Arm® Cortex®-M0 DesignStart™ Eval

Revision: r2p0

User Guide

DUI 0926B (ID092517)

Arm Cortex-M0 DesignStart Eval
User Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
23 December 2015	A	Non-Confidential	First release for r1p0
25 September 2017	B	Non-Confidential	First release for r2p0

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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.

Product Status

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

Web Address

`http://www.arm.com`

Contents
Arm Cortex-M0 DesignStart Eval User Guide

Preface

This preface introduces the *Cortex-M0 DesignStart Eval User Guide*. It contains the following sections:

About this book

This book describes the information required for system design and RTL simulation using Cortex-M0 DesignStart Eval.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Intended audience

This book is written for hardware engineers, software engineers, system integrators, and system designers, who might not have previous experience of Arm products, but want to run a complete example of a working system.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

This chapter describes Cortex-M0 DesignStart Eval and its features.

Chapter 2 *Functional Description*

This chapter describes the design and layout of Cortex-M0 DesignStart Eval.

Chapter 3 *Example System Testbenches*

This chapter describes the testbench components.

Chapter 4 *Using the Simulation Environment*

This chapter describes how to set up and run simulation tests.

Chapter 5 *Software Examples*

This chapter describes the example software tests and the device drivers.

Appendix A *Revisions*

This appendix describes the technical changes between released issues of this book.

Glossary

The *Arm Glossary* is a list of terms used in Arm documentation, together with definitions for those terms. The *Arm Glossary* does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm Glossary* http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html for more information.

Conventions

This book uses the conventions that are described in:

- *Typographical conventions* on page viii.
- *Timing diagrams* on page viii.
- *Signals* on page ix.

Typographical conventions

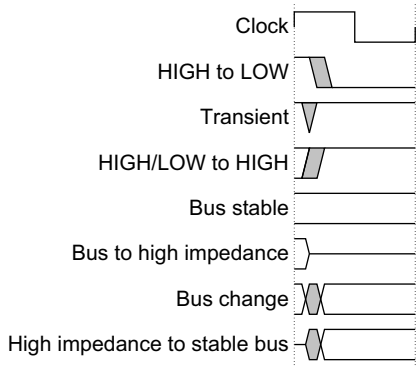
The following table describes the typographical conventions:

Style	Purpose
italic	Introduces special terminology, denotes 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.
`mono`space	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>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The figure named *Key to timing diagram conventions* explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to timing diagram conventions*. If a timing diagram shows a single-bit signal in this way then its value does not affect the accompanying description.

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.

Additional reading

This section lists publications by Arm and by third parties.

See Infocenter http://infocenter.arm.com, for access to Arm documentation.

See Arm CMSIS-Core http://www.arm.com/cmsis, for embedded software development resources including the *Cortex Microcontroller Software Interface Standard* (CMSIS).

Arm publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *Arm® Cortex®-M System Design Kit Technical Reference Manual* (DDI 0479).
- *Arm® Cortex®-M0 Devices Generic User Guide* (DUI 0497).
- *Arm® Cortex®-M0 Technical Reference Manual* (DDI 0432).
- *Arm® Armv6-M Architecture Reference Manual* (DDI 0419).
- *Arm® AMBA®3 AHB-Lite Protocol (v1.0) Specification* (IHI 0033).
- *Arm® Cortex®-M0 DesignStart™ Eval FPGA User Guide* (101125).
- *Arm® Versatile™ Express Cortex®-M Prototyping System (V2M-MPS2 and V2M-MPS2+ Technical Reference Manual* (100112).
- *Application Note AN502 Adapter for Arduino for the Cortex®-M Prototyping System (MPS2 and MPS2+)* (DAI0502).

The following confidential books are only available to licensees:

- *Arm® Cortex®-M0 Integration and Implementation Manual* (DII 0238).
- *Arm® Cortex®-M0 User Guide Reference Material* (DUI 0467).

Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on this product

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

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title.
- The number, DUI 0926B.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1
Introduction

This chapter introduces Cortex-M0 DesignStart Eval. It contains the following sections:

1.1 About Cortex-M0 DesignStart Eval

Cortex-M0 DesignStart Eval provides developers an easy way to simulate SoC designs based on the Cortex-M0 processor. It allows a system designer to design and test on a simulator and then proceed with hardware prototyping using an FPGA.

The Cortex-M0 DesignStart Eval package is aimed at developers who are new to Arm or have limited soft IP system design experience. The package includes the following:

- An Arm Cortex-M0 processor from DesignStart.
- An example system-level design for the Arm Cortex-M0 processor.
- An FPGA system design for the Arm Cortex-M0 processor, suitable for the Arm Cortex-M Prototyping System (MPS2+).
- Reusable AMBA components for system-level development.

The Cortex-M0 processor from DesignStart:

- Is a fixed configuration of the Cortex-M0 processor, enabling low-cost easy access to Cortex-M0 processor technology by offering a subset of the full product.
- Is delivered as a preconfigured and obfuscated, but synthesizable, Verilog version of the full Cortex-M0 processor. It is not intended for production silicon. See *Limitations* on page 1-5 for the Cortex-M0 processor from DesignStart configuration information.

The processor is flattened and obfuscated at the CORTEXM0INTEGRATION level, including debug. A Cortex-M0 DesignStart FPGA image is also available for system prototyping with the Arm® Versatile™ Express Cortex®-M Prototyping System, V2M-MPS2+. The Cortex-M0 DesignStart FPGA image offers an additional route for system design and prototyping on hardware. To purchase the prototyping system, go to the Arm website http://www.arm.com/mps.

The Cortex-M0 processor is a highly deterministic, low gate count, 32-bit processor that implements the Armv6-M architecture with zero deviation instruction determinism in zero wait-state memory systems. While the three-stage pipeline allows for very low area implementation, the Cortex-M0 processor is still capable of achieving performance figures of 2.33 CoreMarks/MHz. The Cortex-M0 processor programmers model is fully upwards compatible with the Cortex-M0+, Cortex-M3, Cortex-M4, and Cortex-M7 processors for portability.

For more information about:

- Using the Cortex-M0 DesignStart Eval system on the Cortex-M prototyping system, see *Arm® Cortex®-M0 DesignStart™ Eval FPGA user guide*.
- Programming the Cortex-M0 processor, see the *Arm® Cortex®-M0 Technical Reference Manual*.
- Software development on a Cortex-M0 device, see the *Arm® Cortex®-M0 Devices Generic User Guide*. This is a generic device user-level reference document.
- The AMBA components that the design kit uses, see the *Arm® Cortex®-M System Design Kit Technical Reference Manual*.
- The Arm architecture that the Cortex-M0 processor complies with, and the instruction set and exception model it uses, see the *Arm® Armv6-M Architecture Reference Manual*.
- The AHB-Lite master interface that the Cortex-M0 processor implements, see the *Arm® AMBA®3 AHB-Lite Protocol (v1.0) Specification*.

1.2 Cortex-M0 DesignStart Eval directory structure

Table 1-1 describes the main directories of the design kit.

Table 1-1 Main directory descriptions

Directory name	Directory contents
`Recovery`	FPGA image, including enctrypted bitfile, BIOS, configuration and software binaries.
`RevC`	FPGA design files (Verilog) and implementation flow.
`cores/cortexm0_designstart_r2p0`	Obfuscated Cortex-M0 Integration level.
`logical`	Cortex-M System Design Kit (CMSDK) Verilog components including AHB-Lite and APB infrastructure components, peripherals, the APB subsystem.
`smm_common`	Common FPGA components and peripherals
`software`	Software files. These include: • CMSIS compatible C header files. • Example program files for the example systems. • An example device driver.
`systems/cortex_m0_mcu`	Design files, testbench files, and simulation setup files for the CMSDK example system.
`systems/fpga_testbench`	Testbench files, and simulation setup files for the FPGA example system.
`documentation`	Documentation files.

Figure 1-1 on page 1-4 shows the location of the main directories for Cortex-M0 DesignStart Eval.

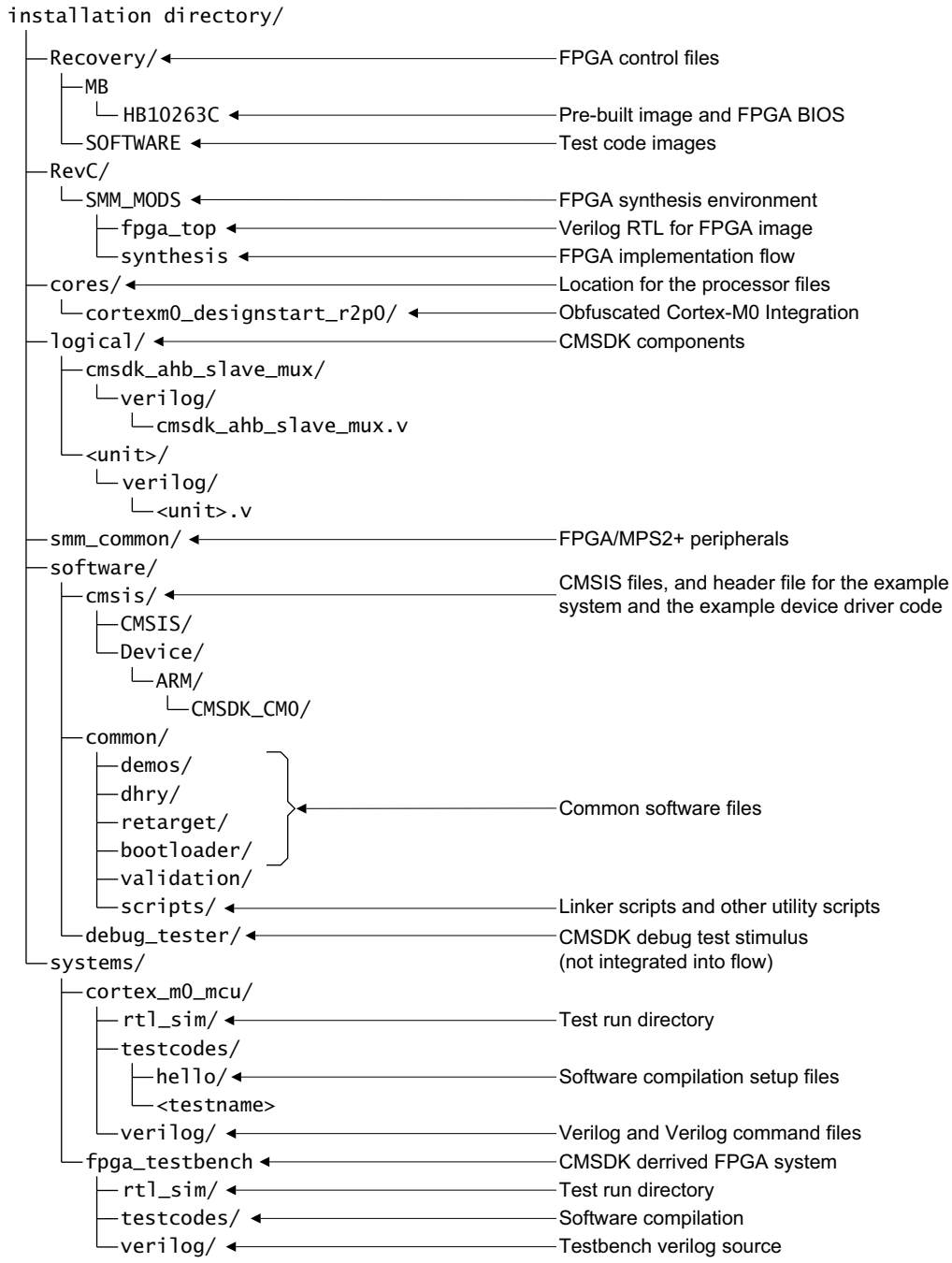


Figure 1-1 Main directories for Cortex-M0 DesignStart Eval

1.3 Limitations

You should not use the processor technology or the supporting deliverables as an indicator of what is received under a full technology license of the Arm Cortex-M0 processor. Cortex-M0 DesignStart Eval provides an easy entry into the Arm ecosystem, rather than a complete solution for all Cortex-M processor design scenarios.

