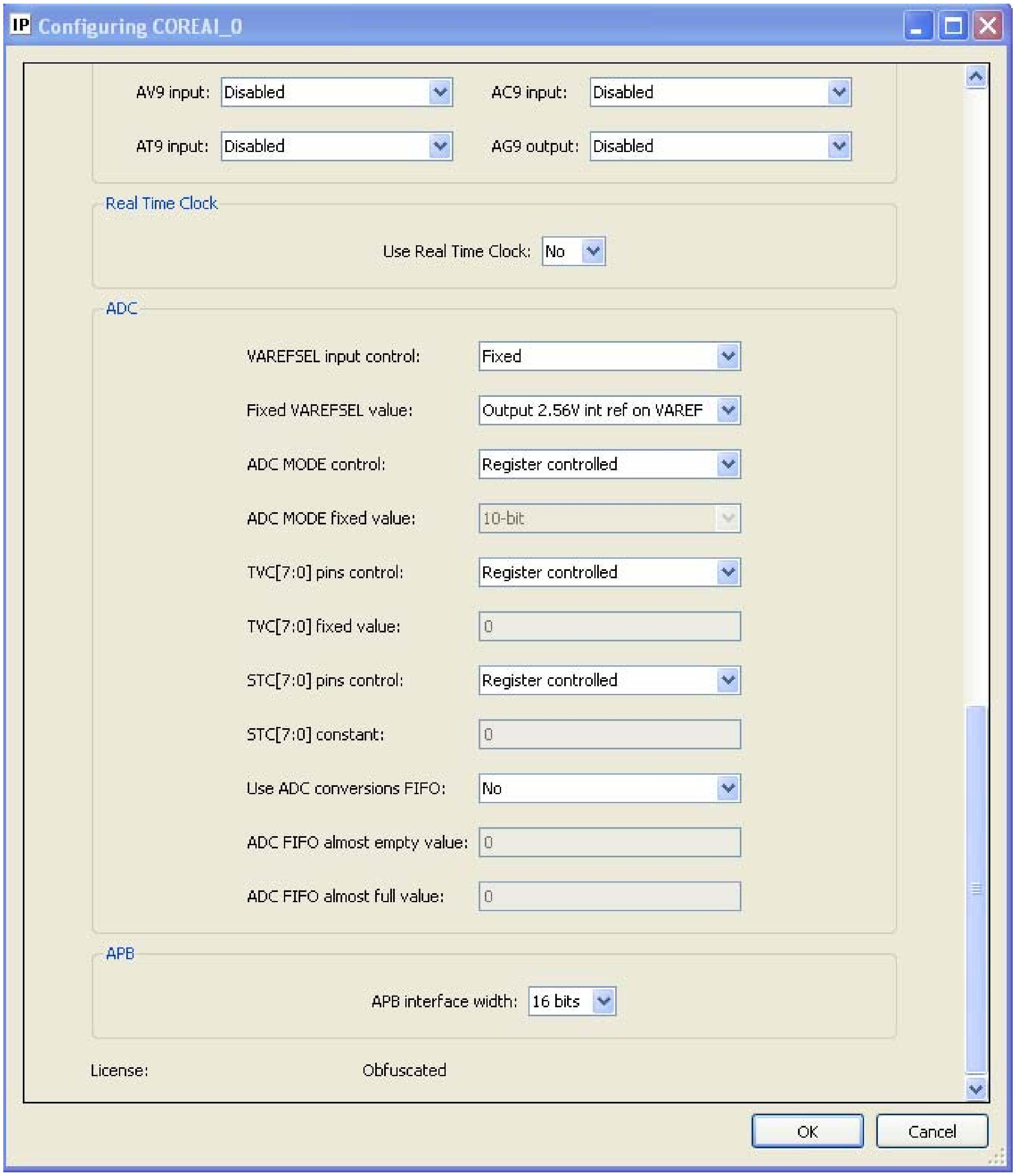
AG0 output: Disabled

Quad 1

AV1 input: 0V to 4V analog input

AC1 input: Disabled

Figure 2-9 · Configuring CoreAI (part 1)



AV9 input: Disabled
AC9 input: Disabled
AT9 input: Disabled
AG9 output: Disabled

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

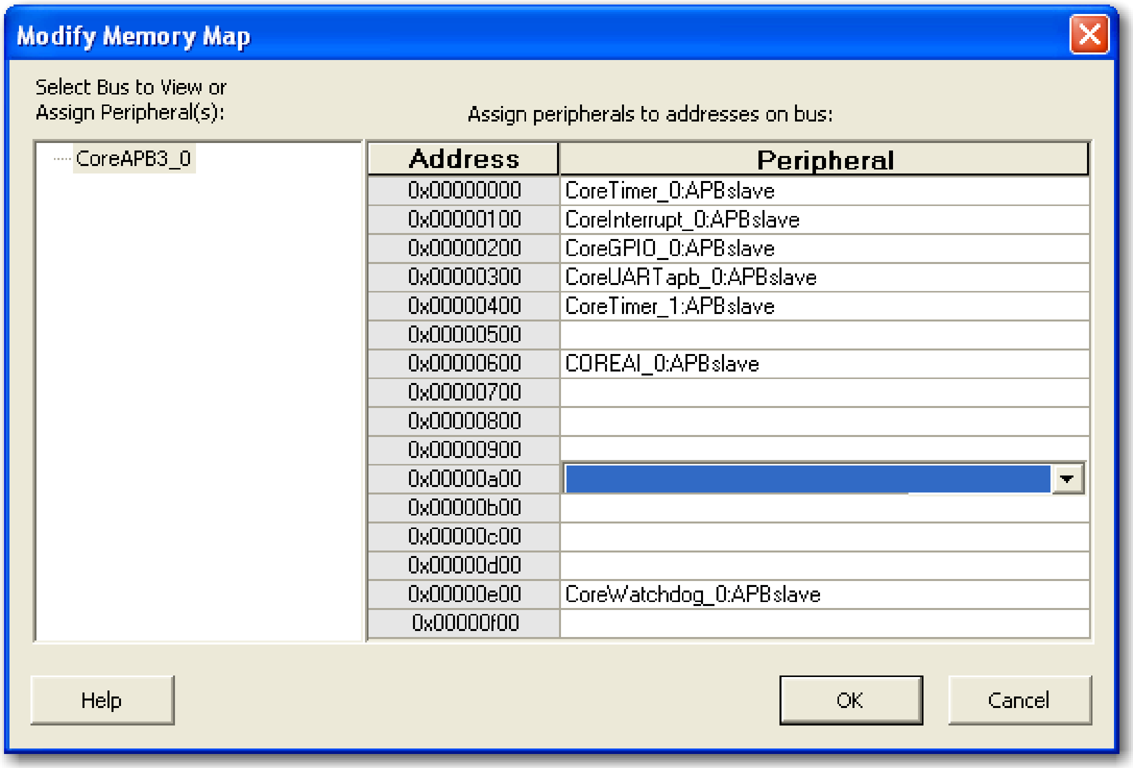
License: Obfuscated

Figure 2-10 · Configuring CoreAI (part 2)

11. Connect peripherals to the APB3.

    Right-click on the Core8051S_sub canvas and select **Auto Connect**. This will connect the MCU and the peripherals to the APB3 bus. The Modify Memory Map box will open. SmartDesign has assigned APB3 slots for each peripheral automatically. In this step you will change those assignments. Clear the assigned peripheral for each address by clicking on the peripheral name for each addresses and selecting the blank (empty) line. Then assign peripherals to the address slots indicated below (Figure 2-11):

    - 0x0000 0000: CoreTimer_0
    - 0x0000 0100: CoreInterrupt
    - 0x0000 0200: CoreGPIO
    - 0x0000 0300: CoreUARTapb
    - 0x0000 0400: CoreTimer_1
    - 0x0000 0600: CoreAI
    - 0x0000 0e00: CoreWatchdog


Modify Memory Map
Select Bus to View or Assign Peripheral(s):
CoreAPB3_0
Assign peripherals to addresses on bus:
Help
OK
Cancel

Address	Peripheral
0x00000000	CoreTimer_0:APBslave
0x00000100	CoreInterrupt_0:APBslave
0x00000200	CoreGPIO_0:APBslave
0x00000300	CoreUARTapb_0:APBslave
0x00000400	CoreTimer_1:APBslave
0x00000500	
0x00000600	COREAI_0:APBslave
0x00000700	
0x00000800	
0x00000900	
0x00000a00	
0x00000b00	
0x00000c00	
0x00000d00	
0x00000e00	CoreWatchdog_0:APBslave
0x00000f00	

Figure 2-11 · Modify Memory Map

12. Promote signals to the top level.

    SmartDesign automatically promoted some signals to the top of the Core8051S_sub block. These include SYSCLK, NSYSRESET, and some analog signals from CoreAI. Promoting signals to the top level allows these signals to be available for connection to other blocks. Only signals that are promoted to the top level will show as ports of the Core8051S_sub component. Now you must promote some additional signals to the top level of this block so that these signals can be connected to other components in the system design.

    Right-click on the name of a port or the port connector and select **Promote to Top Level** to promote a signal to the top level. Promote the signals in Table 2-1 to the top level (Figure 2-12).

Table 2-1 · Promote Signals to Top Level

Block	Port
Core8051s_0	ExternalMemIf
	DebugIf
	MOVX
CoreUARTapb_0	RX
	TX
CoreGPIO	dataIn[31:0]
	dataOut[31:0]
CoreInterrupt	irqSource3

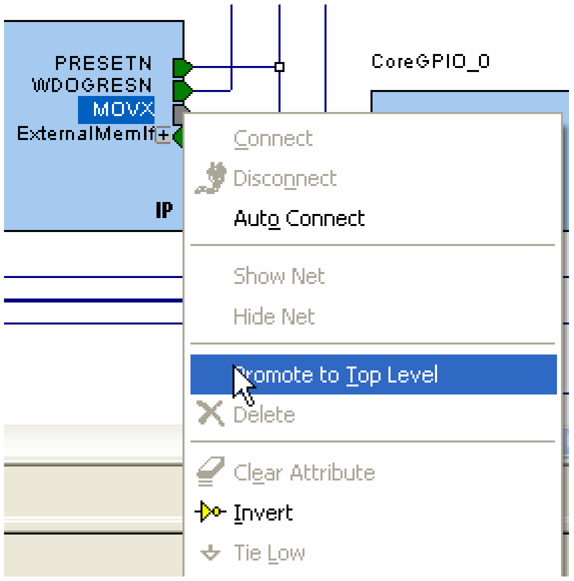


Figure 2-12 · Promote Signal to Top Level

13. Change a top-level port name.

Locate the top level port irqSource3 on the input (left) side of the canvas. This name does not convey any sense of its use in the system. Rename this signal to make its function more easily recognized. Right-click on this port and select **Modify Port** (Figure 2-13). Change the name of this port to **EXTIRQ**, since this signal is used as the external interrupt request in the system (Figure 2-14).

