Cortex™-M1 FPGA Development Kit

Altera Cyclone III Edition Version 1.1

Example System Tutorial
Cortex-M1 FPGA Development Kit
Example System Tutorial

Copyright © 2008 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 September 2008</td>
<td>A</td>
<td>Non-Confidential</td>
<td>First release for version 1.1</td>
</tr>
</tbody>
</table>

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

Altera is a trademark and service mark of Altera Corporation in the United States and other countries. Altera products contain intellectual property of Altera Corporation and are protected by copyright laws and one or more U.S. and foreign patents and patent applications.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is final, that is for a developed product.
Web Address

http://www.arm.com
Contents

Cortex-M1 FPGA Development Kit Example System Tutorial

Preface

About this tutorial ................................................................. xii
Feedback ..................................................................................... xvi

Chapter 1 Introduction

1.1 About the tutorial and example SOPC system ................................. 1-2
1.2 SOPC components and implementation ............................................ 1-3
1.3 Example SOPC system configuration ................................................ 1-8

Chapter 2 Getting Started

2.1 About getting started ........................................................................ 2-2
2.2 Loading the FPGA using the Quartus II Programmer .......................... 2-3
2.3 Software execution ............................................................................ 2-7
2.4 Interacting with the example software ................................................ 2-8
2.5 Viewing the example software source code (optional) ......................... 2-11
2.6 Simulation environment ..................................................................... 2-13

Chapter 3 Demonstration Software and Example SOPC System

3.1 About the demonstration software .................................................... 3-2
3.2 Source files ...................................................................................... 3-3
Chapter 4 Debugging Example SOPC System
4.1 Assumptions ............................................................................................... 4-2
4.2 RealView MDK tools setup ......................................................................... 4-3
4.3 Emulation on the Cyclone III Starter Board using the MDK tools .......... 4-5
4.4 Loading the Cyclone III Starter Board and starting the Cortex-M1 terminal ...... 4-11
4.5 Debugging the demo ................................................................................ 4-12

Chapter 5 Simulating Example SOPC System
5.1 Generated SOPC HDL testbench ............................................................... 5-3
5.2 Simulation software description ................................................................. 5-4
5.3 Generating the ITCM image file ................................................................. 5-5
5.4 Simulating the example SOPC system design ........................................... 5-6

Chapter 6 RTX Real Time Operating System Example
6.1 About the RTX real time operating system example .................................. 6-2
6.2 Examining the project ................................................................................ 6-3
6.3 Program description ................................................................................... 6-4
6.4 Hex file generation .................................................................................... 6-5

Chapter 7 Software Considerations and Customization
7.1 Memory resources ...................................................................................... 7-2
7.2 Flash-FPGA load and code space considerations ..................................... 7-3
7.3 Preloading Altera RAMs with Cortex-M1 instructions ............................... 7-4
7.4 Example SOPC system clocking ............................................................... 7-6

Chapter 8 Configuring Example SOPC System
8.1 Using SOPC Builder to configure and generate the example SOPC system .... 8-2
List of Tables
Cortex-M1 FPGA Development Kit Example
System Tutorial

Change History ........................................................................................................ ii

Table 3-1 Files located in FPGA directory ................................................................ 3-3
List of Figures

Cortex-M1 FPGA Development Kit Example System Tutorial

Figure 1-1  Example SOPC system block diagram ................................................................. 1-4
Figure 1-2  Example SOPC system with Cyclone III Starter Board ........................................ 1-5
Figure 1-3  Example SOPC system configuration ................................................................. 1-8
Figure 2-1  CortexM1_ExampleDesign project selection ....................................................... 2-3
Figure 2-2  Hardware setup .................................................................................................. 2-4
Figure 2-3  Auto Detect button ............................................................................................. 2-4
Figure 2-4  SOF options ...................................................................................................... 2-5
Figure 2-5  Programming complete ....................................................................................... 2-5
Figure 2-6  Cortex-M1 Terminal window .............................................................................. 2-8
Figure 2-7  Cortex-M1 Terminal window awaiting switch press ........................................... 2-9
Figure 2-8  Cortex-M1 Terminal window with test menu ..................................................... 2-10
Figure 2-9  uVision main source listing .............................................................................. 2-12
Figure 2-10 CortexM1_ExampleDesign.qpf ....................................................................... 2-14
Figure 2-11 Generate the example SOPC system ................................................................. 2-14
Figure 4-1  Select project file .............................................................................................. 4-3
Figure 4-2  RealView MDK IDE .......................................................................................... 4-4
Figure 4-3  Emulator target tab .......................................................................................... 4-6
Figure 4-4  Emulator output tab ......................................................................................... 4-7
Figure 4-5  Emulator linker tab ........................................................................................... 4-8
Figure 4-6  Emulator debug tab .......................................................................................... 4-9
Figure 4-7  Target driver setup ............................................................................................ 4-10
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-8</td>
<td>Cortex-M1 Terminal application window</td>
<td>4-11</td>
</tr>
<tr>
<td>4-9</td>
<td>RealView IDE with break at main()</td>
<td>4-13</td>
</tr>
<tr>
<td>4-10</td>
<td>Program break at while(1) loop</td>
<td>4-14</td>
</tr>
<tr>
<td>4-11</td>
<td>Cortex-M1 Terminal initial test menu window</td>
<td>4-15</td>
</tr>
<tr>
<td>5-1</td>
<td>Run Simulator option and button</td>
<td>5-6</td>
</tr>
<tr>
<td>5-2</td>
<td>ModelSim-Altera window</td>
<td>5-7</td>
</tr>
<tr>
<td>5-3</td>
<td>ModelSim-Altera wave view window</td>
<td>5-9</td>
</tr>
<tr>
<td>6-1</td>
<td>Linking the RTX library to a project</td>
<td>6-3</td>
</tr>
<tr>
<td>7-1</td>
<td>Options for Target Emulator</td>
<td>7-5</td>
</tr>
<tr>
<td>8-1</td>
<td>New Project Wizard Introduction window</td>
<td>8-3</td>
</tr>
<tr>
<td>8-2</td>
<td>New Project Wizard Directory window</td>
<td>8-4</td>
</tr>
<tr>
<td>8-3</td>
<td>New Project Wizard Add Files window</td>
<td>8-5</td>
</tr>
<tr>
<td>8-4</td>
<td>New Project Wizard Family &amp; Device Settings window</td>
<td>8-6</td>
</tr>
<tr>
<td>8-5</td>
<td>New Project Wizard EDA Tool Settings window</td>
<td>8-7</td>
</tr>
<tr>
<td>8-6</td>
<td>New Project Wizard Summary window</td>
<td>8-8</td>
</tr>
<tr>
<td>8-7</td>
<td>SOPC Builder Create New System dialog</td>
<td>8-9</td>
</tr>
<tr>
<td>8-8</td>
<td>Initial SOPC Builder screen</td>
<td>8-10</td>
</tr>
<tr>
<td>8-9</td>
<td>ARM Cortex-M1 Processor settings window</td>
<td>8-11</td>
</tr>
<tr>
<td>8-10</td>
<td>Processor instantiation in SOPC Builder</td>
<td>8-12</td>
</tr>
<tr>
<td>8-11</td>
<td>LED IF component parameter settings dialog</td>
<td>8-13</td>
</tr>
<tr>
<td>8-12</td>
<td>LED_IF component instantiated in SOPC Builder</td>
<td>8-14</td>
</tr>
<tr>
<td>8-13</td>
<td>Switch_IF component instantiated in SOPC Builder</td>
<td>8-15</td>
</tr>
<tr>
<td>8-14</td>
<td>Timer component instantiated in SOPC Builder</td>
<td>8-16</td>
</tr>
<tr>
<td>8-15</td>
<td>JTAG_UART component instantiated in SOPC Builder</td>
<td>8-17</td>
</tr>
<tr>
<td>8-16</td>
<td>Launch PLL MegaWizard</td>
<td>8-18</td>
</tr>
<tr>
<td>8-17</td>
<td>PLL parameter settings</td>
<td>8-19</td>
</tr>
<tr>
<td>8-18</td>
<td>PLL Lock output setting</td>
<td>8-20</td>
</tr>
<tr>
<td>8-19</td>
<td>PLL Output Clocks setting</td>
<td>8-21</td>
</tr>
<tr>
<td>8-20</td>
<td>PLL instantiated in SOPC Builder</td>
<td>8-22</td>
</tr>
<tr>
<td>8-21</td>
<td>System ID instantiated in SOPC Builder</td>
<td>8-23</td>
</tr>
<tr>
<td>8-22</td>
<td>Flash settings</td>
<td>8-24</td>
</tr>
<tr>
<td>8-23</td>
<td>SSRAM settings dialog</td>
<td>8-25</td>
</tr>
<tr>
<td>8-24</td>
<td>Tristate Bridge settings dialog</td>
<td>8-26</td>
</tr>
<tr>
<td>8-25</td>
<td>Tristate Bridge connections to SSRAM and flash</td>
<td>8-27</td>
</tr>
<tr>
<td>8-26</td>
<td>Tristate Bridge connection settings dialog</td>
<td>8-28</td>
</tr>
<tr>
<td>8-27</td>
<td>Bottom of SOPC Builder screen</td>
<td>8-28</td>
</tr>
<tr>
<td>8-28</td>
<td>Quartus II Project File settings dialog</td>
<td>8-31</td>
</tr>
<tr>
<td>8-29</td>
<td>Quartus II Import Assignments dialog</td>
<td>8-32</td>
</tr>
<tr>
<td>8-30</td>
<td>Quartus II Device and Pin Options dialog</td>
<td>8-33</td>
</tr>
</tbody>
</table>
Preface

This preface introduces the Cortex-M1 Example System Tutorial. It contains the following sections:

- *About this tutorial* on page xii
- *Feedback* on page xvi.
About this tutorial

This is the example system tutorial for the Cortex-M1 processor in an FPGA Development Kit targeted at Altera Cyclone III devices using the Quartus II tools including the Altera System On a Programmable Chip (SOPC) Builder.