Cortex-M0 DesignStart Eval does not support the implementation of the Cortex-M0 processor into silicon. Any implementation of the Cortex-M0 processor into silicon requires you to obtain Cortex-M0 DesignStart Pro, or take a full Cortex-M0 processor license from Arm.

A Cortex-M0 DesignStart Pro license offers the following:

- The Cortex-M0 processor.
- The Cortex-M System Design Kit.
- Simulation models for the Cortex-M0 processor
- A reference implementation flow for the Cortex-M0 processor.

If you are working on ASIC implementation, then Arm recommends that you license Cortex-M0 DesignStart Pro as early as possible.

1.3.1 Deliverables

The design kit does not include software compilation tools. You must license these products separately.

You must not modify the obfuscated Cortex-M0 processor (`cortexm0ds_logic.v`).

You are only permitted to redistribute the following files (modified or original), with the original headers unchanged, and any modifications clearly identified:

- `fpga_top.v`
- `fpga_system.v`
- `user_partition.v`
- `cmsdk_mcu_system.v`

You must not re-distribute any FPGA bit files or other representations of the design which are produced from Cortex-M0 DesignStart Eval.

You are expected to modify the test code to support any modifications you make to your design. You must not redistribute any test code or binaries from these deliverables unless it is developed using mbed source code.

1.3.2 Processor support

The Cortex-M0 DesignStart Eval product supports a specific version of the Cortex-M0 processor. This DesignStart Cortex-M0 processor has no configuration options.

Table 1-2 shows the differences in the features available in the full Cortex-M0 processor and the Cortex-M0 processor from DesignStart.

Table 1-2 Cortex-M0 processor and Cortex-M0 processor from DesignStart feature differences

Feature	Full Cortex-M0 processor	Cortex-M0 processor from DesignStart
Verilog code	Commented plain-text RTL	Flattened and obfuscated RTL
AMBA®3 AHB-Lite interface	Master and optional slave ports	Master port only
Armv6-M instruction set	Armv6-M instruction set support	Armv6-M instruction set support
Multiplier options	Fast single-cycle or small 32-cycle	Fast single cycle multiplier
Nested vectored interrupt controller (NVIC)	1-32 interrupt inputs	32 interrupt inputs only
Wake-up Interrupt Controller (WIC)	Optional	None
Architectural clock gating	Optional	None
24-bit system timer, SysTick	Optional reference clock	Reference clock supported
Hardware debugger interface	Optional Serial-Wire or JTAG	Serial-Wire only
Hardware debug support	Optional single step with up to four breakpoints, up to two watchpoints and PC sampling	Single step with four breakpoints, two watchpoints and PC sampling
Low-power signaling and domains	Optional state-retention power domains and power control signaling	SLEEPING, TXEV and RXEV signaling only

1.3.3 Endian support

The Cortex-M0 DesignStart Eval example system and its peripherals are *little-endian*.

1.3.4 Platform

This release of the Cortex-M0 DesignStart Eval supports Linux and Unix for the simulation process and FPGA synthesis. If you use Arm Keil *Microcontroller Development Kit* (MDK) for software development, you can install the Cortex-M0 DesignStart Eval in a location that is accessible from Linux, Unix, and Windows. Do this using one of the following procedures:

- Install the Cortex-M0 DesignStart Eval on a network drive that:
  - A Linux or Unix terminal can access.
  - Is mapped to a network drive on a Windows machine.
- Use a personal computer to do the following:
  - Install virtualization software and install a guest *Operating System* (OS).
  - Set up a shared folder to access the design kit through the host OS.
  - Install the Cortex-M0 DesignStart Eval in the shared folder.

Then compile the software with Keil MDK in the Windows environment, and run the simulations in the Linux or Unix environment.

To run the Cortex-M0 DesignStart Eval on other operating systems, modify the `makefiles` to meet your specific requirements.

Chapter 2
Functional Description

This chapter describes the design and layout of Cortex-M0 DesignStart Eval. It contains the following sections:

2.1 Example MCU system level design and design heirarchy

The systems/cortex_m0_mcu/verilog/tb_cmsdk_mcu.v example system is a simple microcontroller design, based on the example provided with the full CMSDK product. It differs in its interrupt connectivity (which is aligned with the FPGA platform). You should use this system if you are interested in RTL prototyping only, or wish to build your own system from scratch. It contains the following:

- A single Cortex-M0 processor.
- Internal program memory.
- SRAM data memory.
- Boot loader.
- The following peripherals:
  - Several timers.
  - *General Purpose Input Output* (GPIO).
  - *Universal Asynchronous Receiver Transmitter* (UART).
  - Watchdog timer.

Figure 2-1 shows the top level view of the example system.

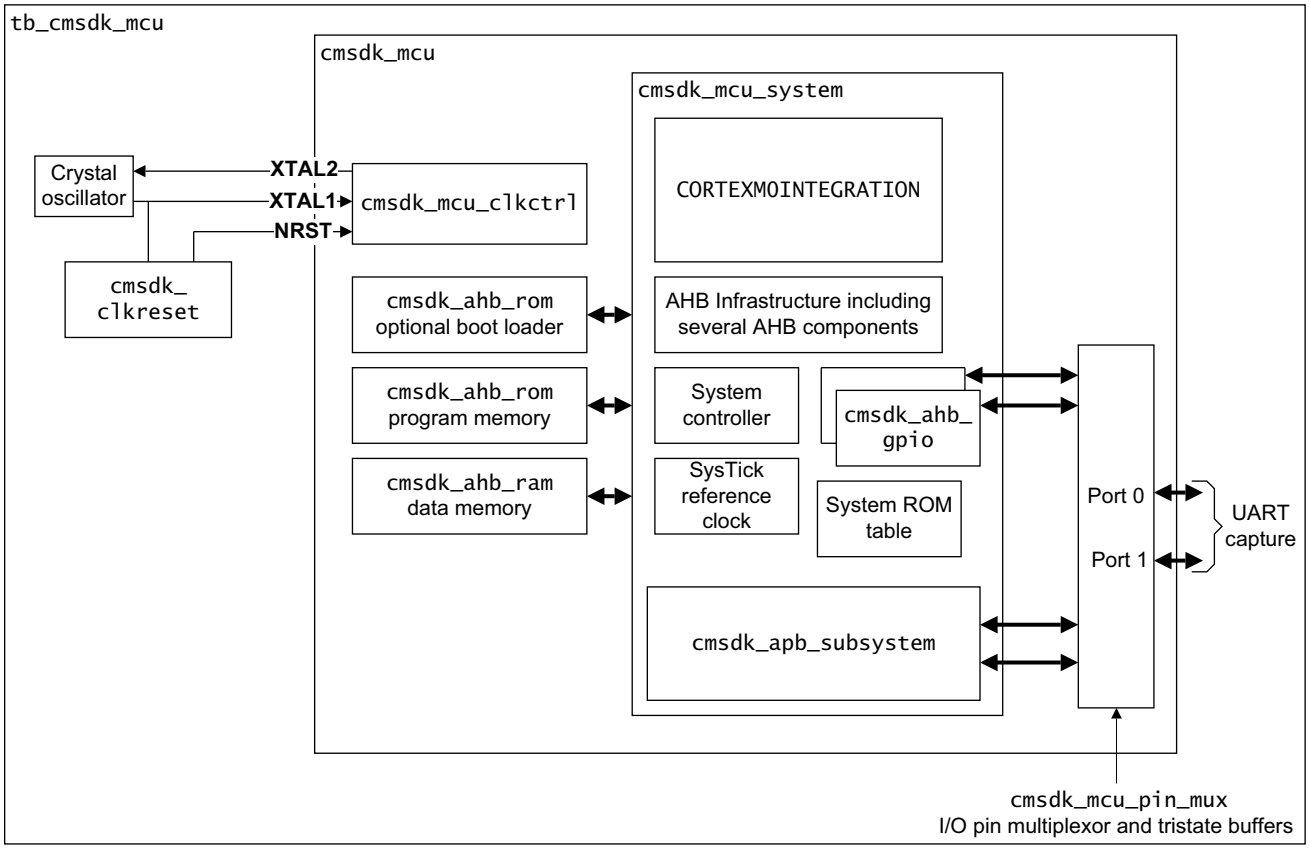


Figure 2-1 Example microcontroller system top level view

Table 2-1 describes the items that the microcontroller contains.

Table 2-1 Microcontroller items

Item	Description
`cmsdk_mcu`	The example microcontroller design. This level contains the behavioral memories and clock generation components.
`cmsdk_mcu_system`	The synthesizable level of the microcontroller design. This instantiates the Cortex-M0 processor.
`CORTEXM0INTERGRATION`	The Cortex-M0 integration layer. This is obfuscated and flattened code.
`cmsdk_apb_subsystem`	A subsystem of APB peripherals and APB infrastructure.
System controller	Contains programmable registers for system control, for example memory remap.
SysTick reference clock	SysTick reference clock generation logic.
`cmsdk_ahb_gpio`	A low-latency GPIO with an AHB interface. Each GPIO module provides 16 I/O pins.
`cmsdk_mcu_clkctrl`	The clock and reset generation logic behavioral model.
`cmsdk_mcu_pin_mux`	The pin multiplexor and tristate buffers for the I/O ports.
`cmsdk_ahb_rom`	A memory wrapper for the ROM to test the behavior of different implementations of memory. You can modify the Verilog parameters to change the implementation.
`cmsdk_ahb_ram`	A memory wrapper for the RAM to test the behavior of different implementations of memory. You can modify the Verilog parameters to change the implementation.
`cmsdk_ahb_cs_rom_table`	An example system level CoreSight ROM table that enables a debugger to identify the system as a Cortex-M0 based system.
`cmsdk_mcu_addr_decode`	Generates the **HSELS** for each memory mapped component based on the CMSDK address map.

Table 2-2 describes the items that are in the testbench but outside the microcontroller.

Table 2-2 Testbench items

Item	Descriptions
`cmsdk_clkreset`	Generates clock and reset signals. XTAL1 runs at 50MHz. It asserts **NRST** LOW for 5ns at the start of the simulation.
`cmsdk_uart_capture`	Captures the text message from UART2 and displays the message during simulation. It displays each line of the message after it receives a carriage return character. To reduce the simulation time, set the baud rate to be same as the clock frequency. You must set the UART to `high speed test mode`.