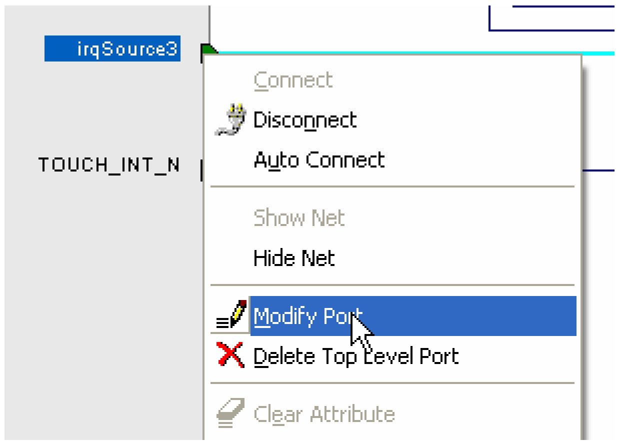
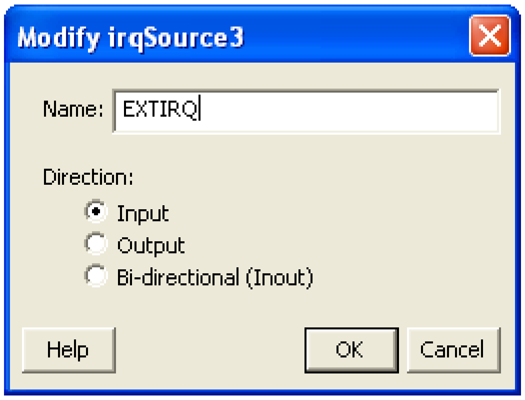


Figure 2-13 · Modify Port


Modify irqSource3

Name: EXTIRQ

Direction:
[✓ Input]
Output
Bi-directional (Inout)

Figure 2-14 · Change Port Name

14. Make connections between ports.

    While holding the CTRL key, click the first port you wish to connect. The port name will be highlighted. Continue holding the CTRL key and click on the next port to connect. When you have highlighted the last port to connect in the net, right-click on this last port and select **Connect**. This will connect the highlighted ports together.

    Most of the peripherals in this design have outputs that can be used as interrupts to the microcontroller core. Core8051s has two non-maskable interrupt inputs. CoreInterrupt can accept several interrupt inputs and provide masking capability. We need to connect interrupt request outputs from the peripherals to the interrupt request inputs of CoreInterrupt. Make the connections listed in Table 2-2.

Table 2-2 · Port Connections

Connection	Block	Port
1	Core8051s	INT0
	CoreInterrupt	IRQ
2	CoreUARTapb	RXRDY
	CoreInterrupt	irqSource2
3	CoreUARTapb	TXRDY
	CoreInterrupt	irqSource1
4	CoreTimer_0	TIMINT
	CoreInterrupt	irqSource0
5	CoreTimer_1	TIMINT
	CoreInterrupt	irqSource4

15. A port can be tied low by right-clicking on the port and selecting **Tie Low**. More than one port can be tied low by holding down the CTRL key while clicking on each of the ports. After clicking the last of the ports, right-click and select **Tie Low**. Tie the signals listed in Table 2-3 low.

Table 2-3 · Tie Signals Low for CoreAI and CoreInterrupt

Block	Port
CoreAI	DDGDON[9:0]
	RTCCLK
CoreInterrupt	fiqSource[7:0]
	irqSource[31:5]

16. Select **Check Design Rules** from the SmartDesign tab in the Libero IDE main menu. Verify that there are no errors. Floating driver warnings can be ignored. Right-click in the Core8051S_sub canvas and select **Generate Design**.
17. This concludes the design of the 8051s subsystem (a good time to save your file). Close the Core8051S_sub component.

Step 3 – Create the Memory Subsystem

The memory subsystem consists of the 64 KB of flash memory (NVM) code memory, 8 KB of SRAM data memory, an interface to an off-FPGA SRAM, and control and interface logic. The off-FPGA SRAM provides 56 KB of SRAM data memory in the 8051's external data memory space and 64 KB of SRAM code memory. This SRAM code memory is intended for use with the debugger for debugging purposes.

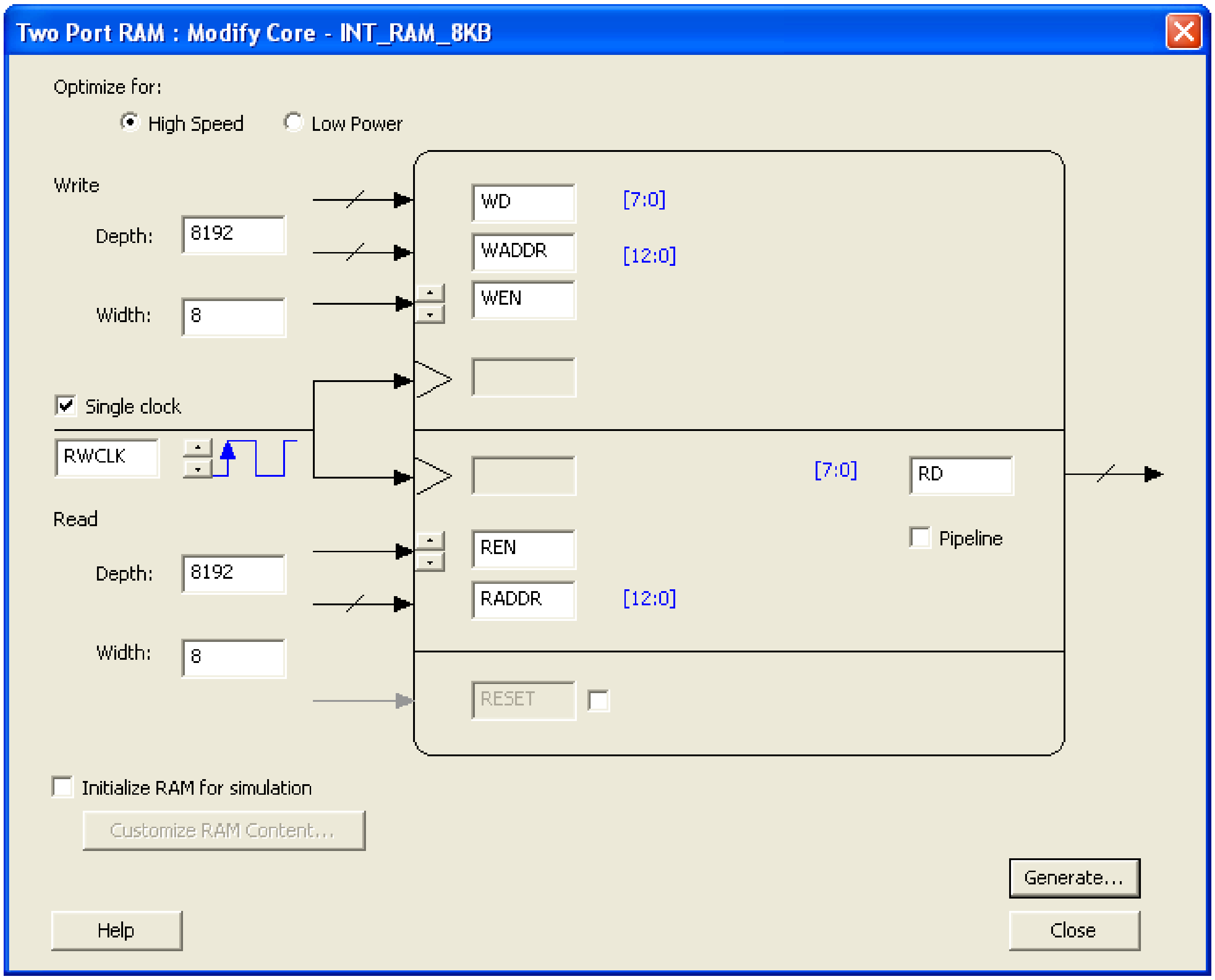
The memory system uses a 16-bit wide address bus for code and data memory, providing 64 KB of addressable memory space. The upper 4 KB of this space (address 0xF000 to 0xFFF) is used by the APB3 interface. When the APB3 interface is accessed, Core8051s will not generate memory access cycles on this address bus but instead will generate accesses on the APB3. Thus it is not necessary to exclude the upper 4 KB for the memory space. This simplifies memory decoding logic.

The data bus used for code and data memory is 8 bits wide and should not be confused with the large data bus width used for the APB3 and the peripherals.

1. Use SmartDesign to create the memory subsystem file. Give the SmartDesign component the name Memory.
2. Create the internal RAM.

   In the Libero IDE catalog, under the heading Memory & Controllers, instantiate an instance of the **RAM – Two Port** component. Configure as indicated below (Figure 2-15 on page 27):

   - Optimize for: High Speed
   - Write Depth: 8192
   - Write Width: 8
   - WEN: Choose active high
   - Clock: Select single
   - Read Depth: 8
   - REN: Choose active high
   - Reset: Clear check box
   - Pipeline: Clear check box
   - Clock: Select rising edge



Two Port RAM : Modify Core - INT_RAM_8KB

Optimize for: [✓ High Speed] Low Power

Write

Depth: 8192

Width: 8

[✓] Single clock

RWCLK

Read

Depth: 8192

Width: 8

WD [7:0]

WADDR [12:0]

WEN

RD [7:0]

[ ] Pipeline

REN

RADDR [12:0]

RESET [ ]