Note

In this tutorial, the term processor means the processor for the Cortex-M1 FPGA Development Kit for Altera Cyclone III, unless the context indicates otherwise.

Intended audience

This tutorial is written for:

- system designers, system integrators, and verification engineers who want to implement the processor in a Field-Programmable Gate Array (FPGA) using the Altera Quartus II tools.
- software developers who want to use the processor in a FPGA.

Using this tutorial

This tutorial is organized into the following chapters:

Chapter 1 Introduction

Read this chapter for a description of the components of the processor, and about the processor instruction set.

Chapter 2 Getting Started

Read this chapter for a description of how to get started with the Cortex-M1 FPGA Development Kit. This chapter describes the example SOPC system design and the Cyclone III Starter Board.

Chapter 3 Demonstration Software and Example SOPC System

Read this chapter for a description of the example SOPC system demonstration (FPGA Code project) software.

Chapter 4 Debugging Example SOPC System

Read this chapter for a description of how to use the RealView MDK tools to debug software targeted at the example SOPC system running on the Cyclone III Starter Board.
Chapter 5 Simulating Example SOPC System
Read this chapter for a description of how to simulate the example SOPC system using the Altera version of ModelSim.

Chapter 6 RTX Real Time Operating System Example
Read this chapter for a description of a real time operating system example using RTX.

Chapter 7 Software Considerations and Customization
Read this chapter for a description of how you can customize the example SOPC system and Software.

Chapter 8 Configuring Example SOPC System
Read this chapter for a description of how to use the Quartus II v8.0 SOPC Builder tool to configure the example SOPC system.

Conventions

Conventions that this tutorial can use are described in:

• Typographical
• Signals on page xiv
• Numbering on page xiv.

Typographical
The typographical conventions are:

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

monospace Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

monospace italic Denotes arguments to monospace text where the argument is to be replaced by a specific value.
monospace bold Denotes language keywords when used outside example code.

< and > Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:
MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>

Signals

The signal conventions are:

**Signal level** The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals
- LOW for active-LOW signals.

**Lower-case n** At the start or end of a signal name denotes an active-LOW signal.

**Prefix H** Denotes Advanced High-performance Bus (AHB) signals.

**Prefix P** Denotes Advanced Peripheral Bus (APB) signals.

Numbering

The numbering convention is:

<size in bits><base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of 0x3F. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Additional reading

This section lists publications by ARM and by third parties. See:

- http://infocenter.arm.com/help/index.jsp for access to ARM documentation
- http://www.altera.com for access to Altera documentation.
ARM publications

This tutorial contains information that is specific to the processor for use with the Cortex-M1 FPGA Development Kit Altera Cyclone III Edition. See the following documents for other relevant information:

- *Application Note 213: Cortex-M1 TCM Initialization in the ARM Cortex-M1 FPGA Development Kit Altera Edition* (ARM DAI 0213)
- *Application Note 214: Flash programming in the ARM Cortex-M1 FPGA Development Kit Altera Edition* (ARM DAI 0214)

Other publications

This section lists relevant documents published by third parties:

- *Altera Corporation, Cyclone III FPGA Starter Kit Quick Start Guide*
- *IEEE Standard, Test Access Port and Boundary-Scan Architecture specification 1149.1-1990 (JTAG).*
Feedback


Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

• the product name
• a concise explanation.

Feedback on this tutorial

If you have any comments on this tutorial, send an e-mail to errata@arm.com. Give:

• the title
• the number
• the relevant page number(s) to which your comments apply
• a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.
Chapter 1

Introduction

This chapter introduces the Cortex-M1 FPGA Development Kit Example System Tutorial. It contains the following sections:

- About the tutorial and example SOPC system on page 1-2
- SOPC components and implementation on page 1-3
- Example SOPC system configuration on page 1-8.
1.1 About the tutorial and example SOPC system

The example System-on-Chip (SoC) design used in this tutorial is built completely using Altera's System On a Programmable Chip (SOPC) Builder software, located in Altera's Quartus II v8.0 toolset. While the design is simple, this tutorial demonstrates how to build a complete SoC for an Altera FPGA using the Cortex-M1 processor and available Altera SOPC peripheral and bus components.

The processor is intended for deeply embedded applications that require a small processor integrated into an FPGA. This tutorial is consistent with typical embedded FPGA processor designs that use a Cortex-M1 processor.

The example SOPC system contains many of the components used in popular microcontrollers. The processor and its subsystem contain the components, buses, and memory map required to run deeply embedded software kernels or Real Time Operating Systems (RTOS) such as RTX from Keil.

This system also contains interrupt (pre-emption) capability by utilizing the processor’s internal Nested Vectored Interrupt Controller (NVIC) as described in the Cortex-M1 FPGA Development Kit Altera Cyclone III Edition Cortex-M1 User Guide.

RTOS support is available by utilizing the NVIC in conjunction with the processor’s internal SYSTICK timer.

The system is built around Altera's System Interconnect Fabric bus structure. For more information about the System Interconnect Fabric, see the Altera website at:

http://www.altera.com

——— Note ————

- The example subsystem is provided for your internal use in accordance with the terms of the accompanying ARM end user license agreement for the ARM Cortex-M1 FPGA Development Kit, including the exclusions and limits of liability described therein. It is your sole responsibility to satisfy yourself of its suitability for use in any product or design, and to complete, validate, and test such design for your intended target application.

- If you are using the evaluation version of the ARM Cortex-M1 FPGA Development kit, you are not licensed to use the example subsystem or ARM Cortex-M1 processor to create files for a released product. You must purchase a separate license if you want to use the ARM Cortex-M1 processor or design files in the production of any commercial products.
1.2 SOPC components and implementation

This section describes the components and implementation of the example SOPC system.

The components of the system are:

- Cortex-M1 processor core with Debug support
- Instruction Tightly Coupled Memory (ITCM) and Data Tightly Coupled Memory (DTCM)
- Nested Vectored Interrupt Controller (NVIC)
- internal system timer, SYSTICK
- System Interconnect Fabric
- Timer
- Switch interface PIO
- LED interface PIO
- JTAG UART
- Tristate Bridge for external memories
- System ID Peripheral
- external SSRAM PLL clock.

Figure 1-1 on page 1-4 shows the structure of the example SOPC system.
Figure 1-1 Example SOPC system block diagram

Figure 1-2 on page 1-5 shows a block diagram of how the system is used inside the FPGA on the Altera Cyclone III Starter Board.
Figure 1-2 Example SOPC system with Cyclone III Starter Board
1.2.1 Example SOPC system

The example SOPC system has the following features:

- System Interconnect Fabric
- Cortex-M1 processor with internal SRAM (ITCM and DTCM) used for boot and application code and data
- external SSRAM and flash used for software test program execution
- timer for testing interrupts, and useful for software kernels or RTOS
- switch interface and LED interface for user input/output
- JTAG UART used to connect to a PC running the Cortex-M1 Terminal application
- system ID component, often used for identifying the version of the design in the FPGA.

1.2.2 Cortex-M1 processor SOPC Builder component

The processor is implemented with an Avalon memory-mapped bus master interface to connect to the System Interconnect Fabric.

The processor component provides debug and non-debug options, selected using the SOPC builder tools.

See the Cortex-M1 FPGA Development Kit Altera Cyclone III Edition Cortex-M1 User Guide for more details about the processor and its configuration options.

1.2.3 System Interconnect Fabric

The System Interconnect Fabric is the bus that connects the processor core to all of the peripherals and cores. The fabric is automatically generated using the SOPC Builder IDE tools. The fabric can be configured as a multi-matrix, multi-master bus for complex systems. However, for the example SOPC system design, the System Interconnect Fabric is configured as a simple single master (Cortex-M1) multiple slave bus. The address decoding for the system peripherals and cores is built into the System Interconnect Fabric. The fabric also provides the data bus multiplexing back to the processor.

1.2.4 Timer and Cortex-M1 Nested Vectored Interrupt Controller (NVIC)

The timer is an interval timer that can be used in a system running a kernel (scheduler) or an RTOS. The timer is typically used with an interrupt controller as a timebase that pre-empts the processor to switch context or tasks.
The tutorial uses the SOPC timer component to demonstrate how to connect interrupt sources to the Cortex-M1 component. The timer is a simple interrupt source used to show how software handles hardware interrupts, using the processor’s internal NVIC.

Note
Software can also use the processor’s internal SYSTICK timer.

1.2.5 JTAG UART

The USB connection on the Altera Cyclone III Starter Board is used by the Quartus II programmer tool to download the FPGA logic and is also used as a communications port to connect to a PC through the JTAG UART component. The tutorial shows how to use the USB connection for both applications.

The FPGA design must instantiate the JTAG UART component to use the USB connection to communicate with the Cortex-M1 Terminal application. The JTAG UART component is instantiated as the USB_UART in the example SOPC system.

1.2.6 Switch and LED interfaces

The switch and LED interfaces are parallel IOs that the example SOPC system uses for user control and observation. The Altera Cyclone III Starter Board has four pushbutton switches that are connected to the FPGA and therefore the switch interface. There are also four LEDs connected to the Cyclone III FPGA. These are connected to the LED interface. This tutorial demonstrates the processor reading the switch inputs and then flashing the LEDs by writing to the LED interface.

1.2.7 External tristate bridge and external memories

This tutorial uses the external flash and SSRAM on the Cyclone III Starter Board to store application code, variables and data. To use the external memories, the example SOPC system must convert the System Interconnect Fabric's unidirectional data buses to a single tristate data bus for the FPGA IO. This is done by instantiating an external tristate bridge in the design. This bridge also converts the 32-bit System Interconnect Fabric to a 16-bit external data bus. The tristate bridge also includes the logic necessary to buffer and steer the data bytes from/to the external memory.