You can configure the system in a number of different ways.

The processor connects to the rest of the system through an AHB Lite interface.

Figure 2-2 on page 2-4 shows the interfaces of the Cortex-M0 example system.

Note

In this design, the DAP and WIC modules are flattened into the CORTEXM0INTEGRATION module. The DAP is configured for Serial Wire mode, and the WIC is configured as not present.

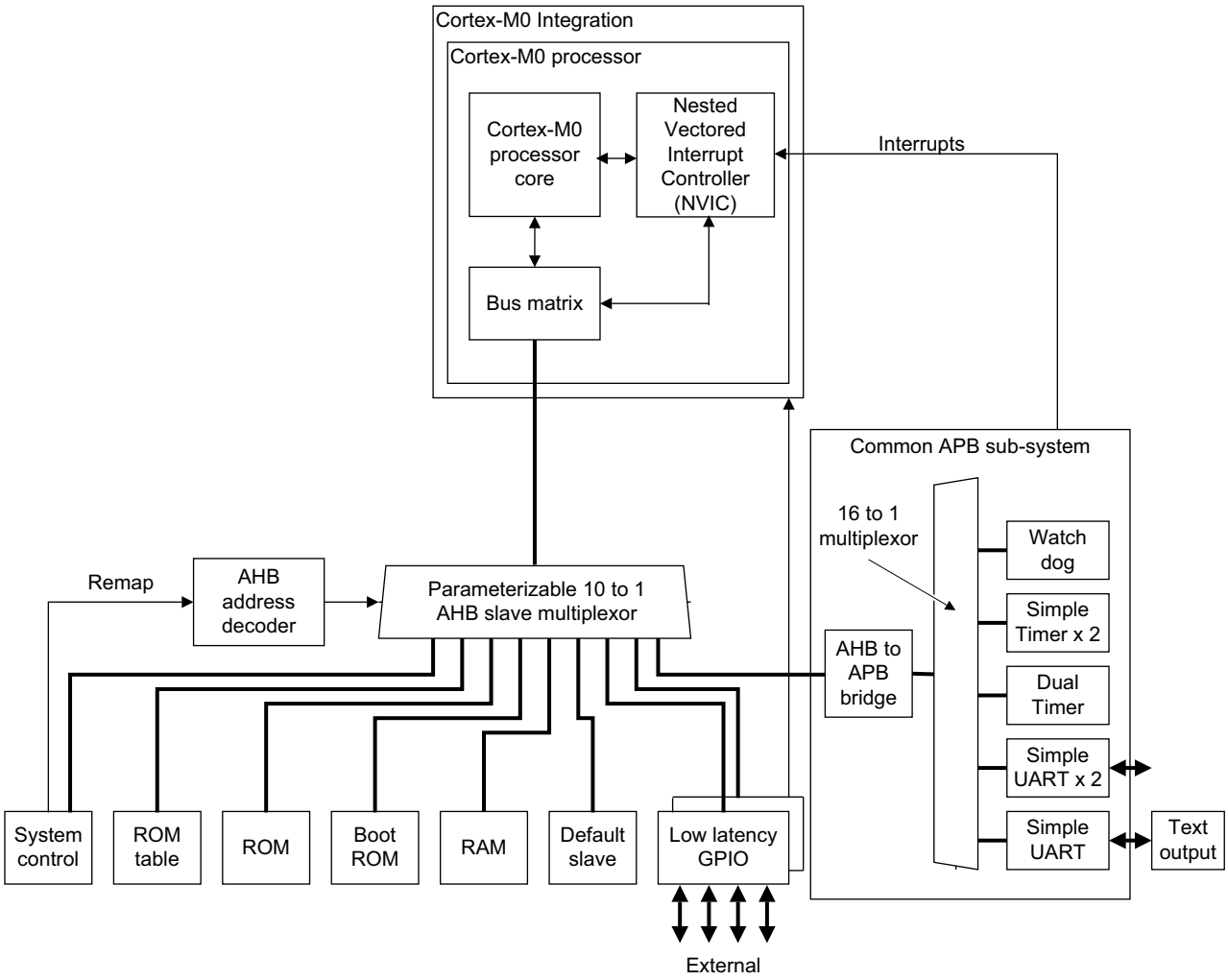


Figure 2-2 Cortex-M0 example system

Table 2-3 describes the peripheral components that the system design includes.

Table 2-3 Peripheral components

Item	Descriptions
`cmsdk_ahb_gpio`	Two low latency GPIO with AHB interfaces. Each GPIO module provides 16 I/O pins.
`cmsdk_apb_timer`	A 32-bit timer.
`cmsdk_apb_uart`	A UART.
`cmsdk_apb_watchdog`	A watchdog component that is compatible with the watchdog in the AMBA design kit.
`cmsdk_apb_dualtimers`	A dual timer module that is compatible with the dual timer in the AMBA design kit.

The APB peripherals are instantiated in the APB subsystem block.

2.2 Example FPGA system level design and design heirarchy

The fpga_testbench system uses a modified CMSDK system, and extends this with the peripherals required to support the interfaces present on the MPS2+ board. In the simulation, it uses the same obfuscated CORTEXM0INTEGRATION level as us used by the simpler cortex_m0_mcu system. Although the systems have a different structure, the software view of the FPGA system is a superset of the example MCU system.

The peripherals present in the FPGA system include:

- Zero Bus Turnaround RAMS (ZBT RAMS).
- SRAMs.
- Pseudo-SRAM (PSRAM).
- Colour LCD Screen.
- SPI interface for Arduino Shield Adaptor.
- I2C interface for Arduino Shield Adpator.
- UARTs for Arduino Shield Adaptor.

Figure 2-3 on page 2-6 shows the top level view of the example FPGA system.

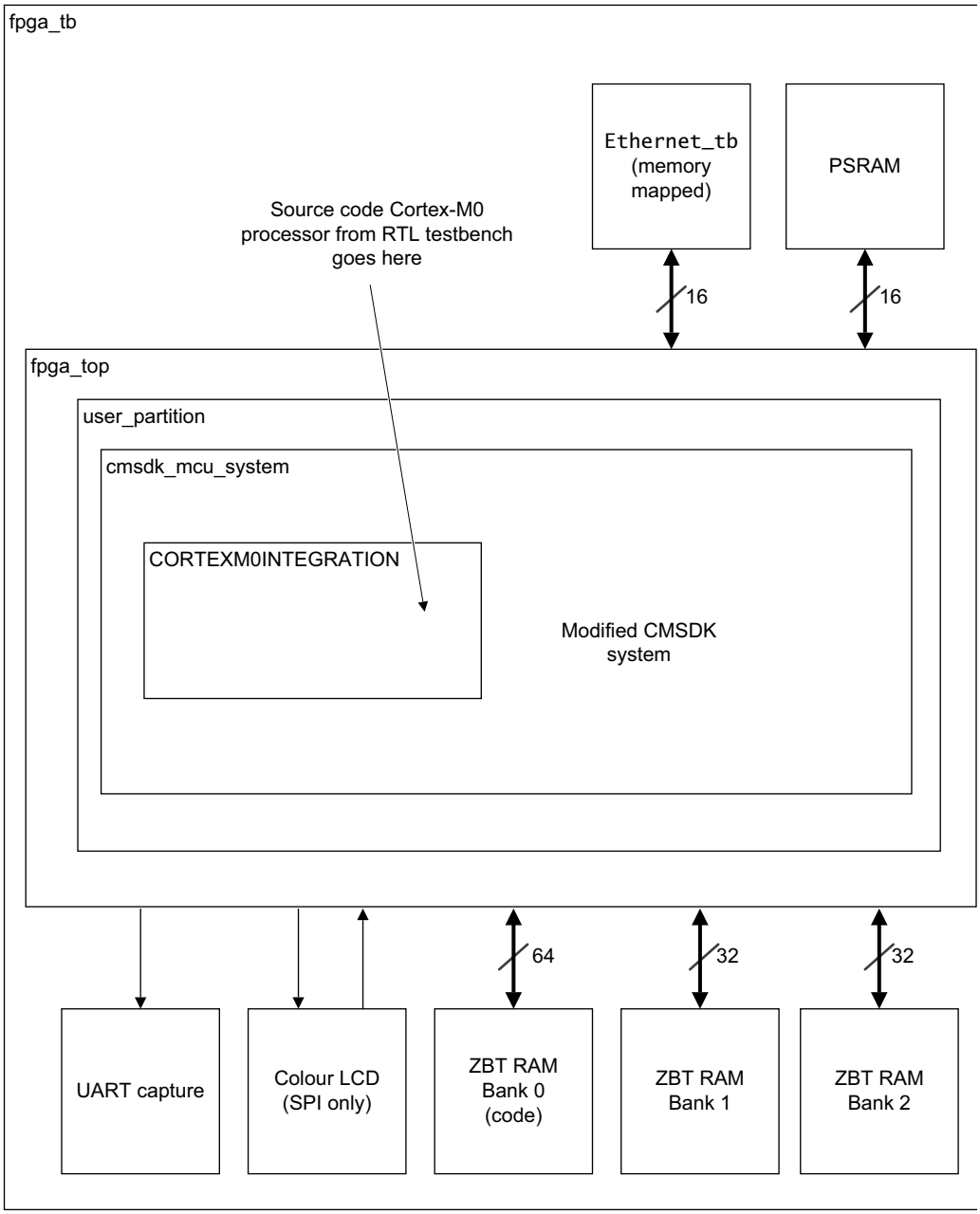


Figure 2-3 Example FPGA system top level view

There are differences between standard CMSDK system, Cortex-M3 DesignStart Eval and the memory/interrupt mapping of the Cortex-M0 DesignStart Eval FPGA. You can use this FPGA testbench to investigate the behavior of any system you design to target the MPS2+ platform, but you may need to update the testbench support for any particular peripheral interface you need to test.

2.3 Design files

This section describes the following design files that are included in Cortex-M0 DesignStart Eval:

- *Verilog files for the cmsdk_mcu example system*.
- *Verilog files for the cortex_m0_mcu testbench*.
- *Verilog files for the FPGA design* on page 2-8.
- *Verilog files for the FPGA testbench* on page 2-8.

2.3.1 Verilog files for the cmsdk_mcu example system

Table 2-4 describes the Verilog files that are included in the Cortex-M0 microcontroller.

Table 2-4 Verilog files for the Cortex-M0 microcontoller

File name	Description
`cmsdk_mcu.v`	Top level of the microcontroller
`cmsdk_mcu_defs.v`	Constant definitions and configuration definitions for the example microcontroller
`cmsdk_mcu_system.v`	Microcontroller system-level design
`cmsdk_mcu_sysctrl.v`	Programmable register block for system-level control
`cmsdk_mcu_stclkctrl.v`	SysTick reference clock generation logic
`cmsdk_mcu_clkctrl.v`	Clock and reset control
`cmsdk_mcu_pin_mux.v`	Pin multiplexer and tristate buffers for the I/O port
`cmsdk_mcu_addr_decode.v`	Generates the **HSELS** for each memory mapped component based on the CMSDK address map
`cmsdk_ahb_cs_rom_table.v`	CoreSight system level ROM table for CMSDK

2.3.2 Verilog files for the cortex_m0_mcu testbench

Table 2-5 describes the Verilog files that are included in the testbench.