[ ] Initialize RAM for simulation

Figure 2-15 · Configure Two-Port RAM

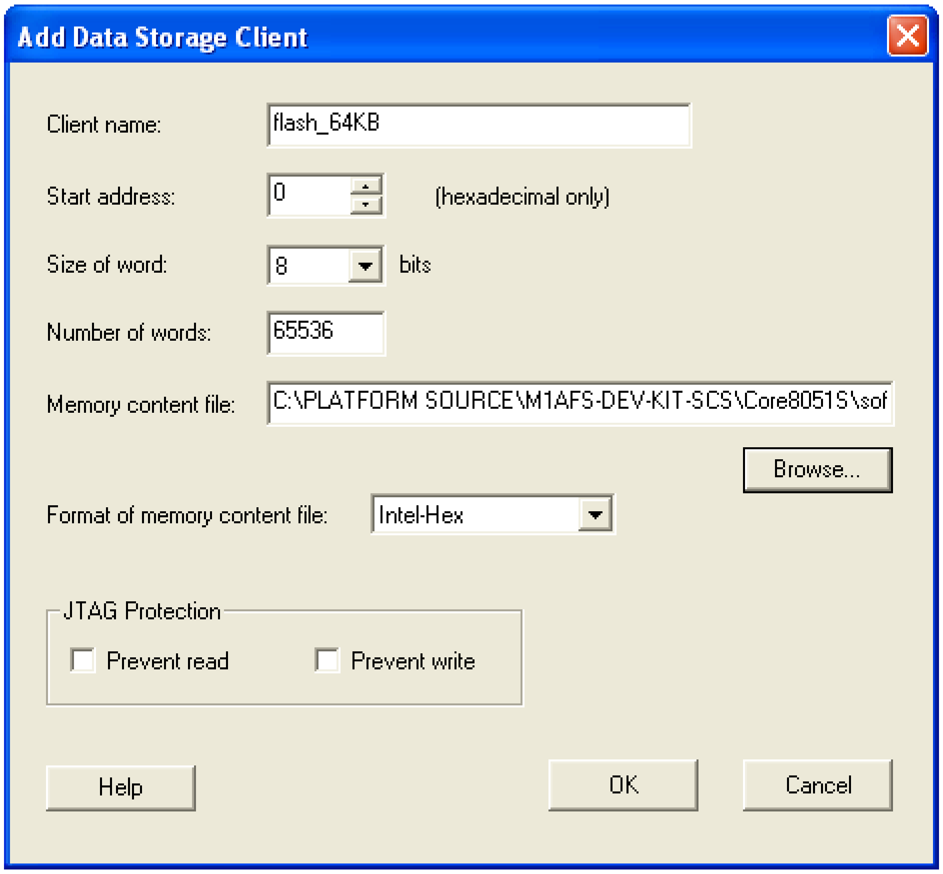
Click **Generate** and name the component **INT_RAM_8KB**.

3. Create the Flash Memory block.

   In the Libero IDE catalog, under the heading Fusion Peripherals, double-click on the **Flash Memory System Builder** item. Double-click on the **Data Storage** client. A box will open. Configure as follows:

   - Client name: flash_64KB
   - Start address: 0
   - Size of word: 8
   - Number of words: 65536
   - Format of file: Intel-Hex
   - JTAG protection: Clear both check boxes

   The memory content file contains the hex code for the end application. You can browse to the *.ihx file provided in the *Tutorial_Files* folder in the *Core8051s_Fusion_FPGA_Design _Tutorial* folder and select this file. Click **Choose** to select the file. Click **OK**. Click **Generate**. Name this block **flash_64KB** (Figure 2-16).



Client name: flash_64KB

Start address: 0

Size of word: 8

Number of words: 65536

Memory content file: C:\PLATFORM SOURCE\M1AFS-DEV-KIT-SCS\Core8051S\sof

Format of memory content file: Intel-Hex

Figure 2-16 · Add Data Storage Client

An instance of flash_64KB will appear on the memory canvas. Right-click on this block and select **Delete**. This will remove this block from the canvas. A wrapper, which instantiates this block, will be used instead.

4. Add user HDL wrappers and decode logic.

   The NVM module created using the Flash Memory System Builder cannot be directly connected to other components in the design. Thus, it needs a wrapper to allow making connections to it. This wrapper is provided.

   This design includes an on-FPGA NVM block, an on-FPGA SRAM block, and an off-FPGA SRAM chip. The memory architecture is described in "Design Overview" on page 7. Some memory decoding logic is required to drive the select signals for the memories, control the bidirectional bus buffer, and the data MUX select lines. This logic is driven by the memory address and the type of memory operation (code read, data read, etc.). This logic is also provided as an HDL file.

   From the main Libero IDE menu, import the following two user HDL files (make certain the file type in the import dialog box is set to HDL source files):

   - MY_NVM.vhd.
   - memdecode.vhd

   Components called nvm_if (MY_NVM.vhd) and mem_decode (memdecode.vhd) will appear in the list of components in the Libero IDE Hierarchy tab. Right-click on each of these components and select **Instantiate in Memory**.

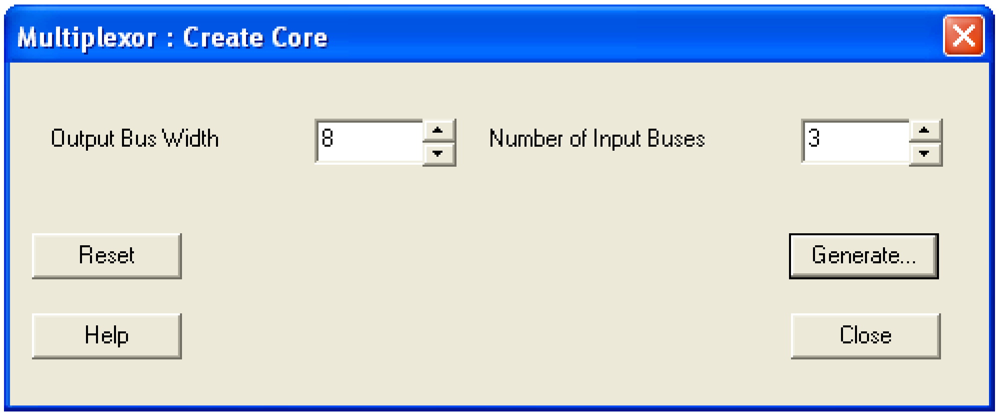
   Right-click on the NVM_BUSY output of the nvm_if_0 block and select **Invert**.

5. Add the data read multiplexer.

   It is not possible to directly connect the outputs of the FPGA memory blocks together, since these block do not have tristate outputs. Instead, use a data multiplexer to gate the outputs onto the microcontroller's input data lines at the appropriate time.

   In the Libero IDE catalog, expand the item **Basic Blocks** and double-click **Multiplexor**. A box will open.

   - Set the bus output width to 8.
   - Set the number of input buses to 3.
   - Click the **Generate** button. A box will open (Figure 2-17).

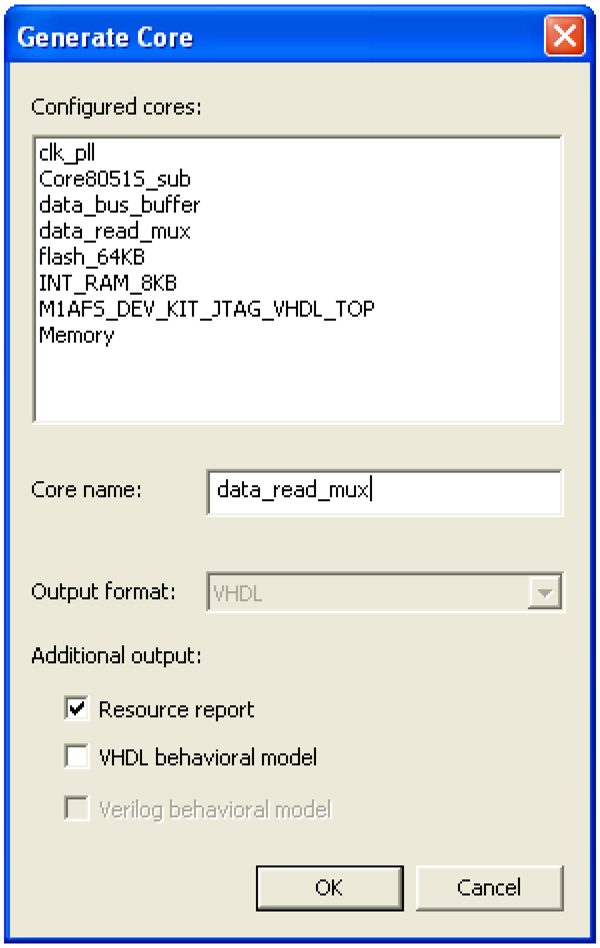

Multiplexor : Create Core

Output Bus Width: 8

Number of Input Buses: 3

Figure 2-17 · Add Multiplexor

- Change the component name to **data_read_mux**.
- Click **OK** (Figure 2-18).


Generate Core

Configured cores:

clk_pll
Core8051S_sub
data_bus_buffer
data_read_mux
flash_64KB
INT_RAM_8KB
M1AFS_DEV_KIT_JTAG_VHDL_TOP
Memory

Core name: data_read_mux

Output format: VHDL

Additional output:

[✓] Resource report
[ ] VHDL behavioral model
[ ] Verilog behavioral model

Figure 2-18 · Change Component Name for MUX

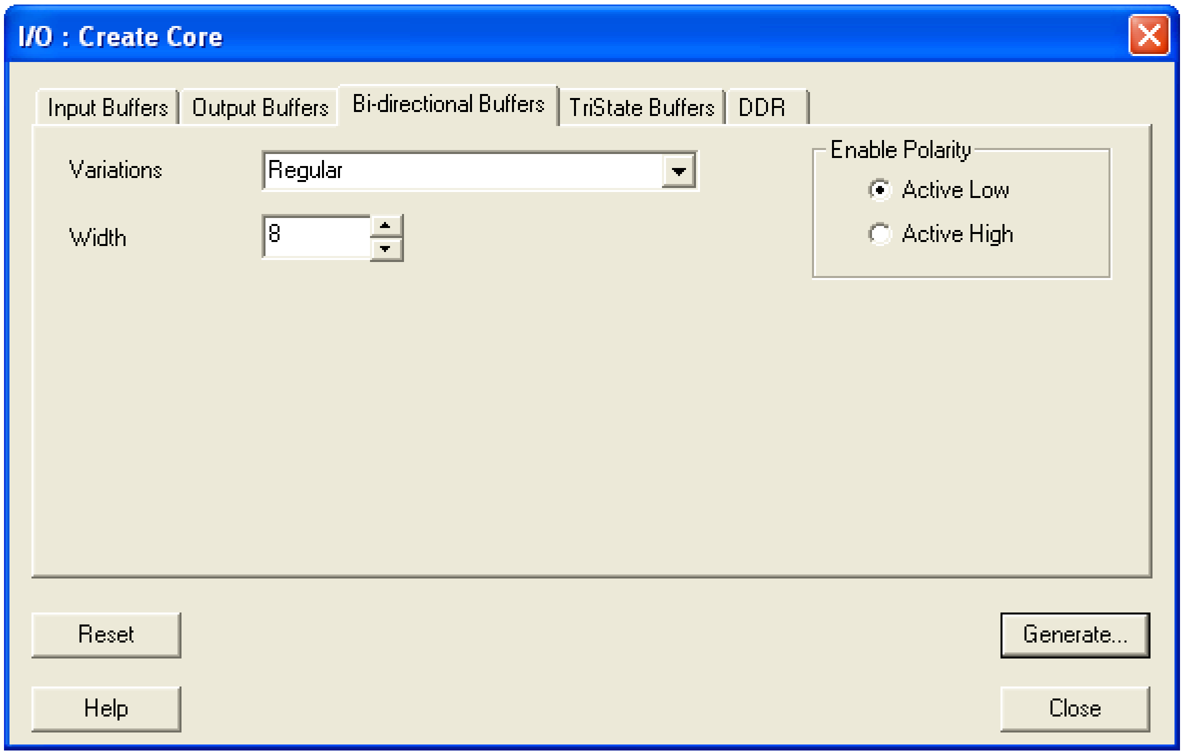
- Right-click on the SmartDesign canvas and select **Auto arrange instances**.