1.2.8 External SSRAM PLL

The external SSRAM on the Altera Cyclone III Starter Board is a synchronous SRAM requiring a clock. The timing of the clock with respect to the external address and data from the FPGA is critical. Therefore a PLL is used to generate the clock with the proper delay timing.
1.3 Example SOPC system configuration

Figure 1-3 shows a screen shot of the full example SOPC system configuration.
Chapter 2
Getting Started

This chapter describes how to get started with the Cortex-M1 FPGA Development Kit by using the provided Example SOPC System design and the Cyclone III Starter Board. It contains the following sections:

- About getting started on page 2-2
- Loading the FPGA using the Quartus II Programmer on page 2-3
- Software execution on page 2-7
- Interacting with the example software on page 2-8
- Viewing the example software source code (optional) on page 2-11
- Simulation environment on page 2-13.
2.1 About getting started

The Cortex-M1 FPGA Development Kit contains several components. The main building block is the processor. An example SOPC system containing hardware, software, and a simulation environment is provided. The example system FPGA design contains the processor with debug enabled, and various Altera SOPC peripherals which are available for use on Altera FPGAs.

——— Note ————
The example hardware is targeted specifically to the Altera Cyclone III Starter Board.

The example SOPC system software consists of a RealView MDK project. The software uses the JTAG UART that is instantiated in the hardware design to display a menu of tests in the Cortex-M1 Terminal application, which runs on a PC. It also accepts user input from the Cortex-M1 Terminal application so that each test can be selected.

The simulation environment contains the same example SOPC system hardware together with a simulation version of the RealView MDK project. The simulation software differs in two main ways from the example SOPC system software:
• no menu is displayed, and the tests run sequentially
• the individual tests are optimized for simulation.

——— Note ————
The evaluation version of the development kit does not support simulation.

To become more familiar with the individual pieces of the Cortex-M1 FPGA Development Kit, a quick walk-through of the product is recommended. The following sections describe the basic steps in running the example software on a prebuilt example system. More details on each aspect of the demonstration, such as building the hardware system, software, and simulation, are given in future chapters.

——— Note ————
Ensure that the Altera Cyclone III Starter Board is connected before you start programming.
2.2 Loading the FPGA using the Quartus II Programmer

To program the FPGA device with the pre-compiled example system hardware design, perform the following steps:

1. Run the Altera Quartus II software on the PC. Select **File → Open Project** from the menu.

2. Browse to the supplied CortexM1_ExampleDesign.qpf project in the ExampleDesign directory of the Cortex-M1 FPGA Development Kit installation and select **Open** as shown in Figure 2-1. This loads the example hardware design project into the Quartus II software.

3. Open the Quartus II Programmer by selecting **Tools → Programmer**.

4. Click **Hardware Setup** and select **USB-Blaster** in the **Currently selected hardware** drop-down list as shown in Figure 2-2 on page 2-4.
5. Click Close.

6. In the Quartus II Programmer window, click Auto Detect from the button list. See Figure 2-3.
7. Select the **EP3C25** device then click on **Change File**.

8. Select the supplied SOF file found in the `ExampleDesign` directory:
   - CortexM1_ExampleDesign.sof if you are using the full version of the ARM Cortex-M1 FPGA Development Kit
   - CortexM1_ExampleDesign_time_limited.sof if you are using an evaluation version of the Development Kit.

9. Turn on the **Program/Configure** option for the device, as shown in Figure 2-4.

<table>
<thead>
<tr>
<th>File</th>
<th>Device</th>
<th>Checksum</th>
<th>Usercode</th>
<th>Program/Configure</th>
<th>Verify</th>
<th>BlaChe</th>
</tr>
</thead>
<tbody>
<tr>
<td>CortexM1_ExampleDesi...</td>
<td>EP3C25F324</td>
<td>0052CA0A</td>
<td>FFFFFFFF</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   **Figure 2-4 SOF options**

10. Click **Start**. The Programmer loads the SOF directly into the FPGA device. Figure 2-5 shows the Programmer when programming completes.

   **Figure 2-5 Programming complete**
—— Note ————
If you are using an evaluation licence for the processor, you will see a status window about the OpenCore Plus evaluation feature. Canceling the information dialog causes the evaluation timeout to expire, and the processor will no longer function. If you disconnect the USB cable or close the Quartus II software, the evaluation will expire when the timeout period is reached.

———  Note  ————
When the development board is turned off this design does not persist so it is necessary to reload the SOF each time the FPGA is turned on or the Reconfigure button is pressed. You can program the design into the flash device on the Cyclone III Starter Board using a POF file so that it is loaded into the FPGA automatically at power-on. For details about this see Chapter 7 Software Considerations and Customization and the Cyclone III Starter Kit User Guide.

——— Note ————
If you recompile the Example Design project, the prebuilt programming file is overwritten with your new design.
2.3 Software execution

When programming is complete, the LEDs will blink in a pattern. This is because the processor has come out of reset, and is executing the code that was initialized into its Instruction Tightly Coupled Memory (ITCM). Interacting with the example software on page 2-8 shows how to explore and interact with the software.
2.4 Interacting with the example software

This section describes:
- Running the Cortex-M1 Terminal
- Initial feedback from the board on page 2-9
- The menu on page 2-10.

2.4.1 Running the Cortex-M1 Terminal

As previously described, the example software provides a user interface through the Cortex-M1 Terminal application. Start the Cortex-M1 Terminal by opening the Windows Start menu and selecting **Programs → Cortex-M1 FPGA Development Kit → Cortex-M1 Terminal**. Figure 2-6 shows the Cortex-M1 Terminal.

Select **Connection → Open…** from the menu. If the **Connection Settings** dialog does not appear, ensure that the correct version of Altera's Quartus II software is installed. Keep the default settings, that is, all **Autoselect**, and select **OK** to activate the link to the JTAG UART. It is now possible to interact with the JTAG UART using the embedded menu.
2.4.2 Initial feedback from the board

The program is executing, and has written some characters to the Cortex-M1 Terminal and is waiting for you to **Press any switch to continue**... as shown in Figure 2-7. The switches refer to any of the four push button switches on the Cyclone III Starter Board.

![Cortex-M1 Terminal window awaiting switch press](image)

**Figure 2-7 Cortex-M1 Terminal window awaiting switch press**

Press and hold any of the four user switches and wait for the terminal to respond with the menu options to continue to the menu portion of the program. This causes the LEDs to stop flashing, and the program prints a menu to the Cortex-M1 Terminal as shown in Figure 2-8 on page 2-10.
2.4.3 The menu

The tests and their functions are described in *Menu tests* on page 3-4. You can run the tests by typing the appropriate number into the Cortex-M1 Terminal window.
2.5 Viewing the example software source code (optional)

The source code that is pre-loaded into the ITCM of the processor is provided in the project FPGACode.Uv2. To explore the example project, run the RealView MDK integrated compiler/debugger tool.

Open the RealView MDK tools by clicking Start → Programs → Keil uVision 3. Load the provided FPGACode.Uv2 project by selecting Project → Open Project from the menu. This project is located in the ExampleSoftware\FPGA directory of the Cortex-M1 FPGA Development Kit.

After loading the project, note that the RealView MDK has a standard look and feel, with the Project Workspace window on the left. Browse through the categories in Project Workspace and double-click on the file(s) of interest for viewing.

Note

Program execution starts in the startup source file, Startup.s. The main() routine is the first non-assembly code that is executed. This is in the Main.c file as shown in Figure 2-9 on page 2-12.
Figure 2-9 uVision main source listing
2.6 Simulation environment

This section describes:

- Generating the environment
- Using the simulation environment on page 2-15.

2.6.1 Generating the environment

The example design in the Cortex-M1 Development Kit is shipped without any automatically generated Verilog source output from SOPC Builder. Verilog source for the example SOPC system is required for simulation. You can generate the Verilog source from the included project files.

Note

It is not possible to run an evaluation version of the processor in simulation. If you generate simulation files for an evaluation version, you will receive an error when you run the simulation.

It is not necessary to regenerate the entire FPGA system to load the design into the FPGA. The FPGA programming files are included in the kit.

Open the example Quartus II Project

Open Quartus II IDE, and select File → Open Project.

The Open Project dialog window appears as shown in Figure 2-10 on page 2-14. Browse to the installed directory and select CortexM1_ExampleDesign.qpf.
Click **Open**.

### Generate the SOPC system

Open SOPC Builder by selecting **Tools → SOPC Builder**.

Click **Generate** located at the bottom of the SOPC Builder window. See Figure 2-11.

The example SOPC system is generated. You can now simulate the system. You can also regenerate the FPGA programming files.
Note

It is not necessary to regenerate the FPGA programming files. These are provided in the kit.

Regenerate the FPGA design (optional)

In the Quartus II window, select **Processing → Start Compilation**. This starts the following sequence:

- compilation
- synthesis
- fitting (place-and-route)
- assembly (programming file generation)
- timing analysis.

### 2.6.2 Using the simulation environment

A simulation environment is provided in the directory, `CortexM1_ExampleDesign_sim`. The simulation consists of several different components:

- SOPC hardware and testbench
- ModelSim scripts
- the simulation software, `ITCM.hex`.

The hardware and scripts are automatically generated by SOPC builder. The simulation software project is provided in the `ExampleSoftware\Simulation` directory. This project, like the example project, was compiled with the RealView MDK toolset.
Chapter 3
Demonstration Software and Example SOPC System

This chapter describes the example SOPC system demonstration (FPGA Code project) software. It contains the following sections:

- *About the demonstration software* on page 3-2
- *Source files* on page 3-3
- *Menu tests* on page 3-4.
3.1 About the demonstration software

The FPGA Code demonstration first blinks the user LEDs on the Cyclone III Starter Board in a back-and-forth pattern on startup to show that the code is executing on the processor. The code displays a menu in the Cortex-M1 Terminal application when one of the four user pushbuttons is pressed.