Table 2-5 Verilog files for the Cortex-M0 microcontroller testbench

File name	Description
`tb_cmsdk_mcu.v`	Testbench of the example microcontroller
`cmsdk_clkreset.v`	Clock and reset generator
`cmsdk_uart_capture.v`	UART capture for text message display
`tbench_M0_DS.vc`	Verilog command file for Cortex-M0 DesignStart Eval

2.3.3 Verilog files for the FPGA design

Table 2-6 describes the major files for the FPGA design (not including the peripherals).

Table 2-6 Verilog files for the Cortex-M0 FPGA design

File name	Description
fpga_top.v	Top level of FPGA with I/O pins, PLL and ddr pad instances, reset control logic
fpga_system.v	Wrapper level
user_partition.v	Main system integration. Port muxing, memory device interfaces, gpio 2/3
cmsdk_mcu_system.v	System based on CMSDK with additional peripherals integrated
cmsdk_mcu_stclkctrl.v	Simple SysTick signal control
cmsdk_mcu_sysctrl.v	Simple system control peripheral, PMU support tied off
cmsdk_ahb_cs_rom_table.v	Standard CMSDK system ROM table
cmsdk_mcu_addr_decode.v	Address decode for FPGA version of cmsdk_mcu_system

2.3.4 Verilog files for the FPGA testbench

Table 2-7 describes the Verilog files that are included in the testbench.

Table 2-7 Verilog files included in the Cortex-M0 FPGA testbench

File name	Description
tb_fpga.v	Testbench top level
cmsdk_uart_capture_ard.v	UART capture/display module
tb_arduino_shield.v	Wrapper for Arduino Shield components
arduino_adaptor.v	Model of Arm adaptor for Arduino Shield
arduino_shield.v	Testbench model of Arduino shield
GS8160Z36DT.v	Memory model
core.v	Memory model component
SPI_EEPROM.v	Memory model
I2C_SRAM.v	Memory model
IS66WVE409616BLL.v	Memory model
scc_tb.v	SCC interface wrapper example

2.4 Processor file location

In a standard CMSDK environment, the location of the Verilog RTL files for the processors is a subdirectory called `cores`. For Cortex-M0 DesignStart Eval the path is:

```
cores/cortexm0_designstart_r2p0/logical/
```

This directory contains a configuration of the Cortex-M0 processor (version r0p0), arranged as a wrapper file `CORTEXM0INTEGRATION.v` and `cortexm0ds_logic.v` which contains the obfuscated processor. If you have licensed the full Cortex-M0 processor, you can directly replace this instance of the integration level by installing the full processor to `cores/at510_cortexm0_r0p0-03rel2`, and changing the search paths to refer to this directory.

See Table 1-2 on page 1-6 for the configuration which was used to generate this processor.

2.5 Configuration options

The example microcontroller system contains several configurable options. You use Verilog preprocessing definitions to set these options.

The file `cortex_m0_mcu/verilog/cmsdk_mcu_defs.v` contains the Verilog preprocessing definitions. To remove a definition, comment-out the line of Verilog code that describes the preprocessing definitions. The following table shows the Verilog preprocessing definitions.

Table 2-8 Verilog preprocessing definitions

Preprocessing macro	Descriptions
`ARM_CMSDK_BOOT_MEM_WS_N`	Defines the number of wait states for boot loader ROM non-sequential accesses. See the *Cortex-M System Design Kit Technical Reference Manual*.
`ARM_CMSDK_BOOT_MEM_WS_S`	Defines the number of wait states for boot loader ROM sequential accesses. See the *Cortex-M System Design Kit Technical Reference Manual*.
`ARM_CMSDK_ROM_MEM_WS_N`	Defines the number of wait states for program ROM non-sequential accesses. See the *Cortex-M System Design Kit Technical Reference Manual*.
`ARM_CMSDK_ROM_MEM_WS_S`	Defines the number of wait states for program ROM sequential accesses. See the *Cortex-M System Design Kit Technical Reference Manual*.
`ARM_CMSDK_RAM_MEM_WS_N`	Defines the number of wait states for RAM non-sequential accesses. See the *Cortex-M System Design Kit Technical Reference Manual*.
`ARM_CMSDK_RAM_MEM_WS_S`	Defines the number of wait states for RAM sequential accesses. See the *Cortex-M System Design Kit Technical Reference Manual*.

2.6 Memory map

This section describes the system memory maps. It contains the following sections:

- *AHB memory map*.
- *APB subsystem memory map* on page 2-12.

2.6.1 AHB memory map

The AHB memory map has a 4GB linear address range, but peripherals only use part of the memory space. If a bus master accesses an invalid memory location with a valid transfer, the default slave replies with an error response to the bus master.

The following files contain the address decoding logic for the two systems. In order to modify the memory map, you must modify the address decoding logic in these files.

- `systems/cortex_m0_mcu/verilog/cmsdk_mcu_addr_decode.v`
- `RevC/SMM_M0DS/fpga_top/verilog/cmsdk_mcu_addr_decode.v`
- `RevC/SMM_M0DS/fpga_top/verilog/user_partition.v`

If you require the example system program to execute from boot loader memory after power-up, set the boot loader option. This enables the system remap feature. After the boot loader starts, the program can switch off the remap feature to enable your program to execute from the start of the memory.

Table 2-9 describes the AHB memory map for the RTL example system and the FPGA system.

Table 2-9 AHB memory map

Address	Example system
`0xF0220000-0xFFFFFFFF`	Unused, except for the private peripheral bus addresses in the Cortex-M0.
`0xF0210000-0xF021FFFF` (64KB)	Unused
`0xF0201000-0xF021FFFF`	Unused.
`0xF0200000-0xF0200FFF` (4KB)	Unused.
`0xF0000401-0xF01FFFFF`	Unused.
`0xF0000000-0xF0000400` (4KB)	System ROM table.
`0x41110000-0xEFFFFFFF`	Unused, except for the private peripheral bus addresses in the Cortex-M0.
`0x41100000-0x4110FFFF`	RTL example: Unused FPGA: VGA Image
`0x41000000-0x4100FFFF`	RTL example: Unused FPGA: VGA Console
`0x40200000-0x402FFFFF`	RTL example: Unused FPGA: Memory mapped ethernet interface
`0x40020000-0x4002FFFF`	APB subsystem for FPGA system
`0x4001F000-0x4001FFFF` (4KB)	System controller registers.
`0x40012000-0x4001EFFF`	Unused.
`0x40013000-0x40013FFF`	CMSDK AHB GPIO #3 (only present in FPGA system)

Table 2-9 AHB memory map (continued)

Address	Example system
0x40012000-0x40012FFF	CMSDK AHB GPIO #2 (only present in FPGA system)
0x40011000-0x40011FFF (4KB)	CMSDK AHB GPIO #1
0x40010000-0x40010FFF (4KB)	CMSDK AHB GPIO #0
0x40000000-0x4000FFFF (64KB)	CMSDK subsystem APB peripherals
0x22000000-0x3FFFFFFF	Unused.
0x21000000-0x21FFFFFF	PSRAM (FPGA system only) 16 MB
0x20800000-0x20FFFFFF	Unused
0x20000000-0x207FFFFF	RTL example: (64KB) RAM FPGA: ZBTSRAM 2 & 3, lower 4MB region implemented
0x01010000-0x1FFFFFFF	Unused.
0x01000000-0x0100FFFF (64KB)	Optional boot loader memory. Actual size 4KB, access above 4KB are aliased
0x00010000-0x00FFFFFF	Unused.
0x00400000-0x007FFFFF	RTL example: Unused FPGA: Alias of ZBTSRAM1
0x00000000-0x003FFFFF	RTL example: (64KB) Program memory FPGA: Lower 32KB can be mapped to BlockRam, remainder is ZBTSRAM1

2.6.2 APB subsystem memory map

Table 2-10 describes the peripherals in the APB subsystem. These are implemented in the `cmsdk_apb_sybsystem_cm0ds` module. Any access to a 'Not Used' location returns a read value of `0x00000000`.

Table 2-10 APB subsystem peripherals

Address	Item	Notes
0x4000F000-0x4000FFFF	APB expansion port 15	Not used, reserved for micro-DMA controller configuration port
0x4000E000-0x4000EFFF	APB expansion port 14	Not used
0x4000D000-0x4000DFFF	APB expansion port 13	Not used
0x4000C000-0x4000CFFF	APB expansion port 12	Not used
0x4000B000-0x4000BFFF	APB test slave	For validation of AHB to APB bridge, remove for real implementation
0x4000A000-0x4000AFFF	APB expansion port 10	Not Used
0x40009000-0x40009FFF	UART4	-
0x40008000-0x40008FFF	Watchdog	-
0x40007000-0x40007FFF	UART3	-
0x40006000-0x40006FFF	UART2	

Table 2-10 APB subsystem peripherals (continued)

Address	Item	Notes
0x40005000-0x40005FFF	UART1	-
0x40004000-0x40004FFF	UART0	Used to retarget STDOUT in simulations and on FPGA.
0x40003000-0x40003FFF	APB expansion port 3	Not Used
0x40002000-0x40002FFF	Dual timer	-
0x40001000-0x40001FFF	Timer1	-
0x40000000-0x40000FFF	Timer0	-

For more information on the APB subsystem, see the *Cortex-M System Design Kit Technical Reference Manual*.

2.6.3 FPGA System Secondary APB

Table 2-11 describes the peripherals in the second APB subsystem which is only present in the FPGA system. These are implemented in the fpga_apb_subsystem module. Any access to a 'Not Used' location returns a read value of 0x00000000 (or an ERROR response in the RTL system where there is no AHB decode for this memory region).

Table 2-11 Second APB subsystem peripherals (FPGA)

Address	Notes
0x4002F000-0x40020FFF	SSC Registers
0x4002E000-0x4002EFFF	Reserved
0x4002D000-0x4002DFFF	Reserved
0x4002C000-0x4002CFFF	Reserved
0x4002B000-0x4002BFFF	Reserved
0x4002A000-0x4002AFFF	Shield1 I2S
0x40029000-0x40029FFF	Shield0 I2S
0x40028000-0x40028FFF	FPGA System control and IO control
0x40027000-0x40027FFF	SPI4 (Shield 1)
0x40026000-0x40026FFF	SPI3 (Shield 0)
0x40025000-0x40025FFF	SPI2 (Shield ADC)
0x40024000-0x40024FFF	Audio I2S (MPS2+ hardware audio)
0x40023000-0x40023FFF	SBCon (Audio Configuration)
0x40022000-0x40022FFF	SBCon (LCD module touch interface)
0x40021000-0x40021FFF	PL022 SPI for LCD Display
0x40020000-0x40020FFF	PL022 SPI

For more information on the APB subsystem, see the *Cortex-M System Design Kit Technical Reference Manual*.