6. Add the databus buffer.

   This design includes off-FPGA memory chips. These devices use a bidirectional databus. In order to interface them, you need a bidirectional buffer. The bidirectional buffer allows you to connect the bidirectional interface of the off-FPGA memories into a separate data_in path and separate data_out path inside the FPGA.

   In the Libero IDE catalog, expand the **Basic Blocks** item and double click on **I/O**. A configuration box will open.

   - Click on the **Bidirectional Buffers** tab.
   - Set buffer width to 8.
   - Select active low polarity.
   - Click **Generate** (Figure 2-19).


I/O : Create Core

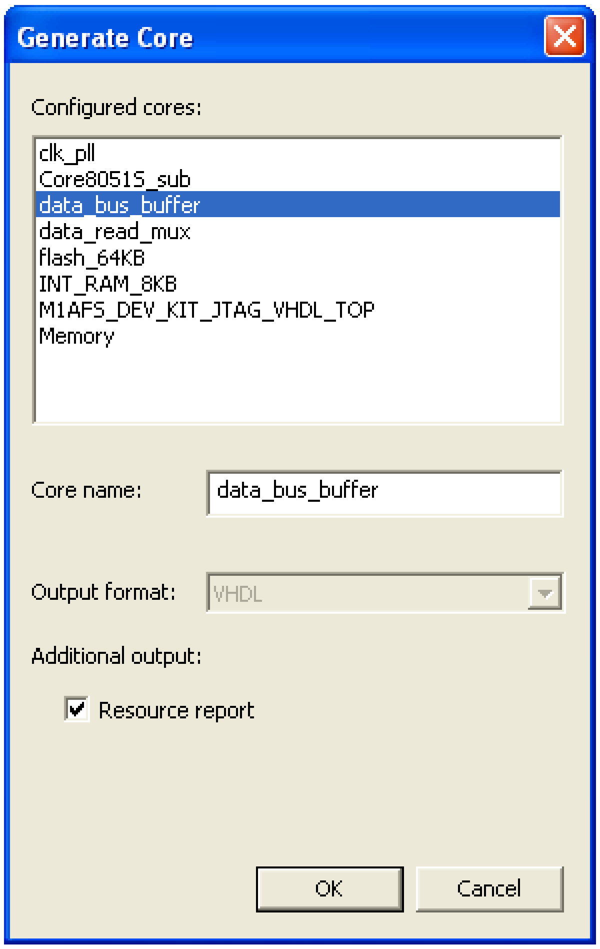
Variations: Regular

Width: 8

Enable Polarity: [✓ Active Low] Active High

Figure 2-19 · Configure Bidirectional Buffers

- Name this component **data_bus buffer** (Figure 2-20). Click **OK**.



clk_pll
Core8051S_sub
data_bus_buffer
data_read_mux
flash_64KB
INT_RAM_8KB
M1AFS_DEV_KIT_JTAG_VHDL_TOP
Memory

Core name: data_bus_buffer

Output format: VHDL

Additional output:

☑ Resource report

Figure 2-20 · Name Bidirectional Buffer

The data bus buffer component will appear on the Memory canvas.

In the following instructions, you will interconnect the components in the Memory component and add top-level ports.

7. A port can be tied low by right-clicking on the port and selecting **Tie Low**. More than one port can be tied low by holding down the CTRL key while clicking on each of the ports. After clicking the last of the ports, right-click and select **Tie Low**. Tie the signals listed in Table 2-4 low.

Table 2-4 · Tie Signals Low for nvm_if_0

Block	Port
nvm_if_0	AUX_BLOCK
	DATA[31:0]
	DISCARD_PAGE
	ERASE_PAGE
	LOCK
	OVERWRITE_PAGE
	OVERWRITE_PROT
	WRITE
	PAGE_STATUS
	PROGRAM
	READ_NEXT
	SPARE_PAGE
	UNPROT_PAGE
	PAGELOSS_PROT
	WIDTH[1:0]

8. Tie the **RESET** port of the nvm_if_0 block high.
9. Click on the SmartDesign button in the Libero IDE menu. Select **Add Port** to add a new top-level port. A box will open. Set the direction to **input**. Enter ADD[17:0] and click **OK**.
10. Similarly, add another new top-level port. Set the direction to **output**. Enter **NVM_BUSY_N** and click **OK**.
11. Add another new top-level port. Set the direction to **input**. Enter **CLK** and click **OK**.
12. Right-click on the **CLK** port and select **Invert**.

13. Now some additional signals must be promoted to the top level of this block. Right-click on the name of a port or the port connector and select **Promote to Top Level** to promote a signal to the top level. Promote the signals in Table 2-5 to the top level.

Table 2-5 · Promote Signals to Top Level

Block	Port
Mem_decode_0	EXT_RAM_CS_N
	EXT_RAM_RD_N
	EXT_RAM_WR_N
	MEMRD
	MEMWR
	PSRD
	PSWR
	A16
	SW0
nvm_if_0	NVM_STATUS[1:0]
data_bus_buffer_0	PAD[7:0]
INT_RAM_8KB	WD[7:0]
data_read_mux_0	Result[7:0]

14. Now change the names of some top-level ports. Right-click on the name of a port or the port connector, select **Modify Port**, and change the port name according to Table 2-6.

Table 2-6 · Modify Port Names

Block	Old port name	New port name
Top level	PAD[7:0]	EXT_DATA[7:0]
	Result[7:0]	RD[7:0]

15. In this step you will separate some busses into smaller group by creating slices. To create a slice, right-click on the name of the port and select **Add slice**. Enter a slice range (the ending and starting bit positions of the slice). Repeat this process if additional slices are needed. Create the slices indicated for the ports shown in Table 2-7.

Table 2-7 · Create Slices for Ports

Block	Port name	slices
nvm_if_0	DOUT[31:0]	[7:0]
	ADD[17:0]	[17:16]
		[15:13]
		[12:0]
Mem_decode_0	SEL[1:0]	[1:1]
		[0:0]
top level	ADD[17:0]	[17:16]
		[15:13]
		[12:0]

16. In this step you will make the connections between ports for the Memory component. Hold the CTRL key and click on the first port you want to connect. The port name will be highlighted. Hold the CTRL key and click on the next port to include in the connection. This next port will be highlighted. When you have highlighted the last port to connect in the net, right-click on this last port and select **Connect**. This will connect the highlighted ports together. Make the connections listed in Table 2-8.

Table 2-8 · Connect Ports for Memory Component

Connection	Block	Port
1	nvm_if_0	DOUT[7:0]
	data_read_mux	Data2_port[7:0]
2	data_read_mux	Data1_port[7:0]
	data_bus_buffer_0	Y[7:0]
2	data_read_mux	Data0_port[7:0]
	INT_RAM_8KB_0	RD[7:0]
4	mem_decode_0	SEL[1]
	data_read_mux	Sel1
5	mem_decode_0	SEL[0]
	data_read_mux	Sel0
6	mem_decode_0	INT_RAM_WR
	INT_RAM_8KB_0	WEN

Table 2-8 · Connect Ports for Memory Component (continued)

Connection	Block	Port
7	mem_decode_0	INT_RAM_RD
	INT_RAM_8KB_0	REN
8	mem_decode_0	INT_NVM_CS
	nvm_if_0	READ
9	mem_decode_0	EXT_RAM_WR_N
	data_bus_buffer_0	Trien
10	top level	CLK
	nvm_if_0	CLK
	INT_RAM_8KB_0	RWCLK
11	INT_RAM_8KB_0	WADDR[12:0]
	INT_RAM_8KB_0	RADDR[12:0]
	nvm_if_0	ADD[12:0]
	top level	ADD[12:0]
12	top level	WD[7:0]
	data_bus_buffer_0	Data[7|0]
	INT_RAM_8KB_0	WD[7:0]
13	mem_decode_0	UP_ADD[15:13]
	nvm_if_0	ADD[15:13]
	top level	ADD[15:13]
14	nvm_if_0	ADD[17:16]
	top level	ADD[17:16]
15	nvm_if_0	NVM_BUSY
	top level	NVM_BUSY_N

17. Select **Check Design Rules** from the SmartDesign tab in the Libero IDE main menu. Verify that there are no errors. Floating driver warnings can be ignored.Right-click in the Memory canvas and select **Generate Design**.
18. This concludes the design of the Memory subsystem (a good time to save your file). Close the Memory subsystem component.

Step 4 – Complete the Top Level

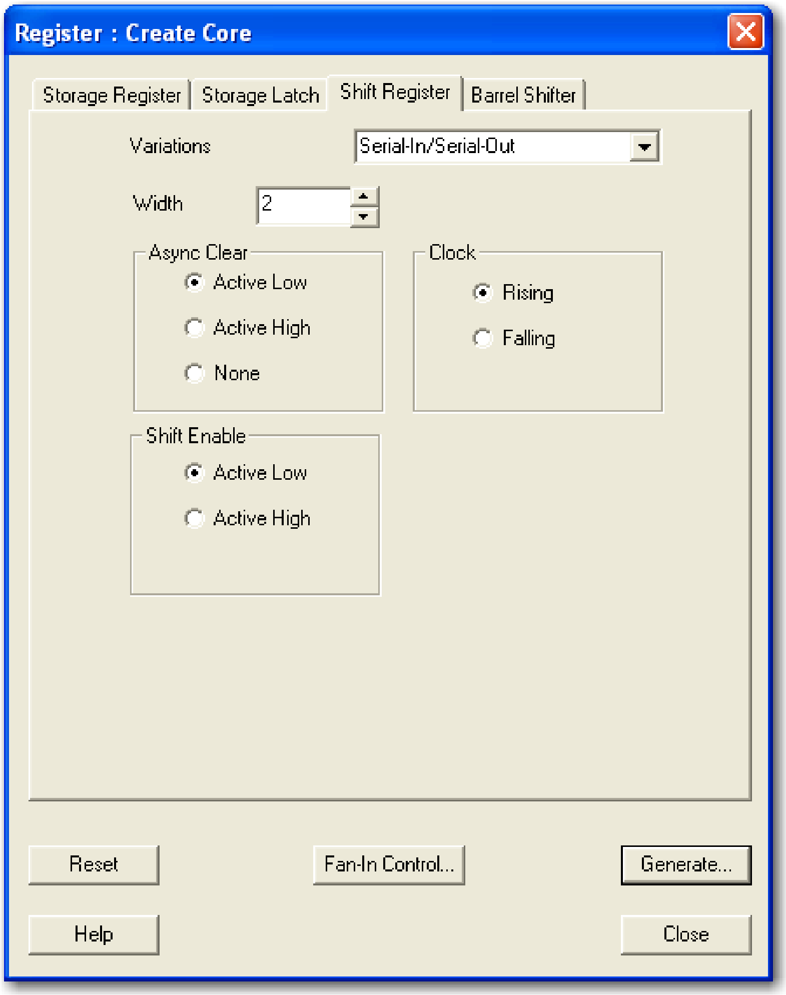
In this step you will create the top level and instantiate the MCU and memory subsystems, add the clock and reset circuits and top-level ports, and make the required connections.

