ARM® Cortex®-M0 DesignStart™ RTL Testbench

Revision: r1p0

User Guide

ARM®

ARM DUI 0926A (ID062017)

ARM Cortex-M0 DesignStart RTL Testbench
User Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
28 September 2015	A	Non Confidential	First draft for r1p0

Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

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 RTL Testbench User Guide

Preface

This preface introduces the *Cortex-M0 DesignStart Design Kit Example System Guide*. It contains the following sections:

About this book

This book is for the Cortex-M0 DesignStart Design Kit.

Implementation obligations

This book is designed to help you implement an ARM product. The extent to which the deliverables can be modified or disclosed is governed by the contract between ARM and the Licensee. There might be validation requirements which, if applicable, are detailed in the contract between ARM and the Licensee and which, if present, must be complied with prior to the distribution of any devices incorporating the technology described in this document. Reproduction of this document is only permitted in accordance with the licenses granted to the Licensee.

ARM assumes no liability for your overall system design and performance. Verification procedures defined by ARM are only intended to verify the correct implementation of the technology licensed by ARM, and are not intended to test the functionality or performance of the overall system. You or the Licensee are responsible for performing system level tests.

You are responsible for applications that are used in conjunction with the ARM technology described in this document, and to minimize risks, adequate design and operating safeguards must be provided for by you. Publishing information by ARM in this book of information regarding third party products or services is not an express or implied approval or endorsement of the use thereof.

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*

Read this for an introduction to the Cortex-M0 DesignStart Design Kit and its features.

Chapter 2 *Functional Description*

Read this for a description of the design and layout of the design kit.

Chapter 3 *Example System Testbench*

Read this for a description of the testbench components.

Chapter 4 *Using the Simulation Environment*

Read this for a description of how to set up and run simulation tests.

Chapter 5 *Software Examples*

Read this for a description of the example software tests and the device drivers.

Chapter 6 *Synthesis*

Read this for a description of how to run synthesis for the example system.

Appendix A *Revisions*

Read this for a description of 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 *ARM Glossary* `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Conventions

This book uses the conventions that are described in:

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

Typographical conventions

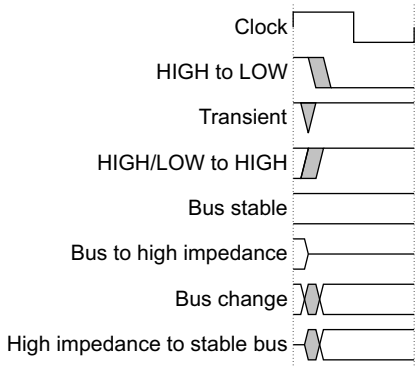
The following table describes the typographical conventions:

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* on page ix 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* (ARM DDI 0479).
- *ARM® Cortex®-M0 Devices Generic User Guide* (ARM DUI 0497).
- *ARM® Cortex®-M0 Technical Reference Manual* (ARM DDI 0432).
- *ARM® ARMv6-M Architecture Reference Manual* (ARM DDI 0419).
- *ARM® AMBA®3 AHB-Lite Protocol (v1.0) Specification* (ARM IHI 0033).

The following confidential books are only available to licensees:

- *ARM® Cortex®-M0 Integration and Implementation Manual* (ARM DII 0238).
- *ARM® Cortex®-M0 User Guide Reference Material* (ARM 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, ARM DUI 0926A.
- 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 the Cortex-M0 DesignStart Design Kit. It contains the following sections:

1.1 About Cortex-M0 DesignStart Kit

The Cortex-M0 DesignStart Kit is intended for system Verilog design and simulation of a prototype SoC based on the Cortex-M0 processor.

The DesignStart Design Kit has:

- An ARM Cortex-M0 processor from DesignStart.
- An example system-level design for the ARM Cortex-M0 processor.
- 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 does not have debug capability, as this is not used in simulation, and is not intended for production silicon. See *Limitations of the design kit* on page 1-5 for the Cortex-M0 processor from DesignStart configuration information.

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 and does include the debug unit. To purchase the prototyping system, go to the ARM website http://www.arm.com/products/tools/development-boards/versatile-express/cortex-m-prototyping-system.php.

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:

- 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 User Guide Reference Material*. 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 Design Kit directory structure

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

Table 1-1 Main directory descriptions

Directory name	Directory contents
`logical`	Verilog components including AHB-Lite and APB infrastructure components, peripherals, the APB subsystem.
`systems`	Design files, testbench files, and simulation setup files for the example system.
`implementation_tsmc_ce018fg`	Synthesis setup files for the example system. The files support the Synopsys Design Compiler.
`software`	Software files. These include: • CMSIS compatible C header files. • Example program files for the example systems. • An example device driver.
`documentation`	Documentation files.
`cores`	This is the location for processor core files.