The menu enables one of six tests to be run which are designed to exercise the functionality of the example SOPC system. The available tests are:

1. External SRAM test.
2. Flash test.
3. Timer test.
4. Read system ID test.
5. Switches and LEDs test.
6. Upper DTCM test.

You can choose to run a specific test by ensuring that the text area in the Cortex-M1 Terminal application has focus, then typing in the required test number using the PC keyboard.

In summary, the organization of the FPGA Code demonstration software is:

- initialization (boot code)
- blink LEDs loop
- menu loop (polled input from Cortex-M1 Terminal using the JTAG UART).
3.2 Source files

Table 3-1 shows the source files located in the ExampleSoftware\FPGA subdirectory of the Cortex-M1 FPGA Development Kit installation.

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup.s</td>
<td>Vector table and initial boot code for the processor.</td>
</tr>
<tr>
<td>Main.c</td>
<td>Main routine for the FPGA software.</td>
</tr>
<tr>
<td></td>
<td>The main() function is implemented as a forever loop that:</td>
</tr>
<tr>
<td></td>
<td>* displays menu text to the JTAG UART terminal</td>
</tr>
<tr>
<td></td>
<td>* samples the user's menu selection</td>
</tr>
<tr>
<td></td>
<td>* executes the test associated with the user's selection.</td>
</tr>
<tr>
<td>Isr.c</td>
<td>Interrupt service routines for the JTAG UART and Timer modules.</td>
</tr>
<tr>
<td>Tests.c</td>
<td>Code that executes the individual tests:</td>
</tr>
<tr>
<td></td>
<td>* external SRAM test</td>
</tr>
<tr>
<td></td>
<td>* flash test</td>
</tr>
<tr>
<td></td>
<td>* timer test</td>
</tr>
<tr>
<td></td>
<td>* read SystemID test</td>
</tr>
<tr>
<td></td>
<td>* switches and LEDs test</td>
</tr>
<tr>
<td></td>
<td>* upper DTCM test</td>
</tr>
<tr>
<td></td>
<td>These tests are described in Menu tests on page 3-4.</td>
</tr>
<tr>
<td>Uart.c</td>
<td>JTAG UART interface code.</td>
</tr>
<tr>
<td>Retarget.c</td>
<td>Configures the implementation of C standard library I/O functions for the</td>
</tr>
<tr>
<td></td>
<td>target hardware.</td>
</tr>
<tr>
<td>BoardIf.h</td>
<td>Useful typedefs and macros for writing and reading to the various</td>
</tr>
<tr>
<td></td>
<td>hardware modules.</td>
</tr>
<tr>
<td>Registers.h</td>
<td>Register map of the hardware design.</td>
</tr>
<tr>
<td>Uart.h</td>
<td>#defines relevant to the operation of the JTAG UART.</td>
</tr>
<tr>
<td>Flash.h</td>
<td>#defines relevant to the Intel Flash chip on the development board.</td>
</tr>
<tr>
<td>Externals.h</td>
<td>Contains function prototypes and extern declarations.</td>
</tr>
</tbody>
</table>
### 3.3 Menu tests

The menu is a console window style interface for selecting tests to execute on the Cyclone III Starter Board. These tests are described in:

- *External SRAM test*
- *Flash test*
- *Timer test*
- *Read SystemID test*
- *Switches and LEDs test* on page 3-5
- *Upper DTCM test* on page 3-5.

#### 3.3.1 External SRAM test

This tests the external SRAM chip, and the tristate bridge interface to the external SRAM. All locations in the external SRAM are written with a random pattern, and all locations are then read back. This is done for 8-bit, 16-bit, and 32-bit accesses. The test only passes if the software reads back the expected SRAM data values.

#### 3.3.2 Flash test

This tests the external FLASH chip, and the tristate bridge interface to the external FLASH. The FLASH is configured so that the manufacturer and device IDs can be read. The test only passes if the device and manufacturer IDs match the expected values, as documented in the Intel FLASH datasheet.

#### 3.3.3 Timer test

This tests the SOPC timer module and the interrupt capability of the processor. In this test, the timer is configured to timeout after 4 seconds and produces an interrupt signal to the processor. The test then waits for a flag to be set inside the timer interrupt service routine. The timer test passes if it detects that the timer flag has been set inside the Interrupt Service Routine (ISR).

#### 3.3.4 Read SystemID test

This test reads and displays the two read-only ID registers in the System ID block. Because the System ID generally changes each time the SOPC system is generated, this test is not self-checking. The user can compare the returned ID to the ID declared in the Verilog source in *sysid.v*. 
3.3.5 Switches and LEDs test

This tests the SOPC PIO modules for the switches and LEDs. This test enters a forever loop that blinks the user LEDs on the Altera Cyclone III Starter Board in a back-and-forth pattern.

3.3.6 Upper DTCM test

This tests the upper portion of the DTCM, which is mapped to Altera on-chip memory. The lower portion of DTCM is used for program stack and global space, so only the upper portion of DTCM is tested. Upper DTCM locations are written with a random pattern, and these locations are then read back. This is done for 8-bit, 16-bit, and 32-bit accesses. The test only passes if the software reads back the expected data values from the DTCM.
Chapter 4
Debugging Example SOPC System

This chapter shows you how to use the RealView MDK tools to debug software targeted at the Example SOPC System running on the Cyclone III Starter Board. This is a step-by-step tutorial showing how you open the supplied FPGA Code project, load the software into the Cortex-M1 ITCM on the Cyclone III FPGA, and then run a debugging session.

This chapter contains the following sections:
• Assumptions on page 4-2
• RealView MDK tools setup on page 4-3
• Emulation on the Cyclone III Starter Board using the MDK tools on page 4-5
• Loading the Cyclone III Starter Board and starting the Cortex-M1 terminal on page 4-11
• Debugging the demo on page 4-12.
4.1 Assumptions

It is assumed that the RealView MDK (Keil μVision3) tools are installed on the host PC, and that the Cyclone III Starter Board is plugged in and loaded with the provided Cortex-M1 FPGA design.
4.2 RealView MDK tools setup

Open the RealView MDK tools by clicking Start → Programs → Keil uVision3.

Load the provided FPGACode.Uv2 project by selecting Project → Open Project from the menu as shown in Figure 4-1. The FPGACode.Uv2 project file is located in the ExampleSoftware\FPGA subdirectory of the Cortex-M1 FPGA Development Kit installation directory.

![Open File](image)

**Figure 4-1 Select project file**

After loading the project, note that the RealView MDK has a standard look and feel, with the Project Workspace window on the left, the edit window on the right, and the output window on the bottom as shown in Figure 4-2 on page 4-4.

See the RealView MDK documentation for more information about how to use the tool. This can be accessed by clicking the Help menu in the RealView MDK tools.
RealView MDK projects can have multiple targets, and each target can have different options. The active target is selected using the target selection list in the build toolbar, which is highlighted in Figure 4-2. The Cortex-M1 FPGA Development Kit example software contains two targets:

- **Emulator** is for running code on an actual embedded ARM Cortex-M1 processor hardware target.

- **Simulator** is for running code on an ARM Cortex-M1 processor Instruction Set Simulator (ISS) within RealView MDK. The Simulator can be useful for identifying software bugs without the requirement for actual hardware.

When the software relies heavily on interaction with custom peripherals, it is desirable to run the code on the actual system that contains the peripherals. That is, on the FPGA with the Emulator target.

The rest of this chapter describes the Emulator target.
4.3 Emulation on the Cyclone III Starter Board using the MDK tools

This section describes:

- Important project settings
- Compilation on page 4-10.

4.3.1 Important project settings

The FPGA Code project has been set up with a number of important settings, such as compiler, linker, debugger, and output options. These settings are required to target the software to the example SOPC system and to generate the correct output files.

This section describes the relevant project settings. You can use similar settings for your own software projects, but you will have to adjust certain settings for your own hardware design.

To show the project settings dialog, right-click on Emulator in the Project Workspace and select Options for Target Emulator from the pop-up menu. Note the settings on each tab for the FPGA Code project.

- Go to the Target tab as shown in Figure 4-3 on page 4-6. This is where information about the memory map of the system is entered, specifically the details about the code and data space. Note the following settings:
  - IROM1 Start: 0x0000, Size: 0x8000
  - IRAM1 Start: 0x20000000, Size: 0x4000
  - Xtal: 50MHz
  - Use MicroLIB: enabled.

Note

The IROM1 region refers to the Cortex-M1 ITCM, and the IRAM1 region refers to the Cortex-M1 DTCM. The RAM1 region refers to the off-chip SRAM.
Go to the Output tab to see the output settings as shown in Figure 4-4 on page 4-7. Note the following settings:

- **Name of Executable**: ITCM
- **Create Executable** radio button is selected
- **Debug Information**, **Create HEX File**, and **Browse Information** check boxes are checked.

**Note**

The name in the **Name of Executable** field controls the name of the compiled software image. The **Create HEX File** option must be enabled if you want to initialize the Cortex-M1 TCMs with the software image.
Go to the **Linker** tab to see the Linker settings shown in Figure 4-5 on page 4-8.

--- **Note**

The **Use Memory Layout from Target Dialog** checkbox is checked, which causes RealView MDK to automatically create a scatter-loading file based on the memory layout in the Target tab. You can deselect this option if you want to write a custom scatter file. This option is not available if you are using RealView MDK in evaluation mode.
Go to the **Debug** tab to see the debug settings as shown in Figure 4-6 on page 4-9.