2.7 System controller

This section describes the system controller. It contains the following sections:

- *About the system controller*.
- *System controller block diagram*.
- *Programmers model* on page 2-15.

2.7.1 About the system controller

The example system contains a simple system controller that provides:

- The ability to enable an automatic reset if the system locks up.
- Information about the cause of the last reset.

2.7.2 System controller block diagram

Figure 2-4 shows the example system controller.

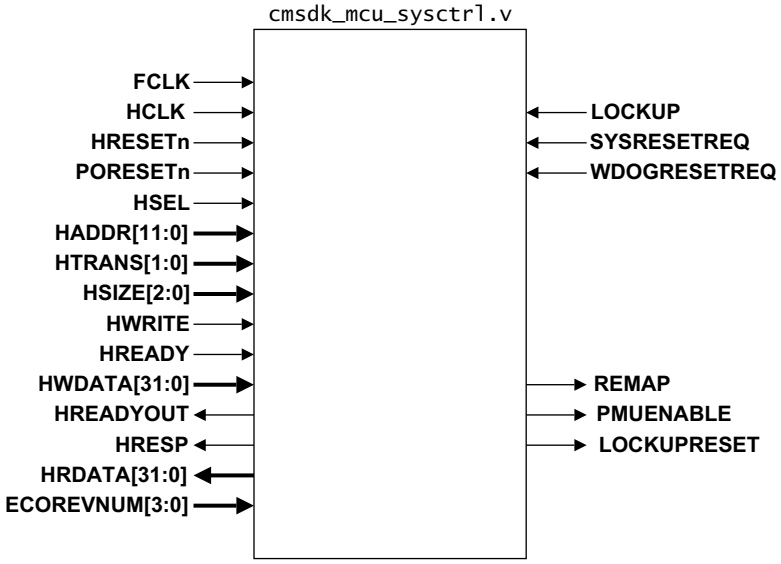


Figure 2-4 Example system controller

Table 2-12 shows the non-AHB signals of the system controller.

Table 2-12 Example system controller non-AHB signals

Signals	Descriptions
LOCKUP	Tells the RSTINFO register that the cause of a system reset is because the processor enters the lockup state
SYSRESETREQ	Enables a status register to capture the System Reset Request event
WDOGRESETREQ	Enables a status register to capture the Watchdog Reset Request event
REMAP	Enables the memory remap feature
PMUENABLE	Enables the PMU for the *WakeUp Interrupt Controller* (WIC) mode deep sleep operation
LOCKUPRESET	Enables the clock and reset controller to generate a system reset automatically if the system locks up

The design provides a 4-bit **ECOREVNUM** input that is connected to peripheral ID register 3. This would be used by a design taken to manufacture as a way to identify late ECO changes to the design. For Cortex-M0 DesignStart Eval, this signal should be tied LOW.

2.7.3 Programmers model

Table 2-13 describes the system controller programmers model.

Table 2-13 System controller programmers model

Address	Name	Type	Reset	Descriptions
0x4001F000	REMAP	RW	1	Bit 0: **1** Enable remap feature. **0** Disable remap feature. Software symbol: CMSDK_SYSCON->REMAP
0x4001F004	PMUCTRL	RW	0	Bit 0: **0** Disable PMU. This bit is Read Only in Cortex-M0 DesignStart. Software symbol CMSDK_SYSCON->PMUCTRL
0x4001F008	RESETOP	RW	0	Bit 0: **1** Automatically generates system reset if the processor is in the LOCKUP state. **0** Does not automatically generate reset when the processor is in the LOCKUP state. Software symbol CMSDK_SYSCON->RESETOP
0x4001F00C	-	-	-	Reserved
0x4001F010	RSTINFO	RW	0	Bit 2 - If 1, processor LOCKUP caused the reset. Bit 1 - If 1, Watchdog caused the reset. Bit 0 - If 1, SYSRESETREQ caused the reset. Write 1 to each bit to clear. Software symbol CMSDK_SYSCON-> RSTINFO
0x4001FFD0	PID4	RO	0x04	Peripheral ID 4. [7:4] Block count. [3:0] jep106_c_code.
0x4001FFD4	PID5	RO	0x00	Peripheral ID 5, not used.
0x4001FFD8	PID6	RO	0x00	Peripheral ID 6, not used.
0x4001FFDC	PID7	RO	0x00	Peripheral ID 7, not used.
0x4001FFE0	PID0	RO	0x26	Peripheral ID 0. [7:0] Part number.
0x4001FFE4	PID1	RO	0xB8	Peripheral ID 1. [7:4] jep106_id_3_0. [3:0] Part number[11:8].
0x4001FFE8	PID2	RO	0x1B	Peripheral ID 2. [7:4] revision. [3] jedec_used. [2:0] jep106_id_6_4.

Table 2-13 System controller programmers model (continued)

Address	Name	Type	Reset	Descriptions
0x4001FFEC	PID3	RO	0x-0	Peripheral ID 3. [7:4] ECO revision number. [3:0] Customer modification number.
0x4001FFF0	CID0	RO	0x0D	Component ID 0.
0x4001FFF4	CID1	RO	0xF0	Component ID 1 (PrimeCell class).
0x4001FFF8	CID2	RO	0x05	Component ID 2.
0x4001FFFC	CID3	RO	0xB1	Component ID 3.

The **PORESETn** signal resets the RSTINFO register. The **HRESETn** signal resets all the other resettable registers.

2.8 I/O pins

The example microcontroller has two 16-bit I/O ports and several debug signal connections. You can switch several I/O port pins to an alternate function.

See the *Cortex-M0 DesignStart Eval FPGA User Guide* for details of the I/O when using the FPGA system.

Figure 2-5 shows the interface of the example microcontroller.

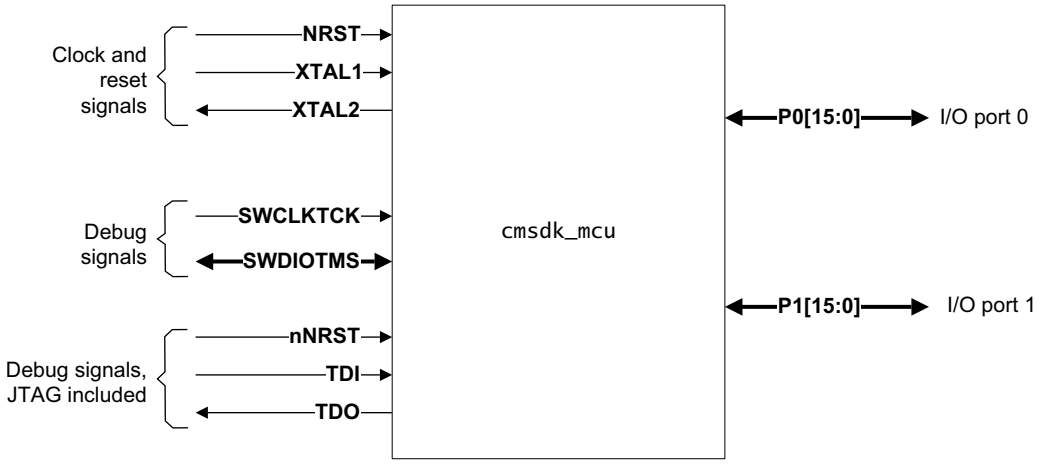


Figure 2-5 Example microcontroller interface

Table 2-14 describes the I/O of the example *MicroController Unit* (MCU).

Table 2-14 Example MCU I/O

Signal	Direction	Description
XTAL1	Input	Crystal oscillator
XTAL2	Output	Crystal oscillator feedback
NRST	Input	Reset, active LOW
P0[15:0]	Bidirectional	GPIO
P1[15:0]	Bidirectional	GPIO
nTRST	Input	JTAG reset, active LOW[a]
TDI[a]	Input	JTAG data in[a]
SWDIOTMS	Bidirectional	Serial Wire Data or JTAG TMS
SWCLKTCK	Input	Serial Wire clock or JTAG clock
TDO[a]	Output	JTAG data out[a]

a. This signal is inactive unless you use the full Cortex-M0 processor, and configure it for JTAG mode.

Table 2-15 shows the alternate functions of the GPIO1 and GPIO1[n] ports that support pin multiplexing.

Table 2-15 GPIO alternate functions

Pin	Alternate function
GPIO[15:10]	No alternate function.
GPIO[9]	Timer 1 EXTIN. Always use as timer 1 external input. The GPIO 1 alternate function setting has no effect.
GPIO[8]	Timer 0 EXTIN. Always use as timer 0 external input.The GPIO 1 alternate function setting has no effect.
GPIO[7]	TSTART to MTB.
GPIO[6]	TSTOP to MTB.
GPIO[5]	UART2 TXD.
GPIO[4]	UART2 RXD. Always use as UART input. The GPIO 1 alternate function setting has no effect.
GPIO[3]	UART1 TXD.
GPIO[2]	UART1 RXD. Always use as UART input. The GPIO 1 alternate function setting has no effect.
GPIO[1]	UART0 TXD.
GPIO[0]	UART0 RXD. Always use as UART input. The GPIO 1 alternate function setting has no effect.

Before you use the I/O pins for alternate functions, you might want to program the corresponding GPIO alternate function registers. This step might not be necessary when you use the alternate function as an input.

2.9 Interrupts and event functions

The example system contains:

- 32 *Interrupt Request* (IRQ) lines.
- One *NonMaskable Interrupt* (NMI).
- One event signal.

Note

Cortex-M0 DesignStart Eval only supports 32 interrupts.

2.9.1 Interrupt assignments

Table 2-16 describes the interrupt assignments.

Table 2-16 Interrupt assignments

IRQ/NMI	Device
NMI	Watchdog
0	UART 0 receive interrupt
1	UART 0 transmit interrupt
2	UART 1 receive interrupt
3	UART 1 transmit interrupt
4	UART 2 receive interrupt
5	UART 2 transmit interrupt
6	UART3 Rx, GPIO0, GPIO2 combined
7	UART3 Tx, GPIO1, GPIO3 combined
8	Timer 0
9	Timer 1
10	Dual timer
11	SPI0, SPI1, SPI2, SPI3, SPI4 combined interrupt
12	Combined UART overflow interrupts
13	Ethernet interrupt (on FPGA system)
14	Audio I2S Interrupt (on FPGA system)
15	Touch Screen interrupt (on FPGA) and DMA interrupt if present.
16	GPIO 0 bit 0 and UART4 Rx combined
17	GPIO 0 bit 1 and UART4 Tx combined
18-31	GPIO 0 bit 15 to GPIO 0 bit 2 individual interrupts

2.9.2 Interrupt synchronization

If a peripheral generates an interrupt signal in a clock domain that is asynchronous to the processor clock, you must synchronize the interrupt signal to the processor clock domain before you connect it to the **NVIC** of the processor. Figure 2-6 shows an example circuit that performs this synchronization.