1. Use SmartDesign to create the top-level file. Give the SmartDesign component the name indicated in Table 2-9, based on your target board. Set the file for this component as the root.

Table 2-9 · SmartDesign Components

Target Board	Filename
M1AFS-DEV-KIT-SCS	M1AFS_DEV_KIT_SCS_8051s_TOP
M1AFS-EMBEDDED-KIT	M1AFS_EMBEDDED-KIT_8051s_TOP
M1AFS-ADV-DEV-KIT	M1AFS_ADV_DEV_KIT_8051s_TOP

2. Instantiate the Core8051s subsystem in the top level by right-clicking on the Core8051s component in the Design Explorer window and selecting **Instantiate in** M1AFS...TOP (the exact file name depends on your board).
3. Instantiate the Memory subsystem in the top level by right-clicking on the **Memory** component in the Design Explorer window and selecting **Instantiate in <name of your root file>**.
4. Click on the **SmartDesign** tab at the top of the Libero IDE. Select **Add Port** to add a new top-level port. A box will open. Set the direction to **input**. Enter **EXTCLK** and click **OK**.
5. In the catalog, under Basic Blocks, click **Register**. Select the shift register tab (Figure 2-21 on page 38). Select **Serial-In/Serial-Out** for the variation. Set a width of 2. Select **Active Low** for the async clear. Select rising clock. Select active low shift enable. Then click **Generate**. Give this core the name "reset_delay." Click **OK**.


Register : Create Core

Storage Register | Storage Latch | Shift Register | Barrel Shifter

Variations: Serial-In/Serial-Out

Width: 2

Async Clear: [✓ Active Low] Active High None

Clock: [✓ Rising] Falling

Shift Enable: [✓ Active Low] Active High

Figure 2-21 · Configuring the Shift Register

6. Open the Libero IDE catalog, expand Actel Cells, and locate and click on **INBUF**. An instance of an INBUF will appear in your design. The input will already be promoted to the top level. Right-click on the top-level port for this INBUF input, select **Modify Port**, and change the name of this port to NSYSRESET. Connect the output of this inbuf to the Aclr input of the reset_delay shift register.
7. From the Libero IDE catalog, expand Clock & Management. Locate and double-click on **PLL – static**. The Static PLL box will open (Figure 2-22 on page 39).

   If your target is the M1AFS-DEV-KIT-SCS board:

   Enter **48.000** (MHz) for the CLKA frequency.

   If your target is the M1AFS-EMBEDDED-KIT or M1AFS-ADV-DEV-KIT board:

   Enter **50.000** (MHz) for the CLKA frequency.

Select **External I/O** from the Clock Source menu. For the Primary frequency, enter **12.000** (MHz). Click the **Generate** button. A box will open. Name this component **clk_pll**. Click **OK**. Close the Static PLL – Create Core box. An instance of the PLL will appear on the top canvas.

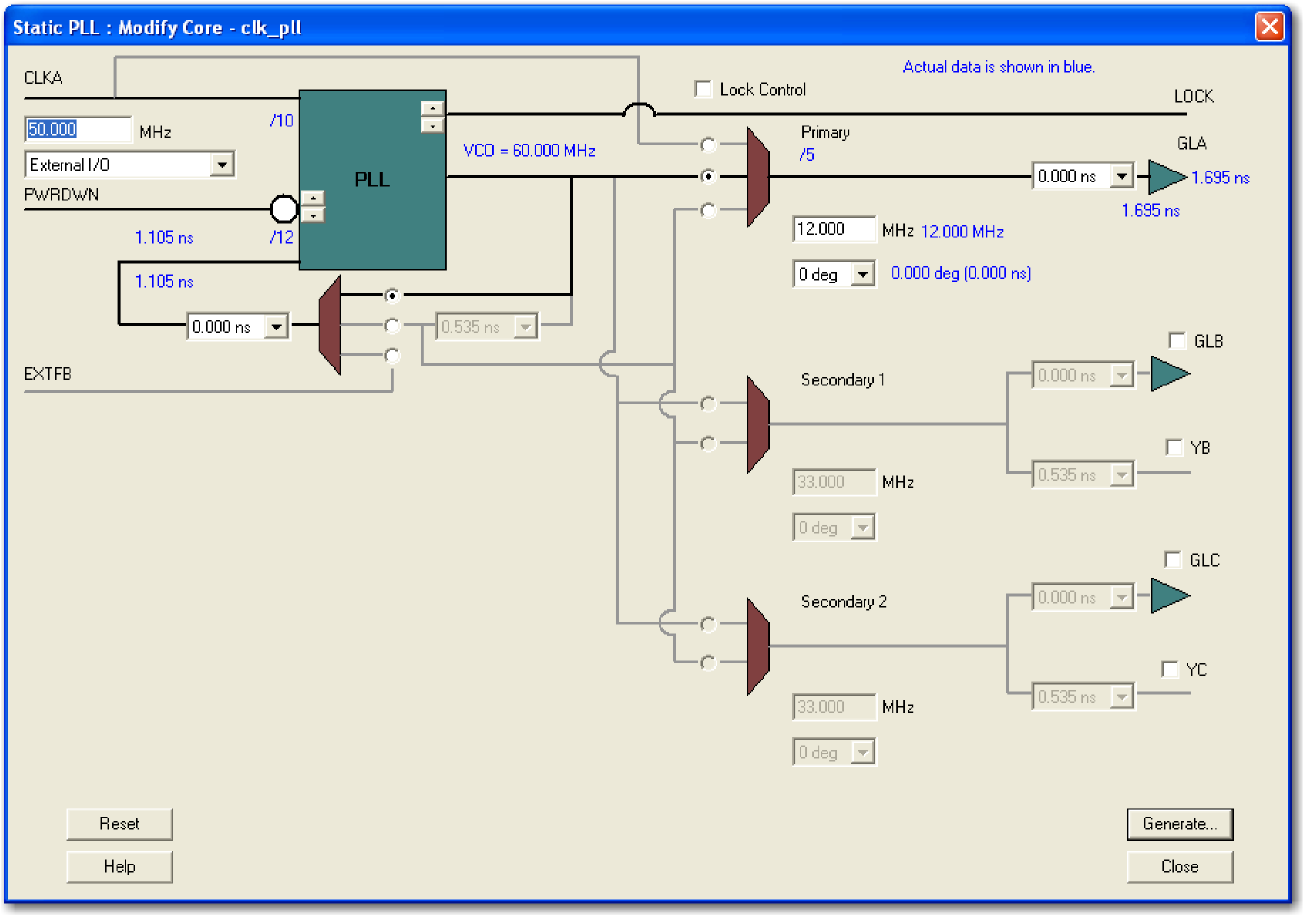


Figure 2-22 · PLL Configuration

8. Some signals must be promoted to the top level of this block. Right-click on the name of a port or the port connector and select **Promote to Top Level** to promote a signal to the top level. Promote the signals in Table 2-10 to the top level.

   Note: Ensure that the External Memory Interface Ports and the Debug Interface Ports are expanded on the Core8051S_sub component when making the top-level signal promotions shown in Table 2-10.

Table 2-10 · Promote Top-Level Signals

Block	Port
Core8051S_sub	MEMADDR[15:0]
	TCK
	TDI
	TMS
	TRSTN
	TDO
	RX
	TX
	Analog Pads
Memory_0	SW0
	EXT_RAM_CS_N
	EXT_RAM_RD_N
	EXT_RAM_WR_N
	A16
	EXT_DATA[7:0]
reset_delay_0	Shiften

9. Right-click on the A16 top-level port and select **Modify Port**.

   If your target is the M1AFS-DEV-KIT-SCS board:

   • Change the name of the port to **MEMADDR18**.

   If your target is the M1AFS-EMBEDDED-KIT or M1AFS-ADV-DEV-KIT board:

   • Change the name of the port to **MEMADDR16**.

10. Click on the SmartDesign tab at the top of the Libero IDE menu. Select **Add Port** to add a new top-level port. A box will open. Set the direction to **Output**. Type in the name of the new port. Click **OK**.