**Note**

The **Load Application at Startup** and **Run to Main** options are selected. The Altera Blaster Cortex Debugger driver, which implements the USB debug connection in the Cortex-M1 FPGA Development Kit, is selected as the active driver.

When a debug session is activated, the **Run Application at Startup** option causes RealView MDK to automatically download the compiled executable to the development board. The **Run to Main** option causes RealView MDK to execute the startup code in `Startup.s` and stop at the entry point into the `main()` code.
By clicking on **Settings**, the specific configurations for the Altera Blaster Debugger are displayed, as shown in Figure 4-7 on page 4-10.

--- **Note** ---
Select **Cancel** on all dialogs to discard any accidental changes.
4.3.2 Compilation

To compile and link the project, right-click on *Emulator* in the Project Workspace, and select **Rebuild All Target Files** from the pop-up menu. Information about the compile and link is displayed in the RealView MDK output window.
4.4 Loading the Cyclone III Starter Board and starting the Cortex-M1 terminal

Power the Cyclone III Starter Board and connect it to the PC using the USB cable.

Ensure that the FPGA has already been programmed with the example system. If not, see *Loading the FPGA using the Quartus II Programmer* on page 2-3 for instructions on how to download the .sof file.

Open Cortex-M1 Terminal by opening the Windows Start menu and selecting **Programs → Cortex M1 FPGA Development Kit → Cortex-M1 Terminal**. Select **Connection → Open…** from the Cortex-M1 Terminal menu. If the **Connection Settings** dialog does not appear, confirm the correct version of Altera's Quartus II is installed. Keep the default connection settings (all autoselect) and select **OK** to activate the link to the JTAG UART.

![Cortex-M1 Terminal application window](image)

**Figure 4-8 Cortex-M1 Terminal application window**

The terminal application is now ready to display messages from the embedded software running on the development board and to accept user input.
4.5 Debugging the demo

You can now load the software into memory on the Cyclone III Starter Board.

In the RealView MDK window, select Debug → Start/Stop Debug Session. At this point, the code has been loaded onto the board, and execution has continued through the initial startup code, and has halted at the `main()` as shown in Figure 4-9 on page 4-13.

Note
If you are running RealView MDK in evaluation mode, you will see warnings about a restricted code size limit.

You can now set breakpoints and step through the code.
4.5.1 Breakpoint example part 1

If RealView MDK is currently displaying a disassembled source view, open the source code for `Main.c` by clicking on the `Main.c` tab at the bottom of the source window. If there is no tab for `Main.c`, open the file from the Project Workspace.

Set a breakpoint on the `while(1)` statement in the `main()` function by double-clicking in the left margin beside the line. A red breakpoint appears, indicating that execution of the program stops here.

Select **Debug → Run.**
Note

The LEDs on the Cyclone III Starter Board are flashing. This is because the breakpoint has not been reached and code is still executing.

There is a loop inside the userIOTest() statement, just above while(1), that is waiting for user input. Press one of the four user switches on the Cyclone III Starter Board and hold it down until the LEDs stop flashing.

In the RealView MDK window, execution has stopped at the breakpoint as shown in Figure 4-10.

Figure 4-10 Program break at while(1) loop
4.5.2 Breakpoint example part 2

Clear the breakpoint by double-clicking on the line, then select **Debug → Run** from the menu to resume program execution. The test menu now appears on the Cortex-M1 Terminal as shown in Figure 4-11.

![Cortex-M1 Terminal initial test menu window](image)

**Figure 4-11 Cortex-M1 Terminal initial test menu window**

To run each test, type the test number in the edit control of the Cortex-M1 Terminal and observe the pass or fail responses.

See the RealView MDK documentation for more information about software debug.
Chapter 5
Simulating Example SOPC System

This chapter describes how to simulate the example SOPC system using the Altera version of ModelSim. SOPC Builder generates an HDL version of the design that can be simulated using the Altera Megafunction library and Cyclone III library elements. It also creates a testbench that you can use to stimulate the example SOPC system.

Note

Simulation in this chapter refers to simulating the Verilog model of the complete Example SOPC System, as opposed to running the MDK simulator described in RealView MDK tools setup on page 4-3, which is purely an Instruction Set Simulator. Skip this chapter if you are using an evaluation version of the ARM Cortex-M1 FPGA Development Kit. Simulation is not supported in the evaluation version, and the simulation project is not supplied.

This tutorial uses the SOPC Builder output and describes how to expand the testbench to generate a complete test environment using software compiled with the RealView MDK tools.

This chapter contains the following sections:

- Generated SOPC HDL testbench on page 5-3
- Simulation software description on page 5-4
Simulating Example SOPC System

- *Generating the ITCM image file* on page 5-5
- *Simulating the example SOPC system design* on page 5-6.
5.1 Generated SOPC HDL testbench

SOPC Builder generates the entire example SOPC system HDL, Cortex-M1 processor and subsystem, in addition to the simulation environment. However, SOPC Builder does not generate any software tests for the example SOPC system. This is the function of RealView MDK. The simulation environment, including simulation scripts, is contained in the directory CortexM1_ExampleDesign_sim, which is in the ExampleDesign directory of the Cortex-M1 FPGA Development Kit installation.

The simulation uses the HDL generated from SOPC Builder for the FPGA, the Altera Cyclone III primitive library models, external memory models, and testbench code.

Note

Because the SOPC Builder testbench only simulates the SOPC Builder-generated subsystem, the FPGA top level, CortexM1_ExampleDesign_Top.v, is not required for the simulation. It is only required for the FPGA synthesis.

The external memory models are in the simulation directory. The testbench HDL code is contained in the SOPC top level file ExampleDesign\CortexM1_ExampleDesign.v in the Cortex-M1 FPGA Development Kit installation directory. This file appears after the SOPC system is generated.
5.2 Simulation software description

There is a separate RealView MDK software project that is targeted specifically to the simulation environment. This project, which is provided with source code, can be opened from the file ExampleSoftware\Simulation\SimCode.Uv2.

The simulation software tests are similar to the emulation software tests provided for the FPGA build, with two significant differences:

- There is no menu interface in main(). The individual tests are instead executed in sequence. This enables the simulation to proceed with minimal user interaction.

- Each test is optimized for speed. This is because simulating even 0.5 seconds of execution could take several hours of actual time using the ModelSim-Altera Web Edition tools.

The simulation software runs through a series of short tests for the following:

- External SRAM test
  - Write 8-bit data to the first eight bytes of SRAM, then read the data back.
  - Write 16-bit data to the first eight bytes of SRAM, then read the data back.
  - Write 32-bit data to the first eight bytes of SRAM, then read the data back.

- Timer test:
  - Test the interrupt capability of the timer. Timer interrupts after about 10 microseconds.

- System ID test:
  - Read the System ID registers.

- Upper DTCM test:
  - Write 8-bit data to 8 bytes DTCM, then read the data back.
  - Write 16-bit data to 8 bytes DTCM, then read the data back.
  - Write 32-bit data to 8 bytes DTCM, then read the data back.

- Switches and LEDs test:
  - Enable the LEDs individually.

Note

The itcm.hex file for the example SOPC system design is automatically placed into the simulation directory by the RealView MDK project. Rebuilding the project replaces the itcm.hex file in the simulation directory.
5.3 Generating the ITCM image file

For the example SOPC system design, the processor ITCM (implemented as an Altera on-chip RAM) must be preloaded with Cortex-M1 instructions. You can create a suitable ITCM image using the RealView MDK tools.

The RealView MDK output .axf format is different from the .hex format read by the Altera on-chip RAM simulation models. A hex2hex conversion tool is included with the release, and its invocation is integrated into the RealView MDK tools so that the final Altera .hex file for the on-chip RAM simulation model is produced after building the project. For more information on the hex2hex conversion script, including how to use it in your own projects, see Application Note 215: Converting memory initialization files in the ARM Cortex-M1 FPGA Development Kit.

The hex2hex converter is integrated into the RealView MDK toolset in the following two ways:

- In the project’s Options dialog in the Output tab, the Create hex file check box is selected so that RealView MDK will create an Intel Hexadecimal file from the original image.
- In the project’s Options dialog in the User tab, the Run #1 checkbox is selected in the Run User Programs after Build/Rebuild section to run the hex2hex conversion utility.

The following command appears in the edit field directly adjacent to the Run #1 checkbox:

```
..\..\Utilities\hex2hex.bat --infile="#H" --outfile="..\..\ExampleDesign\CortexM1_ExampleDesign_sim\%H" --oformat=hex --width=4 --saddr=0x00000000 --osize=32K
```

This command is executed on a build or a rebuild. The hex2hex conversion script, hex2hex.bat, takes several arguments. The #H parameter in the --infile argument is interpreted by RealView MDK as the full path to the original Intel Hexadecimal file that it created. The %H in the --outfile argument is interpreted as the filename (no path) of the original hex file. The result is that the Altera-ready hex file, itcm.hex, is placed in the simulation directory of the example system where it can be used as the ITCM memory initialization file for system simulation.

--- Note ----

The hex2hex conversion utility requires that the Quartus II software is installed.
5.4 Simulating the example SOPC system design

Ensure that ModelSim-Altera Web Edition is installed and that the HDL Simulator path is set correctly, as explained in the Cortex-M1 FPGA Development Kit v1.1 Installation Guide.

Before generating the system ensure that the option **Simulation. Create project simulator files** is enabled in the **System Generation** tab of the SOPC Builder window, as shown in Figure 5-1. Generate the system by pressing the **Generate** button after enabling this option.

From SOPC Builder, click on the **Run Simulator** button to launch ModelSim-Altera.