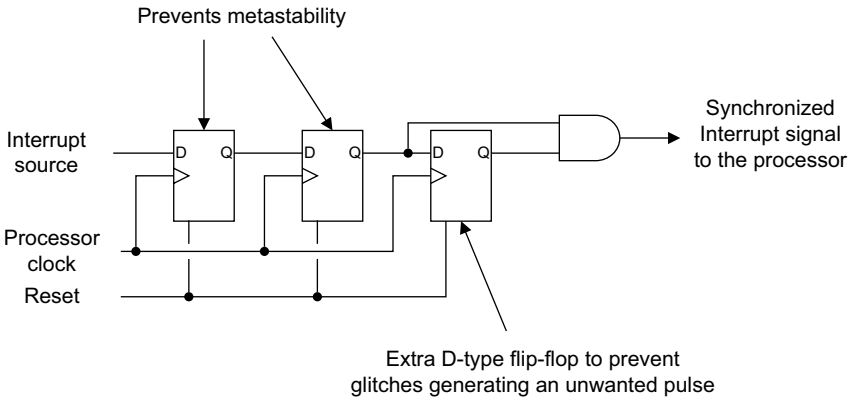


Figure 2-6 IRQ synchronizer

Note

The IRQ synchronizer only works with a level-triggered interrupt source, so the peripheral must hold the interrupt signal HIGH until the processor clears the **ISR** interrupt signal.

The APB subsystem contains several example IRQ synchronizers to demonstrate their use. The synchronizers are optional. They are only enabled if you set the Verilog parameter `INCLUDE_IRQ_SYNCHRONIZER` to a non-zero value. This Verilog parameter is defined in the `apb_subsystem.v` file. It is not overridden in the `cmsdk_mcu_system.v` file.

The example system design uses the same clock source for the processor clock **HCLK** and the peripheral clocks **PCLK** and **PCLKG**. Therefore there is no asynchronous clock domain boundary, so this parameter is set LOW.

2.9.3 Event

The Cortex-M0 processor has an **RXEV** input signal. If software uses the `WFE` instruction to put the processor to sleep, an event received at **RXEV** wakes up the processor.

2.10 Clock and reset

The example microcontroller uses a single reset and a single clock source. The clock and reset controller performs:

- The reset synchronization of the reset input.
- The generation of the reset outputs.
- The clock generation for the peripheral subsystem.

See the *Cortex-M0 DesignStart Eval FPGA User Guide* for details of the clocks and resets when using the FPGA system.

Figure 2-7 shows the clock and reset operation of the example microcontroller.

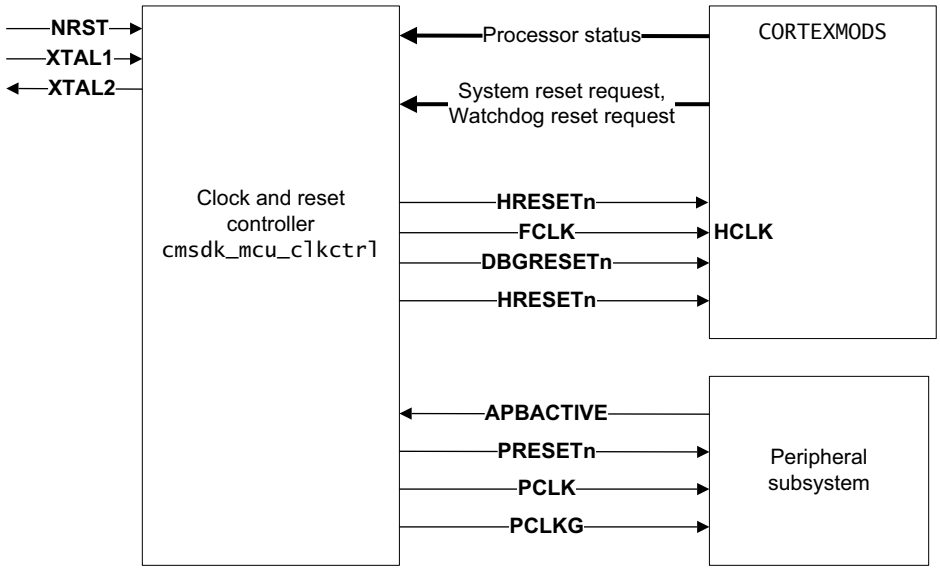


Figure 2-7 Example microcontroller clock and reset operation

The example only demonstrates a simple application scenario. Arm recommends that, for actual silicon projects, you modify the clock controller design for device-specific testability and clocking requirements.

The AHB to APB bridge in the APB subsystem permits the APB peripheral bus to run at a clock rate that is derived from the AHB clock by **PCLKEN**. By default the example system ties HIGH **PCLKEN** that connects to the AHB to APB bridge. Therefore **PCLK** is the same as **HCLK**. If you require a slower APB clock, you must:

- Modify the Verilog file `cmsdk_mcu_clkctrl.v` to generate **PCLKEN** at a reduced rate.
- Use **PCLKEN** and clock gating logic to generate **PCLK**.

The Verilog file `cmsdk_mcu_clkctrl.v` is the clock and reset controller, and provides an example of how to create a lower **PCLK** frequency. The `ARM_CMSDK_SLOWSPEED_PCLK` preprocessing directive enables this feature. The Verilog file also provides a **PCLKG** clock signal used by the APB interface logic in the peripherals. If there is no APB transfer activity, you can turn off the **PCLKG** signal to reduce power. The AHB to APB bridge generates the **APBACTIVE** signal that controls the generation of **PCLKG**.

2.11 SysTick support

The example system includes a simple divider to provide a reference clock for the SysTick timer. The divider has a divide ratio of 1000. The system runs at 50MHz in simulation, so the SysTick reference clock runs at 50KHz.

Table 2-17 describes the bit field values of the *SysTick Calibration Value Register* (SYST_CALIB).

Table 2-17 STCALIB register bit field values

Signal	SysTick->CALIB register field	Value	
STCALIB[25]	NOREF (bit 31)	**0**	Reference clock is available
STCALIB[24]	SKEW (bit 30)	**1**	Calibration value is not accurate
STCALIB[23:0]	TENMS (bit 23 to 0)	**0**	Calibration value is not available

Note

See the *Armv6-M Architecture Reference Manual* for more information about the SYST_CALIB Register.

Chapter 3
Example System Testbenches

This chapter describes the testbench components. It contains the following sections:

3.1 About the testbench design

Both of the example systems include a testbench to enable you to simulate the example microcontroller designs with a supported Verilog simulator.

The testbenches include:

- A loop back connection for UART testing.
- A clock and reset generator.
- Text message capture by the UART.
- Arduino shield model components for the FPGA system.
- External memory models for the FPGA system.

Figure 2-1 on page 2-2 shows the testbench in use with an example system.

For the CMSDK mcu system simulation, XTAL1 runs at 50MHz. **NRST** is asserted LOW for 5ns at the beginning of the simulation.

The serial output of UART0 is connected to a UART capture module that can generate text messages during simulation. See the `cmsdk_uart_capture.v` file. In the FPGA testbench this can be used to control loopback functions in the testbench (although the provided tests do not exersise this).

3.2 UART text output capturing and escape code

When a program wants to display a message in the simulation environment, it can execute the `printf` or `puts` functions. It can also directly call the UART routines to output the message to UART0. When it executes the `printf` or `puts` functions, the UART output routine executes through retargeting code and outputs the characters to the serial output of UART0. The UART capture module captures the input data and outputs the received characters when it receives the *Carriage Return* (CR) character.

To reduce simulation time, the high-speed test mode of the example system UART outputs each bit in one clock cycle. Therefore, the UART capture module captures the input data at one bit per cycle. If the UART outputs serial data at a different speed, you must change the clock that connects to the UART capture module.

You can also use the UART capture module to terminate a simulation. When it receives a character value of `0x4`, unless it receives this character immediately following the ESC (`0x1B`) character, it stops the simulation using the `$stop` Verilog system task. Before the end of the simulation, the UART capture module outputs a pulse on the **SIMULATIONEND** output to enable you to use this signal to trigger other tasks or hardware logic before the end of a simulation.

Chapter 4
Using the Simulation Environment

This chapter describes how to set up and run simulation tests. It contains the following sections:

4.1 About the simulation environment

The simulation environment in this example system enables you to start a system-level simulation quickly. The simulation environment includes software files and simulation setup makefiles.

The simulation environment supports the following Verilog simulators:

- Mentor ModelSim.
- Cadence NC Verilog.
- Synopsys VCS.

The makefile for setting up the simulation is created for the Linux platform.

You can compile the example software using any of the following:

- Arm *Development Studio 5* (DS-5).
- Keil *Microcontroller Development Kit* (MDK).
- GNU Tools for Arm Embedded Processors (Arm GCC).

The Keil MDK is available only for the Windows platform. Therefore, to use Keil MDK you must carry out the software compilation and the simulation in two separate stages. A limited term license of Keil MDK is included with the Cortex-M0 DesignStart Eval product. You will need to install this license to compile some of the tests that are provided.

4.2 Files and directory structure

Figure 4-1 shows the layout of the directories in the example system.

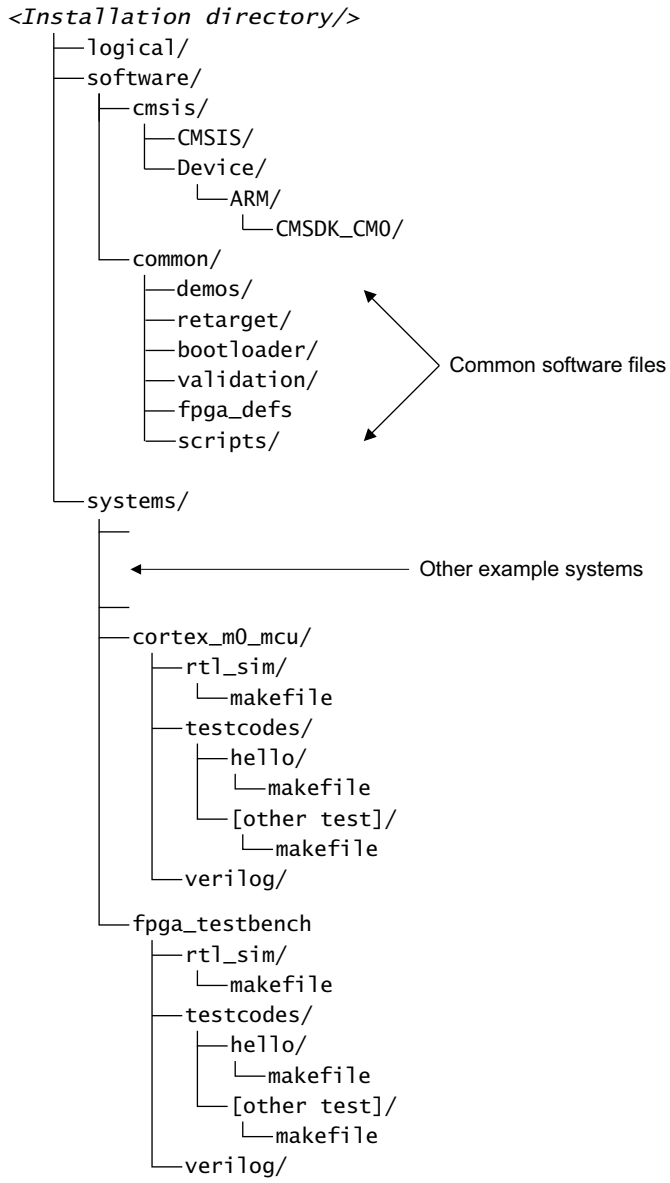


Figure 4-1 Directories for simulation

Table 4-1 on page 4-4 describes the contents of several of the directories in Figure 4-1.