    If your target is the M1AFS-DEV-KIT-SCS board, create the following additional top-level ports:

    - SRBS1_N
    - Flash_HCE_N
    - Flash_LCE_N
    - SRBS2_N
    - Flash_WE_N
    - Flash_RST_N
    - Flash_OE_N
    - SRBS3_N
    - SRSB0_N
    - MEMADDR19
    - dataOut[3:0]
    - dataIn[3:0]

    If your target is the M1AFS-EMBEDDED-KIT board, create the following additional top-level ports:

    - SRAM_BHE0_N
    - SRAM_BHE1_N
    - SRAM_BLE0_N
    - SRAM_BLE1_N
    - MEMADDR17
    - dataOut[3:0]
    - dataIn[3:0]

    If your target is the M1AFS-ADV-DEV-KIT board, create the following additional top-level ports:

    - SRAM_BHE0_N
    - SRAM_BHE1_N
    - SRAM_BLE0_N
    - SRAM_BLE1_N
    - SRAM_CE2
    - Flash_CE0
    - Flash_CE1
    - Flash_CE2
    - Flash_CE3
    - Flash_CE4
    - Flash_CE5
    - Flash_RST_N
    - Flash_OE_N
    - Flash_WE_N
    - Flash_VPEN
    - Flash_BYTE_1
    - Flash_BYTE_2
    - MEMADDR17

- MEMADDR18
- MEMADDR19
- dataOut[3:0]
- dataIn[3:0]

11. Right-click on the ADD[17:0] port of the Memory_0 block and select **Add Slice**. Set the slice range from 15 to 0. Right-click, again, on the ADDR[17:0] port of the Memory_0 block and select **Add Slice**. Set the slice range from 17 to 16.
12. Right-click on the dataOut[31:0] port of the Core8051s_sub_0 block and select **Add Slice**. Set the slice range from 3 to 0. Right-click, again, on the dataOut[31:0] port of the Core8051s_sub_0 block and select **Add Slice**. Set the slice range from 31 to 4.
13. Right-click on the dataIn[31:0] port of the Core8051s_sub_0 block and select **Add Slice**. Set the slice range from 3 to 0. Right-click, again, on the dataIn[31:0] port of the Core8051s_sub_0 block and select **Add Slice**. Set the slice range from 31 to 4.
14. A port can be tied low by right-clicking on the port and selecting **Tie Low**. More than one port can be tied low by holding down the CTRL key while clicking on each of the ports. After clicking the last of the ports, right-click and select **Tie Low**.
    - If the target is the M1AFS-DEV-KIT-SCS board, tie the signals listed in Table 2-11 low.

Table 2-11 · Tie Signals Low for M1AFS-DEV-KIT-SCS Board

Block	Port
Core8051s_sub_0	BREAKIN
	MEMBANK[3:0]
	dataIn[31:4]
	EXTIRQ
Memory_0	ADDR[17:16]
top level	MEM_ADDR19
	SRSB0_N
reset_delay_0	Shiften

- If the target is the M1AFS-EMBEDDED-KIT board, tie the signals listed in Table 2-12 low.

Table 2-12 · Tie Signals Low forM1AFS-EMBEDDED-KIT Board

Block	Port
Core8051s_sub_0	BREAKIN
	MEMBANK[3:0]
	EXTIRQ
	dataIn[31:4]
Memory_0	ADDR[17:16]
top level	MEM_ADDR17
	SRAM_BLE0_N
reset_delay_0	Shiften

- If the target is the M1AFS-ADV-DEV-KIT board, tie the signals listed in Table 2-13 low.

Table 2-13 · Tie Signals Low for M1AFS-ADV-DEV-KIT Board

Block	Port
Core8051s_sub_0	BREAKIN
	MEMBANK[3:0]
	EXTIRQ
	dataIn[31:4]
Memory_0	ADDR[17:16]
top level	MEM_ADDR17
	MEM_ADDR18
	MEM_ADDR19
	SRAM_BLE0_N
	Flash_RST_N
reset_delay_0	Shiften

15. A port can be tied high by right-clicking on the port and selecting **Tie High**. More than one port can be tied high by holding down the CTRL key while clicking on each of the ports. After clicking the last of the ports, right-click and select **Tie High**.
    - If the target is the M1AFS-DEV-KIT-SCS board, tie the signals listed in Table 2-14 high.

Table 2-14 · Tie Signals High for M1AFS-DEV-KIT-SCS Board

Block	Port
Core8051s_sub_0	MEMACKI
clk_pll_0	OADIVRST
	POWERDOWN
top level	Flash_HCE_N
	Flash_LCE_N
	SRBS1_N
	SRBS2_N
	SRBS3_N
	Flash_WE_N
	Flash_RST_N
	Flash_OE_N

- If the target is the M1AFS-EMBEDDED-KIT board, tie the signals listed in Table 2-15 high.

Table 2-15 · Tie Signals High for M1AFS-EMBEDDED-KIT Board

Block	Port
Core8051s_sub_0	MEMACKI
clk_pll_0	OADIVRST
	POWERDOWN
top level	SRAM_BHE0_N
	SRAM_BHE1_N
	SRAM_BLE1_N

- If the target is the M1AFS-ADV-DEV-KIT board, tie the signals listed in Table 2-16 high.

Table 2-16 · Tie Signals High for M1AFS-ADV-DEV-KIT Board

Block	Port
Core8051s_sub_0	MEMACKI
clk_pll_0	OADIVRST
	POWERDOWN
top level	SRAM_BHE0_N
	SRAM_BHE1_N
	SRAM_BLE1_N
	SRAM_CE2
	Flash_BYTE_1
	Flash_BYTE_2
	Flash_WE_N
	Flash_OE_N
	Flash_CE0
	Flash_CE1
	Flash_CE2
	Flash_CE3
	Flash_CE4
	Flash_CE5
	Flash_VPEN

16. In this step you will make various connections. To connect two (or more) ports, hold the CTRL key and click on the first port. The port name will be highlighted. Locate the second port to be included in the net. Hold the CTRL key and click on this port. The port name will be highlighted. Repeat for as many ports as will be added to the net. Right-click on the last port added and select **Connect**. This will connect these ports together. Make the port connections listed in Table 2-17.

Table 2-17 · Port Connections

Connection	Block	Port
1	Memory_0	ADDR[15:0]
	Core8051s_sub_0	MEMADDR[15:0]
2	Memory_0	PSWR
	Core8051s_sub_0	DBGMEMPSWR
3	Memory_0	PSRD
	Core8051s_sub_0	MEMPSRD
4	Memory_0	MEMRD
	Core8051s_sub_0	MEMRD
5	Memory_0	MEMWR
	Core8051s_sub_0	MEMWR
6	Memory_0	RD[7:0]
	Core8051s_sub_0	MEMDATAI[7:0]
7	Memory_0	WD[7:0]
	Core8051s_sub_0	MEMDATAO[7:0]
8	Memory_0	NVM_BUSY_N
	Core8051s_sub_0	MEMPSACKI
9	Memory_0	CLK
	Core8051s_sub_0	SYSCLK
	clk_pll_0	GLA
	reset_delay_0	Clock
10	top level	EXTCLK
	clk_pll_0	CLKA
11	Core8051s_sub_0	dataOut[3:0]
	top level	dataOut[3:0]
12	Core8051s_sub_0	dataIn[3:0]
	top level	dataIn[3:0]
13	reset_delay_0	Shiftout
	Core8051s_sub_0	NSYSRESET
14	reset_delay_0	Shiftin
	clk_pll_0	LOCK

17. Select **Check Design Rules** from the SmartDesign tab in the Libero IDE main menu. Verify that there are no errors. Right-click in the root canvas and select **Generate Design**.

3 Verifying the Design

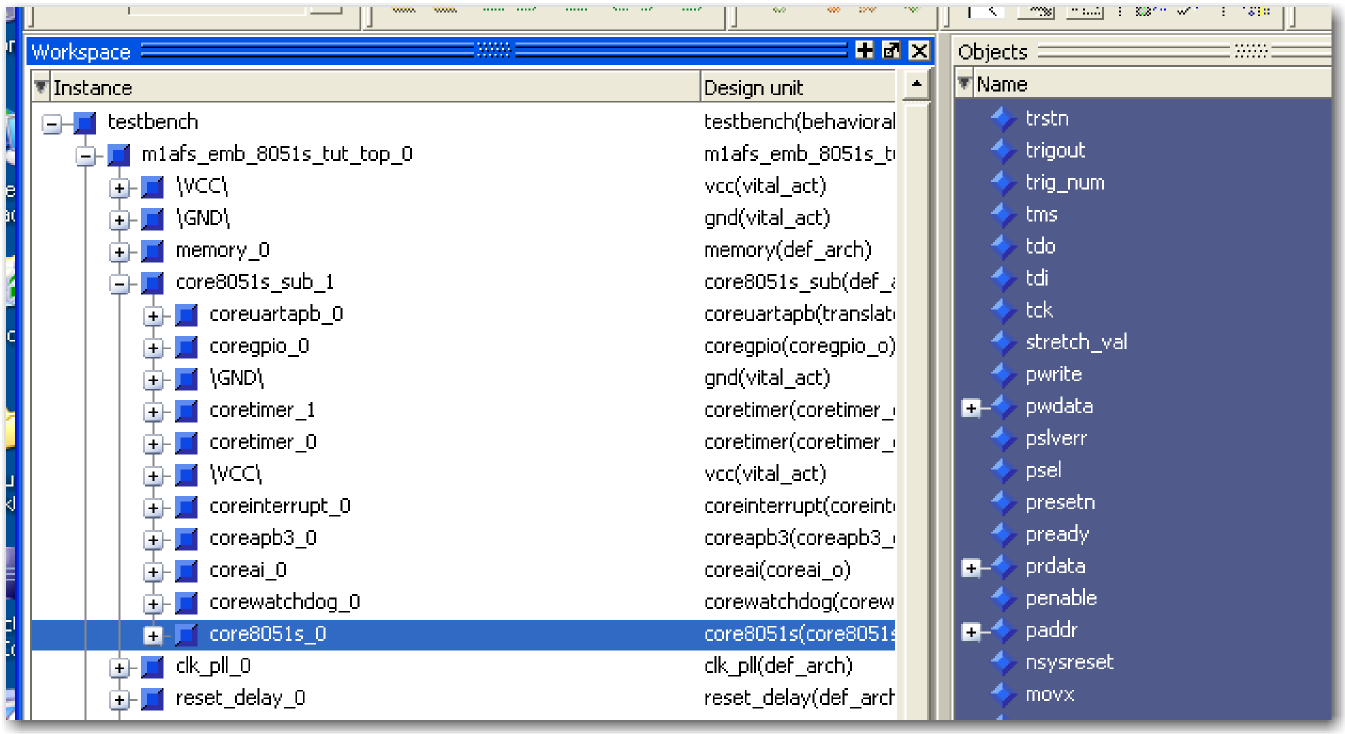
Pre-Synthesis Simulation in Model*Sim*

1. In the Design Explorer box, right-click on the top-level entity (*_TUT_TOP) and select **Organize Stimulus**. In the right-side box inside the Organize Stimulus window, verify that it contains a file, testbench.vhd, whose origin is the *_TUT_TOP entity. If not present, find the file in the list in left-side box and click **Add**.
2. From the main Project tab, select **Settings**. In the Project Settings box, select the Simulation tab. Highlight COREUARTAPB_… in the left-side box under the Libraries category. In the value box for the Library command, click **Delete and recompile**.

   Highlight **Waveforms** in the left-side box. Click the value box for the **Log all signals in the design** item.

   Highlight **DO File**. Set the Simulation runtime item to 1 ms.