![Figure 5-1 Run Simulator option and button](image)

The following sections describe how to run the provided simulation scripts and ModelSim commands to compile the design, view an example wave file, and then begin the simulation:

- Compile and load the design
- Open the Wave viewer with Example SOPC System signals on page 5-8
- Open the USB_UART log on page 5-8
- Run the simulation on page 5-8
- Halt and view the initial simulation results on page 5-9
- Continue the simulation on page 5-9
- Debugging simulation results on page 5-10.

5.4.1 Compile and load the design

When started from SOPC Builder, ModelSim-Altera automatically switches to the correct directory location, CortexM1_ExampleDesign_sim, and executes the setup_sim.do script. The setup_sim.do script is created automatically by SOPC Builder and contains some useful command aliases. One of the aliases is s, which compiles and loads the design.
Type s to compile the simulation models as shown in Figure 5-2.

Note

The following warning is expected:

```markdown
# ** Warning: (vsim-3015) ..../CortexM1_ExampleDesign.v(4347): [PCDPC]
- Port size (16 or 16) does not match connection size (32) for port 'data'.
```
# Region: /test_bench/the_Ext_FLASH_Ctrl

The following errors are also expected from the Web Edition tools, but do not affect the simulation:

# ** Error: (vsim-19) Failed to access library 'stratixgx_gxb_ver' at "stratixgx_gxb_ver".
# ** Error: (vsim-19) Failed to access library 'stratixgx_ver' at "stratixgx_ver".
# ** Error: (vsim-19) Failed to access library 'altgxb_ver' at "altgxb_ver".

## 5.4.2 Open the Wave viewer with Example SOPC System signals

The example SOPC system is supplied with a custom wave file, wave.do, which adds Cortex-M1 signals and signals from other components and the Avalon bus to the waveform display. At the ModelSim prompt type do wave.do to open the wave viewer and display these signals. The signals added by wave.do are useful for observing the bus transactions that are generated by the test software.

--- ** Note  

Although SOPC Builder generates a default wave file which can be invoked using the w alias, this does not add signals from every component to the waveform display.

---

## 5.4.3 Open the USB_UART log

SOPC Builder automatically creates a ModelSim alias to open an interactive output window that views log messages that are printed to the USB_UART component in simulation. Open the USB_UART output window by typing USB_UART_log at the ModelSim prompt. This opens a new window which displays the characters that are printed to the USB_UART as the simulation progresses.

--- ** Note  

The simulation software prints fewer messages than the FPGA software, to speed up the simulation.

---

## 5.4.4 Run the simulation

To run the simulation type:

run -all
5.4.5 Halt and view the initial simulation results

Halt the simulation after a few minutes by selecting Simulate → Break from the menu.

- Note ————

The ITCM signals toggle very soon after the processor comes out of reset. This indicates processor instruction fetches as shown in Figure 5-3.

Figure 5-3 ModelSim-Altera wave view window

5.4.6 Continue the simulation

Continue the simulation by typing:

```
run -all
```
When the simulation has finished, a pass or fail message is printed to the USB_UART log window.

5.4.7 Debugging simulation results

If the simulation fails, the log messages indicate which part of the test caused the failure. You can also turn on a high-verbosity mode in the simulation software to print extra diagnostic information that can help you to identify what caused the failure. To enable high-verbosity mode in simulation:

- terminate any active simulation
- open the SimCode project in RealView MDK, and open the Tests.c file
- switch to the Configuration Wizard tab at the bottom of the Tests.c editor window
- turn on the Increased verbosity option and rebuild the software
- restart the simulation to use the new software image with increased verbosity.

Note

Activating high-verbosity mode increases simulation time.
Chapter 6
RTX Real Time Operating System Example

This chapter describes a real time operating system example using RTX. It contains the following sections:

- About the RTX real time operating system example on page 6-2
- Examining the project on page 6-3
- Program description on page 6-4
- Hex file generation on page 6-5.
6.1 About the RTX real time operating system example

RTX is a real-time operating system that is integrated with the RealView MDK tools. The RTX_Blinky example is a software project that has been adapted to work with the Altera Cyclone III Starter Board. The software demonstrates how to use the built-in RTX libraries to control a traffic light. The traffic light in the RTX_Blinky example is represented by the bank of four user LEDs labeled LED4, LED3, LED2, LED1 on the Cyclone III Starter Board.

Together, LED4 and LED3 represent the traffic light stop, caution, and go signals. The traffic light states are defined as follows:

- LED4 ON and LED3 ON: Traffic Light = Red (stop)
- LED4 OFF and LED3 ON: Traffic Light = Green (go)
- LED4 ON and LED3 OFF: Traffic Light = Yellow (caution)
- LED4 OFF and LED3 OFF: Traffic Light = Off.

Together, LED2 and LED1 represent the corresponding walk signal. The walk signal states are defined as follows:

- LED2 ON and LED1 ON: Walk Signal = Don't Walk
- LED2 OFF and LED1 ON: Walk Signal = Walk
- LED2 OFF and LED1 OFF: Walk Signal = Off.

--- Note ---

The walk signal caution is represented by toggling the states Don't Walk and Off.

---
6.2 Examining the project

Start the μVision tool and open the project using Project → Open Project..., navigate to the ExampleSoftware\RTX_Blinky directory in your installation and then select Blinky.Uv2 and click Open.

The project contains three source files:

**Startup.s**  This file contains the startup code for the application

**Blinky.c**  This file contains main(), RTX OS calls to initialize the OS, create and delete tasks, and general functions that blink the LEDs.

**RTX_Config.c**  This file contains code to configure the RTX operating system.

To enable the RTX library in a RealView MDK project, select Project → Options for Target Emulator from the menu, select the Target tab, and set the Operating System selection to RTX Kernel as shown in Figure 6-1.

![Figure 6-1 Linking the RTX library to a project](image)
6.3 Program description

Program execution begins in Startup.s. The processor loads its stack pointer and vectors from the __Vectors table, which starts at address 0. Execution branches to __main in the C library, which sets up the stack, and initializes global variables. When these program initializations are complete, program execution branches to the main() function in Blinky.c.

In main(), there is only one call, that is, os_sys_init (init). This initializes the RTX operating system and then branches to the init() task, which is also located in Blinky.c. init() creates the blinking() task, waits for 500 ticks then sets a flag, normal_mode. The normal_mode flag is a signal to the blinking() task to create the lights() task. The blinking() task executes for 500 ticks, after which lights() is executed forever.

blinking() blinks the lights as follows:
- traffic light alternates yellow and off
- walk signal is a solid red.

lights() iterates through the following states:
- traffic light red, walk signal stop
- traffic light green, walk signal stop
- traffic light yellow, walk signal stop
- traffic light red, walk signal stop
- traffic light red, walk signal walk
- traffic light red, walk signal alternates stop and off indicating caution.
6.4 Hex file generation

For convenience, Altera-compatible .hex files are generated for FPGA ITCM initialization and for Simulation. After the RTX project is built, these files are placed in the project's obj directory. You can rename and copy these .hex files to the desired Altera Quartus II project for use in preloading FPGA memories or system simulation.
Chapter 7
Software Considerations and Customization

You can customize the provided design, software, or both for a specific purpose. This chapter describes several design considerations when making customizations. It contains the following sections:

- Memory resources on page 7-2
- Flash-FPGA load and code space considerations on page 7-3
- Preloading Altera RAMs with Cortex-M1 instructions on page 7-4
- Example SOPC system clocking on page 7-6.
7.1 Memory resources

There is about 66KB of internal RAM in the targeted Cyclone III FPGA on the Cyclone III Starter Board. There is 1MB of external SSRAM, 16MB of external FLASH, and 32MB of DDR SDRAM.

Note
SDRAM is beyond the scope of this document, but is mentioned because some specialized designs might require this resource.

The ITCM and DTCM tightly coupled memories in the processor can be configured individually to use varying amounts of the internal RAM resource from 0 to about 95%. In the example design, there is 32KB of instruction memory, and 16KB of data read/write memory. This is generally sufficient for small programs.

If more memory is required for Cortex-M1 instructions or code, the following applies:

- Allocate more or all of the internal memory resources to the ITCM. The DTCM size might have to be reduced or removed altogether depending on the chosen ITCM size. If this is the case, some of the external resources, that is SRAM, SDRAM or both, must be allocated to the processor for its data space.

- Supplement the internal memory resources with an external memory resource, flash, SRAM, or SDRAM. It is advantageous to have an additional amount of SRAM and flash to load and debug large programs that would not fit into internal FPGA RAM. The drawback to having code in an external RAM is that the code cannot be preloaded.

- It might also be necessary to create a scatter-loaded Cortex-M1 executable to load instructions into two distinct memories. This is described in more detail later in this chapter.

- Flash is recommended for instructions only after the entire design and software have been debugged, and a release version of the ARM Cortex-M1 code is ready.

If more memory is required for Cortex-M1 read/write data, the following applies:

- Allocate more or all of the internal memory resources to the DTCM. The ITCM size might have to be reduced or removed altogether depending on the chosen ITCM size. If this is the case, some of the external resources such as FLASH, SRAM, or SDRAM must be allocated to the Cortex-M1 for its instruction space.

- Supplement the internal memory resources with an external memory resource: SRAM, or SDRAM. This is generally the best solution for mid-size applications. Flash is not in read/write space, and cannot be used for this purpose.
7.2 Flash-FPGA load and code space considerations

The Altera Cyclone III Starter Board holds the default FPGA power-on image in the lower half of flash. If you want to use the onboard flash for storage, it is recommended that you use the upper part of flash.

All of flash is available to you in the example design, but you can easily restrict your software project to use only the upper portion of flash. In RealView MDK you can configure the valid address range in the Target tab of the project options window to be the upper portion of flash, or you can create a custom scatter file. Alternatively, you can protect the lower area of flash by modifying the address assignment in the top-level Verilog file, for example:

```vhdl
//assign tristate_bridge_address[23:1] = tristate_bridge_address_internal[23:1];
assign tristate_bridge_address[23:1] = {1'b1, tristate_bridge_address_internal[22:1]};
```

This does not affect the SRAM address, because the SRAM is sixteen times smaller and does not use the upper address bits. This change effectively cuts the size of the flash to 8MB.