Table 4-1 Installation directory information

Directory name	Directory contents
software/cmsis/CMSIS	Example processor support files.
software/cmsis/Device/ARM/	Example device-specific processor support files and example system header files, for example, CMSDK_CM0 for Cortex-M0.
systems/cortex_m0_mcu/verilog	Verilog and Verilog command files.
systems/cortex_m0_mcu/rtl_sim/	Files for simulation that includes a makefile to compile the Verilog and run the simulation. It also invokes a makefile in the testcodes directory to compile the software.
systems/cortex_m0_mcu/testcodes/	Testcodes for software testing.
systems/cortex_m0_mcu/testcodes/*<testname>*/makefile	Makefile for example testcodes, for DS-5 and Arm GCC.
systems/cortex_m0_mcu/testcodes/keil_multiple/cm0ds_all.uvmpw	Keil MDK Multi-project workspace.
systems/fpga_testbench/verilog	Verilog and Verilog command files.
systems/fpga_testbench/rtl_sim/	Files for simulation that includes a makefile to compile the Verilog and run the simulation. It also invokes a makefile in the testcodes directory to compile the software.
systems/fpga_testbench/testcodes/	Testcodes for software testing.
systems/fpga_testbench/testcodes/*<testname>*/makefile	Makefile for example testcodes, for DS-5 and Arm GCC.
systems/fpga_testbench/testcodes/keil_multiple/cm0ds_all.uvmpw	Keil MDK Multi-project workspace.
software/common/demos	C program codes for demonstration.
software/common/validation	C program codes for functional tests.
software/common/bootloader	Example boot loader.
software/common/dhry	Dhrystone demonstration.
software/common/retarget	Support files to handle printing.
software/common/scripts	Linker scripts.
software/common/fpga_defs	Header files and drivers for FPGA peripherals.

4.3 Setting up the simulation environment

This section describes how to set up the simulation environment. It contains the following:

- *Modifying the rtl_sim/makefile*.
- *Modifying configuration files*.
- *Setting up tools*.

There are two similar simulation environments, for the simple RTL testbench in `systems/cortex_m0_mcu` and the FPGA system in `systems/fpga_testbench`. Each environment must be configured seperately, although they use a common `software` directory.

4.3.1 Modifying the rtl_sim/makefile

The makefile in the `rtl_sim` directory controls the following simulation operations:

- Compiling the RTL.
- Running the simulation in batch mode.
- Running the simulation in interactive mode.

You must specify several variables inside this makefile. Table 4-2 describes the variables.

Table 4-2 Makefile variables

Variable	Descriptions
`TESTNAME`	Name of software test to be executed, for example, `hello` or `dhry`. This name must match the software directory name inside the `systems/cortex_m0_mcu/testcodes/`or `systems/fpga_testbench/testcodes/` directory.
`TEST_LIST`	List of tests available.

Note

- You do not have to edit all of these variables every time you run a different test. You can override the makefile variables with command line options. For example, you can keep the `TESTNAME` variable unchanged, and override it only when you run a simulation.
- See *Run the simulation* on page 4-8 for example test programs.

4.3.2 Modifying configuration files

The `systems/cortex_m0_mcu/verilog/cmsdk_mcu_defs.v` file specifies most of the configurations of the example system. Arm recommends that you use the example MCU system to investigate the impact of these parameterized configurations, the FPGA system is not validated to be complete if the configuration is changed.

4.3.3 Setting up tools

The simulation requires one of the supported Verilog simulators and tools, for compiling and assembling the software code.

4.4 Running a simulation in the simulation environment

This section describes how to run a simulation in the design toolkit. It contains the following sections:

- *Compile the RTL*.
- *Compile the test code*.
- *Run the simulation* on page 4-8.

4.4.1 Compile the RTL

After you have configured the environment, you must compile the Verilog RTL in the `rtl_sim` directory. To do this, use the following command:

```
<installation directory>/systems/cortex_m0_mcu/rtl_sim> make compile
```

This starts the compilation process. Depending on the system which you are using, one of the following Verilog command files are used to specify the relevant source directories:

- `systems/cortex_m0_mcu/verilog/tbench_M0_DS.vc`
- `systems/fpga_testbench/rtl_sim/tbench.vc`

The compile stage ignores the `TESTNAME` setting.

You can use the command line to override variables in the makefile. For example, the following command line specifies that Modelsim is used for compilation:

```
<installation directory>/systems/cortex_m0_mcu/rtl_sim> make compile SIMULATOR=mti
```

4.4.2 Compile the test code

Software compilation for the FPGA testbench and the standard CMSDK mcu system differ, primarily in their memory map. The FPGA testbench will compile with read-only and read-write regions both targetting the code memory region starting at `0x00000000`. For this reason, the `testcode` directory structures are duplicated within each system directory. Note that the cortex_m0_mcu system has a more exhaustive set of tests for the system. With the FPGA system, testing has been performed using the Cortex-M Prototyping System testcode (the source for this is provided with the MPS2+ platform).

Before you compile the software code, you might want to change some of the settings for the software compilation. Each software test has a corresponding subdirectory in the `systems/<system>/testcodes` directory. Inside each of these directories is a makefile for software compilation. The makefiles support Arm DS-5, Keil MDK (as a placeholder for compilation) and Arm GCC. Table 4-3 lists the settings contained in the makefiles.

Table 4-3 Makefile settings

Variable	Descriptions
`TOOL_CHAIN`	This can be set to one of the following: **ds5** Arm DS-5. **gcc** Arm GCC. **keil** Keil MDK. If you select `keil`, the make process pauses so you can manually continue the compilation from Keil MDK in the Windows environment.
`TESTNAME`	Name of the software test. This must match the directory name.
`COMPILE_MICROLIB`	Use only for the DS-5 option. **0** Normal C runtime library. This is the default value. **1** MicroLIB, a C runtime library optimized for microcontroller applications.
`USER_DEFINE`	A user-defined C preprocessing macros. Set to `-DCORTEX_M0` for most test codes. This enables a piece of test code to include the correct header for the processor when multiplex example systems share the test code. You can add more preprocessing macros for your applications.
`SOFTWARE_DIR`	Shared software directory
`CMSIS_DIR`	Base location of all CMSIS source code.
`DEVICE_DIR`	Device specific support files, for example, header files, and device driver files.
`STARTUP_DIR`	Startup code location.
`ARM_CC_OPTIONS`	Arm C Compiler options. Use only for the DS-5 option.
`ARM_ASM_OPTIONS`	Arm Assembler options. Use only for the DS-5 option.
`ARM_LINK_OPTIONS`	Arm Linker options. Use only for the DS-5 option.
`GNU_CC_FLAGS`	gcc compile option. Use only for the Arm GCC option.
`LINKER_SCRIPT`	Linker script location. Use only for the Arm GCC option.

Note

A Keil-specific project file specifies the options for Keil MDK.

Use makefiles to compile your software. You can use one of the following makefiles:

- *The makefile in testcodes/<testname>*.
- *The makefile in rtl_sim, software compilation only* on page 4-8.

The makefile in testcodes/<testname>

Execute the following:

`make all` This starts the software compilation process for DS-5 or Arm GCC.

You can override the variable in the makefile, for example, by executing the following:

`make all TOOL_CHAIN=ds5 COMPILE_MICROLIB=1`

This causes the program to compile using DS-5 with the MicroLIB option enabled.

`make clean` This cleans all intermediate files created during the compilation process invoked by `make all`. If changes are made in code other than the testcode itself, for example, in the CMSIS header files, running `make clean` ensures that these changes are detected by a subsequent `make all`.

The makefile in rtl_sim, software compilation only

For example, in `systems/cortex_m0_mcu/rtl_sim/`, you can execute:

`make code` The makefile in the `rtl_sim` directory changes the current directory to the one specified by the `TESTNAME` variable. By default there is no `TESTNAME` specified in the makefile. If `make code` is executed without specifying a `TESTNAME` on the make command line or by editing the makefile, a message is printed requesting a `TESTNAME` to be specified.

You can use the command line to specify the software test that you want to run by executing the following:

`make code TESTNAME=hello`

This causes the `hello` test and the bootloader code to compile. The process then copies the compiled code images to the `rtl_sim` directory.

Note

Use the `make code` option to debug compilation errors because this option does not invoke simulation.

4.4.3 Run the simulation

After the RTL compilation, you can start the simulation in the `systems/cortex_m0_mcu/rtl_sim/` directory using one of the following commands:

`make sim` For interactive simulation.

`make run` For batch mode simulation.

The makefile in the `rtl_sim` directory automatically invokes the makefiles in the testcodes directories. Figure 4-2 on page 4-9 shows the interaction of the makefiles.

When you run an interactive simulation, you can step (and set breakpoints) in the Verilog code. You can also log the signals in the design and investigate the hardware operation.

Note

The `make run` and the `make sim` step automatically runs the `make code` operation. Therefore, if you have previously compiled a test using `make code` with specific options, you must repeat the same options when you invoke `make run` or `make sim`.

For example:

- `make code TESTNAME=sleep_demo TOOL_CHAIN=ds5 COMPILE_MICROLIB=1`
- `make sim TESTNAME=sleep_demo TOOL_CHAIN=ds5 COMPILE_MICROLIB=1 SIMULATOR=vcs`

If you do not do this, the software test might be recompiled without the previous configuration settings. The command `make code` enables you to test that a program file compiles correctly. It does not start the simulation.

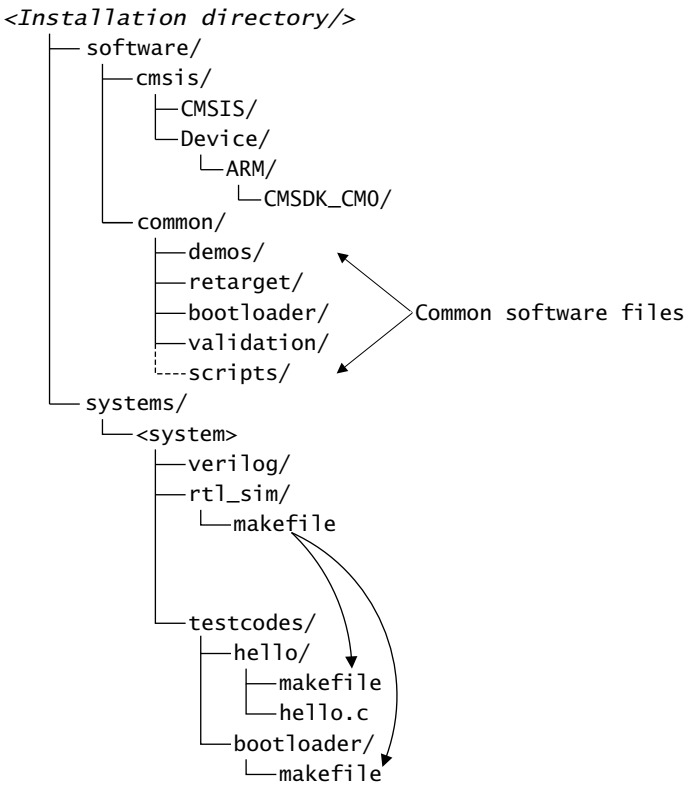


Figure 4-2 Interaction of makefiles

The software directory contains shared header files and shared test programs.