   Click **OK**.
3. Right-click the Simulation button in the Project Flow window of Libero IDE and select **Run Pre-Synthesis Simulation**.
4. After the simulator has loaded, in the Instance window, locate and click on the ***_top_0 entity**. Expand the top entity. Scroll, find, and expand the core8051s_sub_0 entity. Click on the **core8051s_0 entity**.
5. In the objects window, click and add the following signals to the simulation: clk, nsysreset, mempsacki, mempsrd, memaddr, and memdati (Figure 3-1).



Instance	Design unit
testbench	testbench(behavioral
m1afs_emb_8051s_tut_top_0	m1afs_emb_8051s_t
\VCC\	vcc(vital_act)
\GND\	gnd(vital_act)
memory_0	memory(def_arch)
core8051s_sub_1	core8051s_sub(def_
coreuartapb_0	coreuartapb(translat
coregpio_0	coregpio(coregpio_o)
\GND\	gnd(vital_act)
coretimer_1	coretimer(coretimer_
coretimer_0	coretimer(coretimer_
\VCC\	vcc(vital_act)
coreinterrupt_0	coreinterrupt(corein
coreapb3_0	coreapb3(coreapb3_
coreai_0	coreai(coreai_o)
corewatchdog_0	corewatchdog(corew
core8051s_0	core8051s(core8051
clk_pll_0	clk_pll(def_arch)
reset_delay_0	reset_delay(def_arc

Name
trstn
trigout
trig_num
tms
tdo
tdi
tck
stretch_val
pwrite
pwdata
pslverr
psel
presetn
pready
prdata
penable
paddr
nsysreset
movx

Figure 3-1 · Selecting Core8051s Signals

6. Click on the Zoom Full icon (filled magnifying glass). You can now use the zoom and scroll functions to observe the processor execute code in the waveform window.

7. Undock the waveform window by clicking on the undock button in the waveform window. This is the icon with the box and upper right arrow. In the resulting waveform window (Figure 3-3), click **File** > **Save Format**. Click **Waveform formats** in the Save Contents box. Click **OK** (click **Yes** if prompted to overwrite).

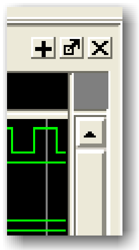


Figure 3-2 · Undocking the Simulation Window

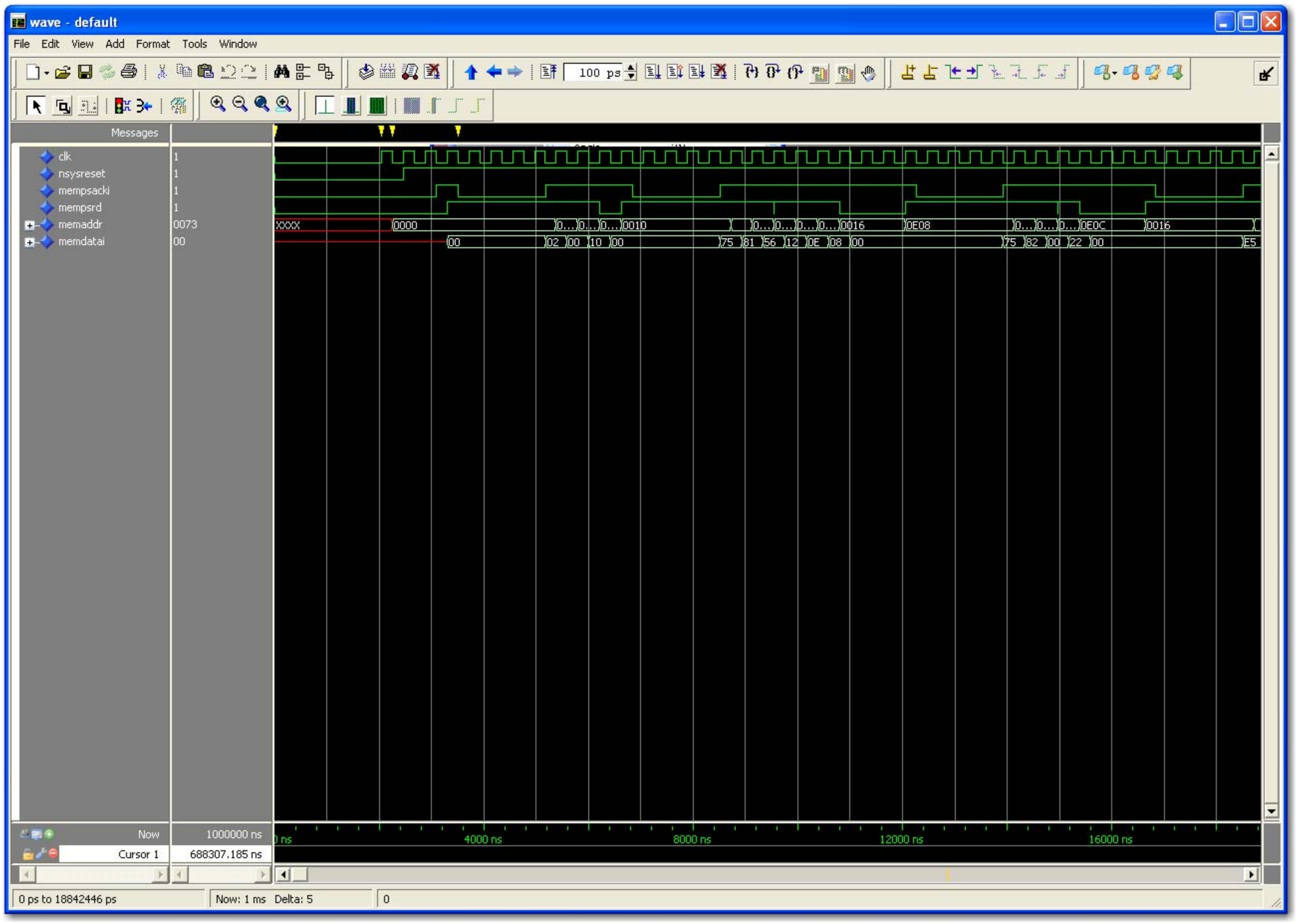


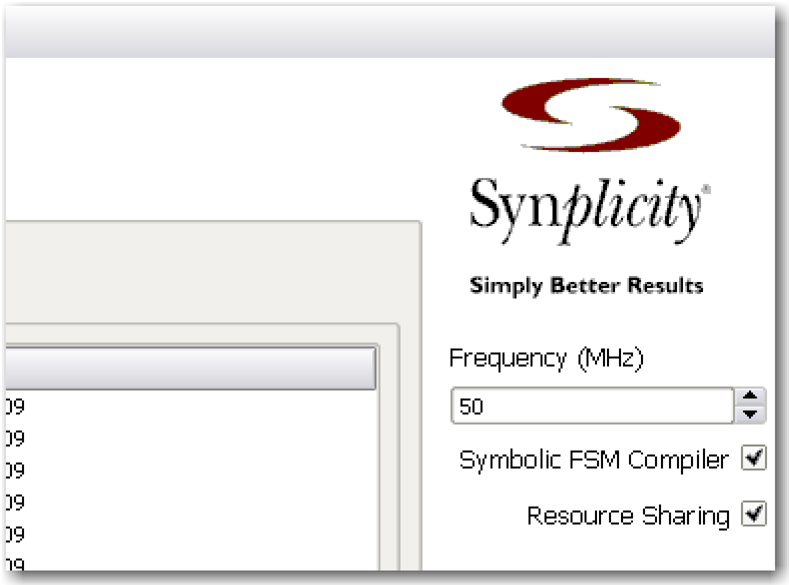
Figure 3-3 · Simulation of CPU Instruction Fetches

Note: In Figure 3-3, the path names for the signals have been suppressed by setting the Display Signal Path to one element.

Implementing the Design

Step 1 – Synthesizing the Design

1. Click on the **Project Flow** tab in Libero IDE. Click on the **Synthesis** box.



Frequency (MHz): 50
Symbolic FSM Compiler ✓
Resource Sharing ✓

Figure 4-1 · Setting the Frequency in Synplify

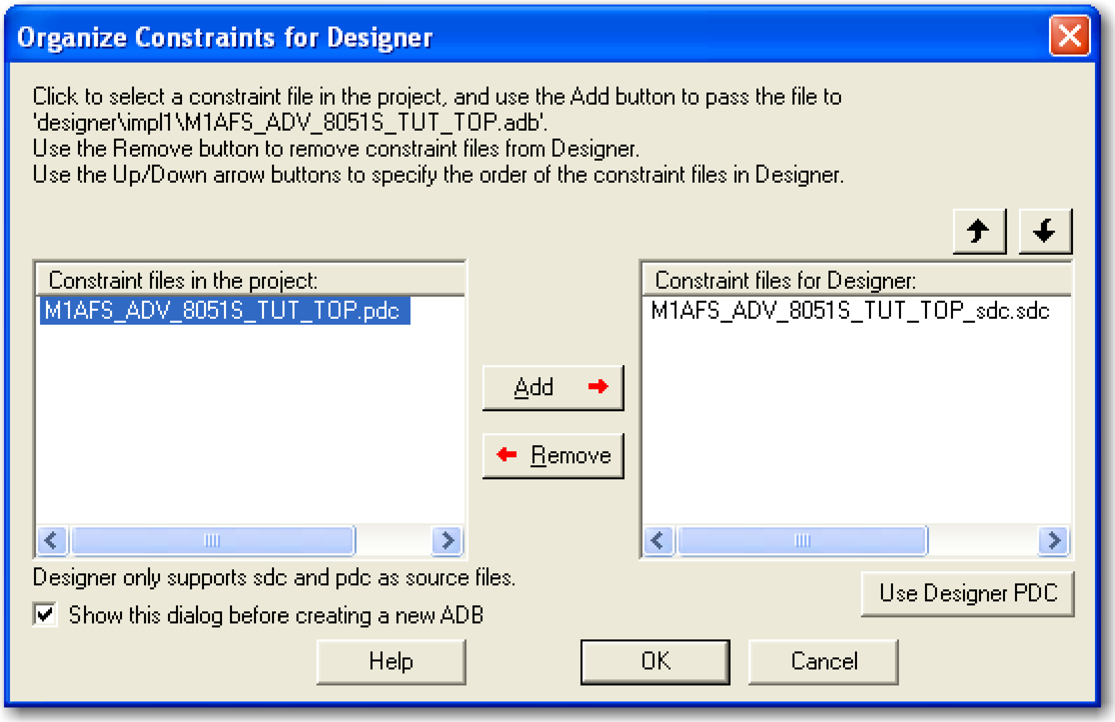
If your target is the M1AFS-EMBEDDED-KIT board or the M1AFS-ADV-DEV-KIT board, set the frequency to 50 MHz.