If you want your design to be initialized from flash memory when the board is powered on, instead of using the Quartus II programmer each time, see Application Note 214: Flash programming in the ARM Cortex-M1 FPGA Development Kit. This Application Note also shows how to initialize software code into the flash.
7.3 Preloading Altera RAMs with Cortex-M1 instructions

This section describes:

- Loading on-chip memories
- Scatter loading.

7.3.1 Loading on-chip memories

After a Cortex-M1 .axf image file has been built with the RealView MDK, no more steps are necessary if you are using the debugger to load the on-board memories. However, if you want to preload the memories so that they are initialized at power-on, you must create a Hex file that is compatible with the Quartus II tools and use this when you synthesize the design.

For information about how to pre-initialize the Cortex-M1 TCMs, see Application Note 213: Cortex-M1 TCM Initialization.

7.3.2 Scatter loading

For some designs you might want to have code loaded into separate memories. In the example design, it is only necessary to preload a single memory, ITCM, so a custom scatter-load is not necessary. If it becomes necessary to load instructions or data across several internal memories, and your memory layout is too complex to describe in the RealView MDK Target options window, you can use a custom scatter file.

With the FPGACode.uv2 project loaded, right-click on Emulator in the Project Workspace, and select Options for Target Emulator from the pop-up menu.

On the dialog box, click the Linker tab, and uncheck the Use Memory Layout from Target Dialog option. The option to edit the scatter file is enabled as shown in Figure 7-1 on page 7-5.

See the Realview MDK User Guide for instructions on how to write a custom scatter file.

If necessary, click Cancel to undo these changes.
Figure 7-1 Options for Target Emulator
7.4 Example SOPC system clocking

For high-performance designs, a faster clock might be necessary to drive the internal logic.

Changes in the clock frequency might require changes to the external SRAM PLL, and to the number of wait states necessary for the external flash. If an SDRAM is implemented, it is also affected by an input clock frequency change. The configuration changes are generally quite technical, and highly dependent on the specific development board. Your local Altera FAE can provide relevant characterizations for the Cyclone III Starter Board.
Chapter 8
Configuring Example SOPC System

This chapter describes how to use the Quartus II v8.0 SOPC Builder tool to recreate the example SOPC system. It is a step-by-step guide that shows how to use the SOPC Builder tool to instantiate the Cortex-M1 component and all the peripheral cores, and properly connect them to the System Interconnect Fabric. It contains the following section:

• Using SOPC Builder to configure and generate the example SOPC system on page 8-2.
8.1 Using SOPC Builder to configure and generate the example SOPC system

The following sections describe how to build the system:

- Step 1: Create the shell project
- Step 2: Configure the system on page 8-8
- Step 3: Generate the testbench on page 8-28
- Step 4: Create the top level wrapper on page 8-29
- Step 5: Add the top-level file to the project on page 8-30
- Step 6: Synthesize the design on page 8-32
- Step 7: Import the pin assignments on page 8-32
- Step 8: Configure the dual-purpose pins on page 8-33
- Step 9: Import timing constraints on page 8-34
- Step 10: Create TCM initialization files on page 8-34
- Step 11: Compile the design on page 8-34.

8.1.1 Step 1: Create the shell project

Open the Quartus II tool and select File → New Project Wizard.

The New Project Wizard Introduction screen is shown in Figure 8-1 on page 8-3.
Figure 8-1 New Project Wizard Introduction window

Click Next >.

Enter a new working directory for the project, as shown in Figure 8-2 on page 8-4. For consistency with this tutorial, use the same project name, CortexM1_ExampleDesign. Accept the default top-level entity name, CortexM1_ExampleDesign. The top-level entity name will be changed after the project is created, rather than choosing a different name here to make Quartus II generate all project files using the name CortexM1_ExampleDesign.
Figure 8-2 New Project Wizard Directory window

Click Next >.

There are no files to add at this time, as shown in Figure 8-3 on page 8-5, so click Next >.
Select the following parameters as shown in Figure 8-4 on page 8-6:

- Device family = **Cyclone III**
- Package = **FBGA**
- Pin count = **324**
- Speed grade = **Any**
- Available devices = **EP3C25F324C8**.

--- Note ---

These values are for the Altera FPGA on the Altera Cyclone III Starter Board.
Figure 8-4 New Project Wizard Family & Device Settings window

Click Next >.

Use the settings for EDA Tool Setup as shown in Figure 8-5 on page 8-7.
Click **Next >**.

Review your entries on the summary page. See Figure 8-6 on page 8-8.
8.1.2 Step 2: Configure the system

Launch SOPC Builder by selecting Tools → SOPC Builder from the menu. This is the tool you must use to build the Cortex-M1 based system, including the testbench.

Enter CortexM1_ExampleDesign in the Create New System dialog as shown in Figure 8-7 on page 8-9.
Figure 8-7 SOPC Builder Create New System dialog

Click **OK**.

The Cortex-M1 component must be added onto the IP Search Path as described in the *Cortex-M1 FPGA Development Kit Installation Guide*.

Add the ARM Cortex-M1 processor component by selecting the component from the tree view on the left side of the SOPC Builder window and click **Add** as shown in Figure 8-8 on page 8-10.
Configuring Example SOPC System

The ARM Cortex-M1 Processor settings window appears as shown in Figure 8-9 on page 8-11.

Configure the processor with following parameters:

- Debug enabled = checked
- Debug node ID = 0
- Number of IRQs = 8
- ITCM size = 32 Kbytes, Read only = checked
- Initialize ITCM contents = checked, From file: = itcm.hex
- DTCM size = 16 Kbytes
- Initialize DTCM contents = not checked, From file: = dtcm.hex (disabled).

Figure 8-8 Initial SOPC Builder screen
Click **Finish**.

You have instantiated the Cortex-M1 component in the example SOPC system design, as shown in Figure 8-10 on page 8-12. Rename the Cortex-M1 component to `arm_cortexm1_inst` by right-clicking on the instantiated component and selecting **Rename**.
Add a parallel IO block that connects to the bank of four LEDs on the Cyclone III Starter Board.

Select the **PIO (Parallel I/O)** block from the tree view on the left, as shown in Figure 8-10, and click **Add**.

The dialog window in Figure 8-11 on page 8-13 appears.

Configure the width of the PIO port to be 4. This corresponds to the four LEDs on the starter board. The remaining configurations are defaults.
Click Finish.

You have added the PIO module for the LEDs to the system. Make the following configurations:

- Rename the instantiated module to something more descriptive, such as LED_IF. To do this, right-click on the default instantiated module name pio and select Rename.
• Connect the Avalon Slave interface of the LED_IF to the Avalon Master interface of the ARM Cortex-M1 processor component by clicking the graphical connection icon in the Connection column.

• Configure the base address of the LED_IF component to be in the peripheral space as defined in the Cortex-M1 User Guide. To do this, double-click in the Base column of the LED_IF component, and change the default address value to 0xa0000000.

Figure 8-12 shows the system after these configuration changes.

![Figure 8-12 LED_IF component instantiated in SOPC Builder](image)

Figure 8-12 LED_IF component instantiated in SOPC Builder

You must add another PIO component for the bank of four user switches on the starter board.

Configure the PIO as follows:

• Configuration Dialog (Basic Settings):
  — Width = 4
— Direction = **Input Ports Only**.

- Configuration Dialog (Simulation):
  — select **Hardwire PIO inputs in test bench**
  — 0x0 in the box labeled **Drive Inputs to**.

Click **Finish** on the configuration dialog window to instantiate the Switch PIO block.

In the SOPC Builder, set:
- SOPC Module Name = **Switch_IF**
- Connection, and Base = 0xa0001000.

Connect the Switch_IF slave interface to the Cortex-M1 master interface by clicking in the **Connection** column next to the Switch_IF. Figure 8-13 shows the SOPC configuration window after you have added the Switch_IF.

![Figure 8-13 Switch_IF component instantiated in SOPC Builder](image)

Add an Interval Timer to the system.
Select the Interval Timer component from the tree view on the left side of the SOPC Builder window.

Click **Add**.

Keep the default settings in the configuration dialog and click **Finish**.

Make the following configuration changes to the instantiated timer:

- **Module Name** = **Timer**
- **Connection, and Base** = **0xa0002000**.

_____ **Note** ________

The timer has an interrupt output that is automatically connected to IRQ input vector (location 0) of the processor.

Figure 8-14 shows the SOPC configuration after you have added the Timer.
Add a JTAG UART to the system. Select the JTAG UART component from the tree view on the left side of the SOPC Builder window.

Click **Add**.

Keep the default settings, in the configuration dialog, and click **Finish**.

Make the following configuration changes to the instantiated UART:

- **Module Name** = **USB_UART**
- **Connection, and Base** = **0xa0003000**.

The UART has an interrupt output that is automatically connected to IRQ input vector (location 1) of the ARM Cortex-M1 processor.

Figure 8-15 shows the SOPC configuration window after you have added the JTAG_UART.

![Figure 8-15 JTAG_UART component instantiated in SOPC Builder](image-url)
Add a PLL module. The only purpose is to generate a phase-shifted output clock that connects to the external SSRAM chip on the board. You can configure the PLL to compensate for off-chip clock skew. The Cyclone III Starter Board has been characterized for a 50MHz clock and a clock phase shift of -3.3ns is required.

In the PLL configuration dialog window, click the Launch Altera's ALTPLLL MegaWizard button. See Figure 8-16.