To compile the software and run a simulation, execute `make run TESTNAME=hello` in the `rtl_sim` directory:

- The makefile in the `rtl_sim` directory uses the makefile in the `bootloader` directory to compile the boot loader code and copy the resulting image back to the `rtl_sim` directory.
- The makefile in the `rtl_sim` directory uses the makefile in the `hello` directory to compile the hello world code and copy the resulting image back to the `rtl_sim` directory.
- The makefile in the `rtl_sim` directory starts the simulator.

If you set the software toolchain in the makefile to `keil`, this causes the make process to pause and prompt you to compile your project in Keil MDK. You can resume the process by pressing any key.

You can use command line options to override the makefile variables. For example:

- `make sim TESTNAME=sleep_demo SIMULATOR=vcs TOOL_CHAIN=ds5`

The last action of the simulation writes the value `0x4` to UART0. When the `cmsdk_uart_capture` device captures this value, it triggers the simulation to stop.

For example, the `hello` test results in the following output for the Cortex-M0 processor:

```
# 30490 ns UART: Hello world
# 52410 ns UART: ** TEST PASSED **
# 54270 ns UART: Test Ended
# ** Note: $stop    : ../verilog/cmsdk_uart_capture.v(208)
#    Time: 54270 ns  Iteration: 1  Instance: /tb_cmsdk_mcu/u_cmsdk_uart_capture
# Break at ../verilog/cmsdk_uart_capture.v line 208
# Stopped at ../verilog/cmsdk_uart_capture.v line 208
# quit -f
```

The Verilog file `cmsdk_uart_capture.v` contains the text `Test Ended` that it displays before the simulation stops. See *Retargeting* on page 5-6 for details on retargetting `STDOUT`.

To compile the testbench and run all the tests that the `TEST_LIST` variable specifies, execute the following on the command line:

```
make all
```

You can use the command line to override several test parameters. For example, to specify the VCS simulator execute the following:

```
make all SIMULATOR=vcs
```

4.4.4 Debugging tests

When you run tests in the simulation, you can only observe the verilog signals in the system, and the processor architectural registers. There is no specific support for software debug, for example connecting a debugger to the debug hardware which is being simulated. To debug your software you can:

- Monitor the instruction fetches on the AHB bus.
- View the `.lst` files which are generated in the `testcode` directory.
- Use the processor simulator in Keil MDK.

A 'tarmac' trace of executed instructions is supported by the full Cortex-M0 processor (which you can license as part of Cortex-M0 DesignStart Pro). Tarmac trace is also supported by the cycle model which is included with the Cortex-M3 DesignStart Eval (although the two DesignStart Eval systems are not directly interchangeable).

Chapter 5
Software Examples

This chapter describes the example software tests and the device drivers. It contains the following sections:

5.1 Available simulation tests

Table 5-1 shows the example software tests that this design kit contains.

Table 5-1 Example software test list

TESTNAME	Descriptions
hello	Simple test to display the Hello world message. It uses the retargeting action that redirects printf to the UART output.
interrupt_demo	Demonstration of interrupt features. Only present in cortex_m0_mcu.
sleep_demo	Demonstration of sleep features.
dhry	Simple Dhrystone test.
self_reset_demo	Demonstration of the self reset feature that uses the signal **SYSRESETREQ**.
dualtimer_demo	Demonstration of the APB Dual Timer.
watchdog_demo	Demonstration of the APB Watchdog.
rtx_demo	Demonstration of the Keil RTX OS.
gpio_tests	Tests the low latency AHB GPIO. Supports I/O GPIO.
timer_tests	Tests the simple APB timer.
uart_tests	Tests the simple APB UART. Only present in cortex_m0_mcu.
default_slaves_tests	Tests the default slave activation. It accesses invalid memory locations.
gpio_driver_tests	Simple test for the GPIO device driver functions. Only present in cortex_m0_mcu.
timer_driver_tests	Simple test for the simple timer device driver functions.
uart_driver_tests	Simple test for the UART device driver functions. Only present in cortex_m0_mcu.
apb_mux_tests	Simple test for the APB slave multiplexer.
memory_tests	Simple test for the system memory map. Only present in cortex_m0_mcu.
designstest_m0	Test of the peripherals specific to the FPGA system. Only present in fpga_testbench.

Some of the tests are timing dependent and are written for a system with zero wait states. The test might fail if you change the wait states of the system.

The RTX OS is a feature in Keil MDK. The example software package includes a precompiled hex file of the RTX demonstration test, therefore you can simulate this test without a Keil MDK setup. For this test, the example software package includes the project files that you can modify and recompile if you require.

The config_id.h header file in the testcodes/generic directory contains the defines for each of the available functions in the Cortex-M0 processor. The values are set to match the fixed configuration of the CortexM0 processor from DesignStart.

5.2 Creating a new test

You can add new tests to the testcodes directory. Use the hello test as a guide to the format you can use. For example, you can use the following process to create a new test:

1. Create a new directory in the `testcodes/` directory. For example:
   a. `cd <installation_directory>/systems/<system>/testcodes`
   b. `mkdir mytest`
2. Copy the files that are located in the `hello/` directory to your new test directory, and then rename the test file. For example:
   a. `cd mytest`
   b. `cp ../hello/* .`
   c. `mv hello.c mytest.c`
3. Edit the makefile to rename `hello.c` to `mytest.c`
4. Ensure that the output hex file has the same name as the directory name, for example `mytest.hex`. This enables the `makefile` in `rtl_sim/` directory to copy the hex file to the `rtl_sim/` directory before the simulation starts.
5. If required, you can add the name of your new test to the `TEST_LIST` variable in the `makefile` located in the `rtl_sim/` directory.

5.3 Example header files and device driver files

The example software uses header files that are based on the *Cortex Microcontroller Software Interface Standard* (CMSIS). The example software includes the following types of files:

- Generic Cortex-M0 processor header files, located in directory `software/cmsis/CMSIS/Include/`.
- Device-specific header files, located in directory `software/cmsis/Device/ARM/CMSDK_CM0/`.
- Device-specific startup codes, located in directory `cmsis/Device/ARM/CMSDK_CM0/Source/`.
- Device-specific example device drivers, located in directory `cmsis/Device/ARM/CMSDK_CM0/`.

Note

You must update to the latest version of the CMSIS-Core files when preparing your own CMSIS software packages. See Arm CMSIS-Core http://www.arm.com/cmsis.

Table 5-2 shows the generic Cortex-M0 processor support files.

Table 5-2 Generic Cortex-M0 processor support files

Filename	Descriptions
core_cm0.h	CMSIS 3.0 compatible header file for processor peripheral registers definitions.
core_cmInstr.h	CMSIS 3.0 compatible header file for accessing special instructions.
core_cmFunc.h	CMSIS 3.0 compatible header file for accessing special registers.

Table 5-3 shows the device-specific header files.

Table 5-3 Device-specific header files

Filename	Descriptions
CMSDK_CM0.h	CMSIS compatible device header file including register definitions
system_CMSDK_CM0.h	CMSIS compatible header file for system functions
system_CMSDK_CM0.c	CMSIS compatible program file for system functions

Table 5-4 shows the device-specific startup codes.

Table 5-4 Device-specific startup codes

Filename	Descriptions
cmsis/Device/ARM/CMSDK_CM0/Source/ARM/startup_CMSDK_CM0.s	CMSIS compatible startup code for Arm DS-5 or Keil MDK
cmsis/Device/ARM/CMSDK_CM0/Source/GCC/startup_CMSDK_CM0.s	CMSIS compatible startup code for ARM GCC

Table 5-5 shows the device-specific example device drivers.

Table 5-5 Device-specific example device drivers

Filename	Descriptions
CMSDK_driver.h	Header file for including driver code
CMSDK_driver.c	Driver code implementation

To use these header files, you only have to include the device-specific header file CMSDK_CM0.h. This file imports all the required header files. Because some of the shared program files in the software/common directory also support different types of processor, these programs include the following header code:

```
#ifdef CORTEX_M0
#include "CMSDK_CM0.h"
#endif
```

The makefile in directory systems/cortex_m0_mcu/testcodes/<testname> contains the USER_DEFINE variable that defines the C preprocessing directive CORTEX_M0. This ensures that the simulation uses the correct version of the header file.

5.4 Retargeting

Several test programs use the `printf` and `puts` functions to display text messages during the simulation. The retargeting code performs this function. It redirects text output to UART0. The `tb_uart_capture` device in the testbench captures the text and outputs it to the simulation console during the simulation.

You must ensure that your code calls `UartStdOutInit()` before any `printf` or similar functions are called.

The retargeting is written to use the high-speed simulation mode of the CMSDK UART. If you want to use the same retargetting in an actual FPGA (with realistic baud rates) you will need to add a polling check whilst the characters are transmitted.

For the ARM DS-5 and Keil MDK environments, the retarget function for text output is `fputc`. The retarget function for ARM GCC, and most gcc based C compilers, is the `_write_r` function. These functions are located in file `software/common/retarget/retarget.c`.

Table 5-6 shows the files required for retargeting support.

Table 5-6 Retargeting support files

Files	Descriptions
`software/common/retarget/retarget.c`	Retargeting implementation for ARM DS-5, Keil MDK, and ARM GCC
`software/common/retarget/uart_stdout.h`	Header for UART functions used by `retarget.c`
`software/common/retarget/uart_stdout.c`	Implementation of UART functions

The UART support files are `uart_stdout.c` and `uart_stdout.h`. Table 5-7 shows the UART functions.

Table 5-7 Support file functions

Function	Descriptions
`void UartStdOutInit(void)`	Initialize UART0 and GPIO0 (for pin multiplexing) for text message output.
`char UartPutc(unsigned char my_ch)`	Output a single character to UART 0.
`char UartGetc(void)`	Read a character from UART.
`char UartEndSimulation(void)`	Terminate the simulation by sending value `0x4` to UART 0. When `tb_uart_capture` receives this data it stops the simulation.

Appendix A
Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1 Issue A

Change	Location	Affects
First release for r1p0	-	-

Table A-2 Differences between Issue A and Issue B

Change	Location	Affects
First release for r2p0	Across the whole document, updates to reflect change to CORTEXM0INTEGRATION obfuscation, including addition of debug. FPGA Example system memory map added in *Memory map*Memory map on page 2-11	r2p0
FPGA testbench described in this document.	*Example FPGA system level design and design heirarchy*Example FPGA system level design and design heirarchy on page 2-5 Chapter 3 *Example System Testbenches* Chapter 4 *Using the Simulation Environment* Chapter 5 *Software Examples*	r2p0
GCC and Keil toolchain support added for FPGA testbench.	Chapter 4 *Using the Simulation Environment*	r2p0