If your target is the M1AFS-DEV-KIT-SCS board, set the frequency to 48 MHz.

2. Click the **Run** button after Synplify opens. Close Synplify when it completes.

Step 2 – Performing Place-and-Route, Pin Assignments

1. Click on the Designer button. In the Organize Constraints for Designer box, click on the *.pdc file in the project files (left) box. Click the **Add** button. The *.pdc file will appear in the Designer files (right) box. Click **OK**. Normally, when you click on the Designer button for the first time with a new project, the Organize Constraints for Designer box will automatically launch.


Organize Constraints for Designer

Click to select a constraint file in the project, and use the Add button to pass the file to 'designer\impl1\M1AFS_ADV_8051S_TUT_TOP.adb'.
Use the Remove button to remove constraint files from Designer.
Use the Up/Down arrow buttons to specify the order of the constraint files in Designer.

Constraint files in the project:
M1AFS_ADV_8051S_TUT_TOP.pdc

Constraint files for Designer:
M1AFS_ADV_8051S_TUT_TOP_sdc.sdc

Designer only supports sdc and pdc as source files.

☑ Show this dialog before creating a new ADB

Figure 4-2 · Organize Constraints for Designer

2. The Device Selection Wizard will open. Choose the Fusion die that matches your target board (refer to "Step 1 – Create the Libero IDE Project" on page 9).

   Choose the 484 FBGA package. Select Std speed range. Click **Next**.

   The Device Selection Wizard - Variations box will open. Accept the defaults and click **Next**.

   The Device Selection Wizard - Operating Conditions box will open. Click **Finish**.
3. In Designer, click the **Compile** button. Answer **OK**. Wait for the Compile button to turn green.
4. The **I/O Attribute Editor** in the MultiView Navigator section of Designer can be used to assign signals to physical device pins. An alternative is to import a *.pdc file that already contains pin assignments for the design. This was done earlier in our project. This file was then selected for Designer to use when we organized the constraint files, as shown in Figure 4-2.
5. Click on the Constraints Editor button in the Designer GUI. In the "Constraints Editor" box, click on the frequency value for TCK and change it to 20.

   Note: The constraints Editor box opened inside of the SmartTime box.
6. Select **Actions** > **Constraint** > **Clock** from the SmartTime Constraint Editor menu to open the Create Clock Constraint Dialog box. Select the clock beginning with the name Core8051S_sub from the pull-down menu in the Create Clock Constraint dialog box. Enter 10 MHz. Click OK.

7. Select **Actions** > **Constraint** > **Clock** from the SmartTime Constraint Editor menu to open the Create Clock Constraint Dialog box. Select the EXTCLK clock beginning from the pull-down menu in the Create Clock Constraint dialog box.

   If your target is the M1AFS-DEV-KIT-SCS board, enter 48 MHz; otherwise enter 50 MHz. Click **OK**.
8. Click on the **Layout** button. Answer **OK**. Wait for the Layout button to turn green.
9. Click on the **Programming file** button. Click the **Finish** button in the FlashPoint – Programming File Generator box. Click **Finish**. Click **Generate** in the Generate Programming Files box. Answer **Yes** if prompted to replace files. Close Designer after the Programming Files button turns green. Answer **Yes** if prompted to save changes.

   The generated file will include the FPGA design plus the firmware for the 64KB NVM block. Later, FlashPro can be used to update the contents of the NVM block.

Step 3 – Programming the FPGA Using FlashPro

1. In this step we will connect the PC to the FlashPro programmer.

   If your target is the M1AFS-DEV-KIT-SCS board:
   - Connect a USB cable from a USB port on your PC to the USB PROG connector on the board. This board has the FlashPro programmer and debugger integrated into the board design.

   If your target is the M1AFS-EMBEDDED-KIT board:
   - Plug the LC Programmer board into connector J1 of the M1AFS-EMBEDDED-KIT board.
   - Connect a USB cable from a USB port on your PC to the USB connector on the LC Programmer board.

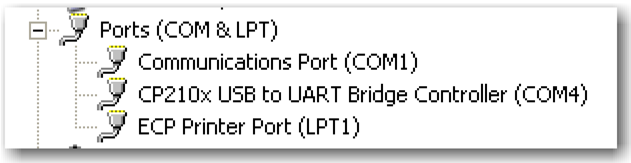
   If your target is the M1AFS-ADV-DEV-KIT board:
   - Plug the 10-pin ribbon connector from a FlashPro3 unit into connector J6 of the M1AFS-ADV-DEV-KIT board.
   - Connect a USB cable from a USB port on your PC to the USB connector on the FlashPro3 unit.
2. Connect power to the power connector of the board.
3. Install the USB drivers for FlashPro3 if not previously installed.
4. Click on the **Programming** box in the Project Flow tab.
5. Maximize the FlashPro dialog box.
6. Verify that a programmer has been identified in the Programmer Name list. If no programmer name is shown, click the **Refresh/Rescan for Programmers** button.
7. FlashPro will read the PDB file created by Designer and obtain the path for the *.efc and *.ihx files for the NVM block. These can be verified by clicking the **PDB Configuration…** button. Click the **Modify** button in the FlashPoint – Programming File Generator box. Click the **Import Configuration File…** button to check the *.efc file. Click the **Import content…** button to verify the *.ihx file. The **Import content…** button can be used to update the code for the NVM block when making code changes.
8. Click the **Program** button. The FlashPro Programmer will program both the FPGA array and the Embedded Flash Memory Module. Do not disrupt the power during the programming operation, as it can damage the FPGA. Close FlashPro. Answer **Yes** if prompted to save file(s).

Step 4 – Running the Design

1. If your target is the M1AFS-DEV-KIT-SCS board:
   - Disconnect the USB cable from the USB PROG connector on the board.
   - Install jumpers on JP1, JP2, JP3, JP4 (1-4), JP5, JP7 (2-3), JP8 (2-3), JP9, JP10 (1-2), and JP13. All other jumpers on this board are not installed.

   If your target is the M1AFS-EMBEDDED-KIT board:
   - Unplug the LC Programmer board from connector J1 of the M1AFS-EMBEDDED-KIT board.

   If your target is the M1AFS-ADV-DEV-KIT board:
   - Unplug the 10-pin ribbon connector of the FlashPro3 unit from connector J6 of the M1AFS-ADV-DEV-KIT board.
2. Install the drivers for the CP2102 USB serial port if not already installed on your computer. You can download the drivers from http://www.actel.com/products/hardware/devkits_boards/fusion_embedded.aspx
3. Connect a USB cable between the USB serial port connector on your target board and your PC.
4. Determine the COM port assigned to the USB interface. Open the Windows Control Panel and double-click the **System** icon. Click on the **Hardware** tab. Click the **Device Manager** button. Expand the **Ports (Com & LPT)** item in Device Manager. Look for the CP2102 USB to UART Bridge Controller in the list of ports. The COM port in parentheses next to it is the COM port assigned to this device.



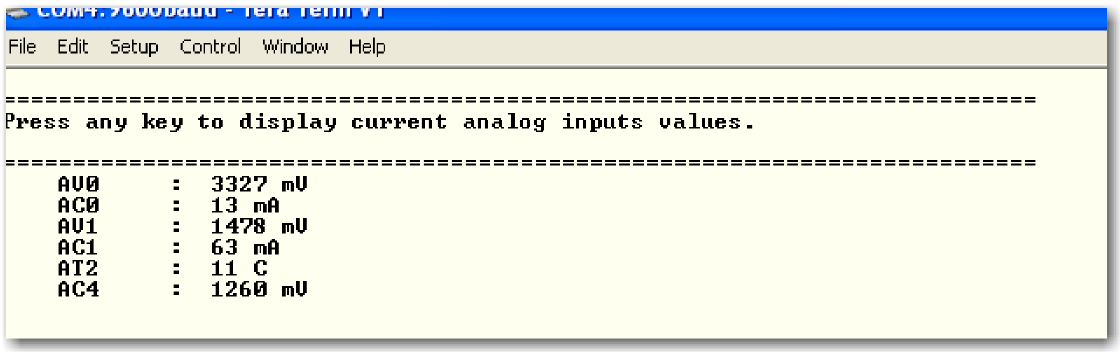
```
Ports (COM & LPT)
    Communications Port (COM1)
    CP210x USB to UART Bridge Controller (COM4)
    ECP Printer Port (LPT1)
```

Figure 4-3 · Determining the USB Serial Port's COM Port

5. Open a terminal emulator program and set the communications parameters to 9600 baud, 8 data bit, no parity, and no handshaking. Set the COM port to match the USB serial port of the board.
6. If your target is the M1AFS-DEV-KIT-SCS board set position 10 of switch SW1 to the OFF position.

   If your target is the M1AFS-EMBEDDED-KIT board or the M1AFS-ADV-DEV-KIT board remove the jumper, if any, between pins 1 and 2 of connector J5. When the switch is off or this jumper is removed, the NVM block occupies the microcontroller's code memory space. When switch SW1 is in the ON position or the jumper is in place, external SRAM is used for code storage during debugging.
7. Cycle power to the board.

8. Momentarily push the reset button on the board. The terminal program should show a sign-on message. Click any key on your keyboard to send a character to the board. The board should respond by printing voltage, current, and temperature measurements on the board (Figure 4-4).



```
==========================================================================
Press any key to display current analog inputs values.

==========================================================================
    AV0      :  3327 mV
    AC0      :  13 mA
    AV1      :  1478 mV
    AC1      :  63 mA
    AT2      :  11 C
    AC4      :  1260 mV
```

Figure 4-4 · Application Running

Summary

In this tutorial you have created a Core8051s-based microcontroller system on an Actel Fusion device. This design includes on-chip flash code memory, on-chip (logically external) data memory, and an interface to off-chip SRAM that is used for both code and data memory.

This design can serve as the basic starting point for other Core8051s designs. Only the CoreAI function and the on-chip flash memory are specific to the Fusion family.

If these are removed, this design can be ported to other families such as ProASIC®3 and IGLOO.® The specific details are outside of the scope of this document. In general, this can be accomplished by creating a Libero IDE project for the new target family. Then import the *.cxf file for each of the three components: *.TOP, Core8051S_sub, and Memory. Regenerate these components in SmartDesign.

Product Support

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

Customer Service

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

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

Actel Customer Technical Support Center

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

Actel Technical Support

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

Website

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

Contacting the Customer Technical Support Center

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

Email

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

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

Phone

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

650.318.4460
800.262.1060

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

Index

Actel is the leader in low-power and mixed-signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at www.actel.com.

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 • USA*
Phone 650.318.4200 • Fax 650.318.4600 • Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd**.** *• River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

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

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

50200155-1/3.09