![Launch PLL MegaWizard](image)

**Figure 8-16 Launch PLL MegaWizard**

The first MegaWizard dialog window appears, as shown in Figure 8-17 on page 8-19. Check that the input clock is set to 50MHz. This is the frequency of the on-board oscillator.
Click the Inputs/Lock tab and select **Create 'locked' output**, as shown in Figure 8-18 on page 8-20. The locked output is connected to the system reset in the top level of the design.
Click the **Output Clocks** tab, and set **Clock Phase shift** to -3.33ns, as shown in Figure 8-19 on page 8-21.
Click **Finish** to exit the PLL configuration dialog then click **Finish** to exit status dialog.

PLL MegaWizard configurations are now complete. Click **Finish** in the PLL dialog to complete the PLL instantiation in SOPC Builder.

Figure 8-20 on page 8-22 shows the SOPC configuration after you have added the PLL.
Make the following configuration changes to the instantiated PLL:

- Module Name = Ext_SRAM_PLL
- Connection, and Base = 0xa0004000

**Figure 8-20 PLL instantiated in SOPC Builder**

Add a System ID peripheral. Select the System ID Peripheral from the tree view on the left and click **Add**. There are no configurations to make for the System ID Peripheral.

Click **Finish** on the dialog box.

Make the following configuration changes to the instantiated System ID:

- Module Name: sysid
- Clock: clk
- Base: 0xa0005000

Figure 8-21 on page 8-23 shows the SOPC configuration after you have added the System ID.
Add a Flash Memory peripheral. Select the Flash Memory (CFI) from the tree view on the left side of the SOPC Builder window.

Click **Add**.

On the configuration dialog, change the **Preset** field to Intel 128P30, as shown in Figure 8-22 on page 8-24. This corresponds to the Flash chip on the Cyclone III Starter Board.
Click **Finish**.

Make the following configuration changes to the Flash Memory peripheral:

- Module Name = **Ext_FLASH_Ctrl**
- **Clock** = **clk**
- **Base** = **0x60000000**.

--- **Note** ---

This base address is in memory space, not peripheral space for the processor.

---

Add a SSRAM Memory peripheral. Select Cypress CY7C1380C SSRAM from the tree view on the left side of the SOPC configuration window.

Click **Add**.

On the SSRAM configuration dialog, change the Memory size (MBytes) to 1, as shown in Figure 8-23 on page 8-25. This corresponds to the size of the SSRAM memory on the Cyclone III Starter Board.
Figure 8-23 SSRAM settings dialog

Click Finish.

Make the following configuration changes to the SSRAM:

- Module Name = Ext_SRAM_Ctrl
- Clock = clk
- Base = 0x20100000.

Note: This base address is in memory space, not peripheral space for the processor.
To connect the external resources such as Flash and SSRAM to the System Interconnect Fabric, you must instantiate an Avalon-MM Tristate Bridge component. This component acts as a slave device to the System Interconnect Fabric and as a master to the external resources.

To add an Avalon-MM Tristate Bridge component, select the Avalon-MM Tristate Bridge component from the tree view on the left side of the SOPC configuration window and click **Add**.

On the Avalon MM Tristate Bridge dialog, keep the default settings as shown in Figure 8-24.

![Avalon-MM Tristate Bridge settings dialog](image)

**Figure 8-24 Tristate Bridge settings dialog**

Click **Finish**.

You must make the connections to the SSRAM and flash components as shown in Figure 8-25 on page 8-27.
Make the following configurations to the Tristate Bridge:

- Clock = clk.

![Figure 8-25 Tristate Bridge connections to SSRAM and flash](image)

Note

You must connect the Tristate Bridge Avalon slave port to the ARM Cortex-M1 Processor Master port, and its Tristate Master port to the Flash and the SSRAM modules.

Note

Edit the configuration settings by right-clicking on the module name and selecting **Edit**.

Note

This brings up the dialog configured in an earlier step.

Click **Next**.
On the Cyclone III Starter Board, the external FLASH and external SSRAM share the same address bus. Indicate this by selecting both address check boxes. See Figure 8-26.

![Figure 8-26 Tristate Bridge connection settings dialog](image)

Click Finish.

**8.1.3 Step 3: Generate the testbench**

Click Next on the bottom of the SOPC Builder dialog box, as shown in Figure 8-27.

![Figure 8-27 Bottom of SOPC Builder screen](image)
This brings up the **System Generation** tab. To make SOPC Builder automatically create a testbench for the design, select the **Simulation** checkbox and click **Generate**.

### 8.1.4 Step 4: Create the top level wrapper

Create the top-level wrapper around the generated SOPC system. This release includes a template for the FPGA top level file named `CortexM1_ExampleDesign_Top.v` in the default `ExampleDesign` directory. Copy this file into your project directory.

--- Note ---

This is a transparent wrapper that instantiates the top-level SOPC module.

---

The following code extract includes the signals that have been modified in the example template provided. You will also need to make these assignments if you create your own top-level system.

```vhdl
//wire declarations
wire [23:0] tristate_bridge_address_internal;
wire EDBGRQ_to_the_arm_cortexm1_inst_internal;
wire NMI_to_the_arm_cortexm1_inst_internal;
wire [3:0] out_port_from_the_LED_IF_internal;
wire [3:0] in_port_to_the_Switch_IF_internal;
wire reset_n_internal;
wire locked_from_the_Ext_SRAM_PLL;

//assignments
assign tristate_bridge_address[23:1] = tristate_bridge_address_internal[23:1];
assign EDBGRQ_to_the_arm_cortexm1_inst_internal = (EDBGRQ_to_the_arm_cortexm1_inst & 1'b0);
assign NMI_to_the_arm_cortexm1_inst_internal = (NMI_to_the_arm_cortexm1_inst & 1'b0);
assign in_port_to_the_Switch_IF_internal = ~in_port_to_the_Switch_IF;
assign out_port_from_the_LED_IF = ~out_port_from_the_LED_IF_internal;
assign reset_n_internal = reset_n & locked_from_the_Ext_SRAM_PLL;
```

The following signals inputs to the processor are not used in the example system and must be driven LOW:
- **EDBGRQ**
- **NMI**.

The `out_port_from_the_LED_IF` are inverted to allow a logic 1 to turn on the LED. The `in_port_to_the_Switch_IF` is inverted so a pressed switch produces a logic 1 in the internal FPGA design.
The reset_n signal is anded with locked_from_the_Ext_SRAM_PLL to produce the FPGA reset_n_internal signal.

--- Note

In other Altera FPGAs, these signals are held LOW with pull-down resistors in the FPGA. The Cyclone III only has pull-ups. Pull-downs are not available to the user.

8.1.5 Step 5: Add the top-level file to the project

You must update the project settings to select the new top-level entity, CortexM1_ExampleDesign_Top. Add the top-level file to the project by selecting Project → Add/Remove files in Project... from the menu to bring up the Files pane of the Quartus II project Settings window as shown in Figure 8-28 on page 8-31.
Select the top-level file, CortexM1_ExampleDesign_Top.v by pressing the ... button next to the File name field and browsing for the file. After opening the file in the file selection dialog, click the Add button in the Files pane of the settings window to add the file to the project.

You must also select CortexM1_ExampleDesign_Top as the top-level design entity. Select General from the Category list in the project Settings window. In the General pane which now appears, replace the contents of the Top-level entity field with CortexM1_ExampleDesign_Top and press OK.
8.1.6  **Step 6: Synthesize the design**

The design is ready to be synthesized. On the menu, select **Processing → Start → Start Analysis and Synthesis**. Analysis and synthesis takes a few minutes, depending on the PC.

8.1.7  **Step 7: Import the pin assignments**

You must create pin assignments for the project. The pin assignments from the provided project can be imported because they are the same as those for the project being built. To do this, select **Assignments → Import Assignments** from the menu. Browse to the provided CortexM1_ExampleDesign.qsf file, located in the original provided example system, by clicking the ... button.

Click **OK** as shown in Figure 8-29.

![Figure 8-29 Quartus II Import Assignments dialog](image)

This assigns the top-level I/O to specific pins in the design. If necessary, you can view and edit the pin assignments by selecting **Assignments → Pins** from the menu. This invokes the Altera Pin Planner.

--- **Note** ---

The pin assignments in the original CortexM1_ExampleDesign.qsf file were created using the Pin Planner tool.
8.1.8 Step 8: Configure the dual-purpose pins

Altera FPGAs permit some pins to be dual-purpose pins. Two pins in this design are dual purpose pins.

Select the Assignments → Device menu then click Device and Pin Options.

Select the Dual-Purpose Pin tab. If necessary, change the settings in the Value column as shown in Figure 8-30.

Click OK and then click OK again to return to Quartus.
8.1.9 Step 9: Import timing constraints

The design requires timing constraints. Copy CortexM1_ExampleDesign.sdc from the installed ExampleDesign directory to the working directory. Add this file to the Quartus II project by selecting Project → Add/Remove Files in Project from the menu, and adding the constraints file in the same way as you added the top-level design entity in Step 5. You might have to change the file type filter in the file selection dialog box to see the constraints file.

8.1.10 Step 10: Create TCM initialization files

You must create initialization files for the Cortex-M1 TCMs in the Quartus II project directory. You can copy itcm.hex from the ExampleDesign directory into your current project directory, or create new TCM initialization files using RealView MDK. See Application Note 213: Cortex-M1 TCM initialization in the ARM Cortex-M1 FPGA Development Kit for information about how to create your own TCM initialization files.

Note
If simulation is required, you must create memory initialization files in the SOPC Builder simulation directory, which is CortexM1_ExampleDesign_sim in the current project.

8.1.11 Step 11: Compile the design

The design is ready to be fully compiled. Invoke the design compilation process by selecting Processing → Start Compilation from the menu.

Note
Design compilation takes several minutes to complete.