Figure 1-1 on page 1-4 shows the location of the file directories in the design kit.

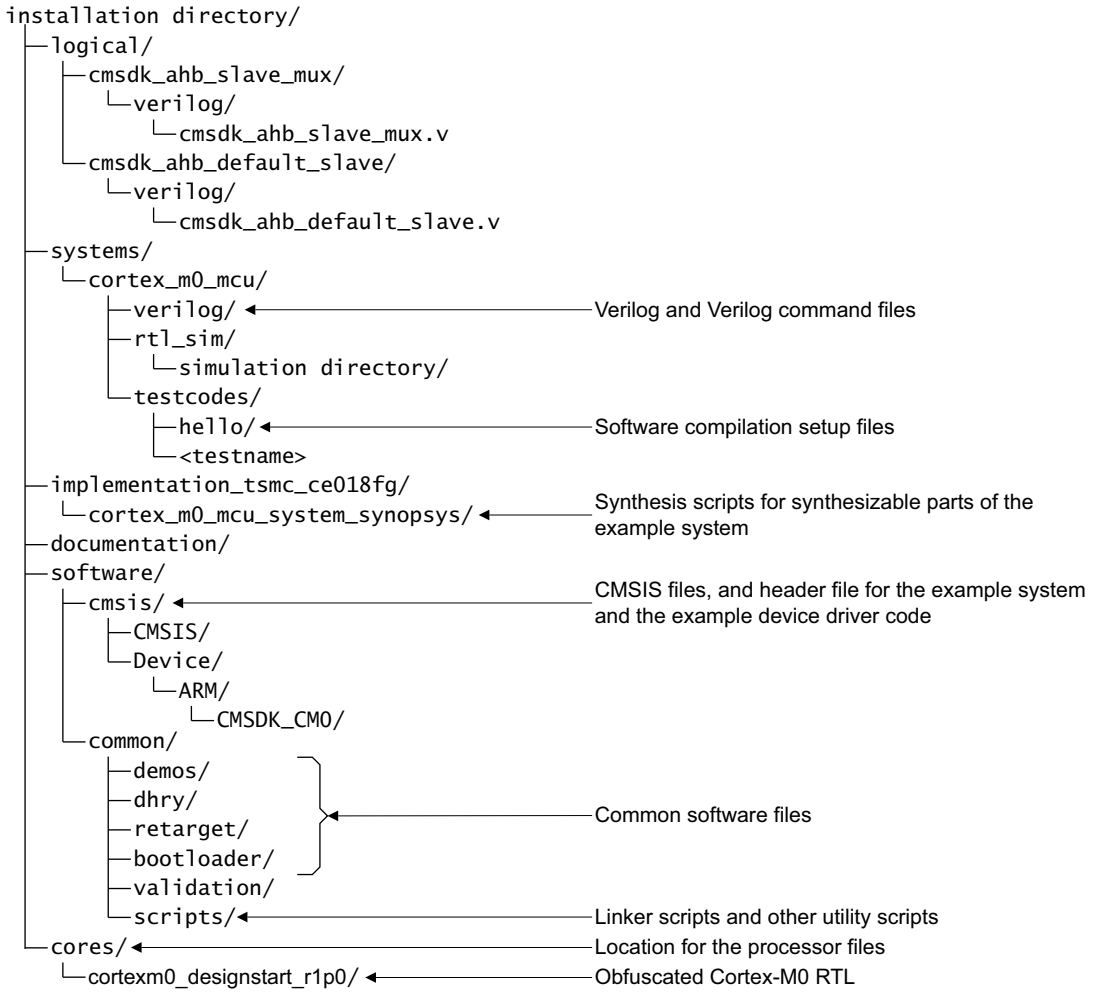


Figure 1-1 Directory structure

1.3 Limitations of the design kit

This section describes the limitations of the design kit. 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.

1.3.1 Deliverables

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

1.3.2 Processor support

The design kit supports the Cortex-M0 processor from DesignStart.

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	None
Hardware debug support	Optional single step with up to four breakpoints, up to two watchpoints and PC sampling	None
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 processor example system and its peripherals in the design kit are *little-endian*.

1.3.4 Platform

This release of the Cortex-M0 DesignStart Design Kit supports Linux and Unix for the simulation process and the synthesis process. If you use Keil MDK-ARM for software development, you can install the design kit in a location that is accessible from Linux, Unix, and Windows. Do this using one of the following procedures:

- Install the design kit 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 design kit in the shared folder.

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

To run the design kit on other operating systems, modify the `makefiles` to meet your specific requirements.

Chapter 2
Functional Description

This chapter describes the design and layout of the design kit. It contains the following sections:

2.1 System-level design and design hierarchy

The example system is a simple microcontroller design. 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.

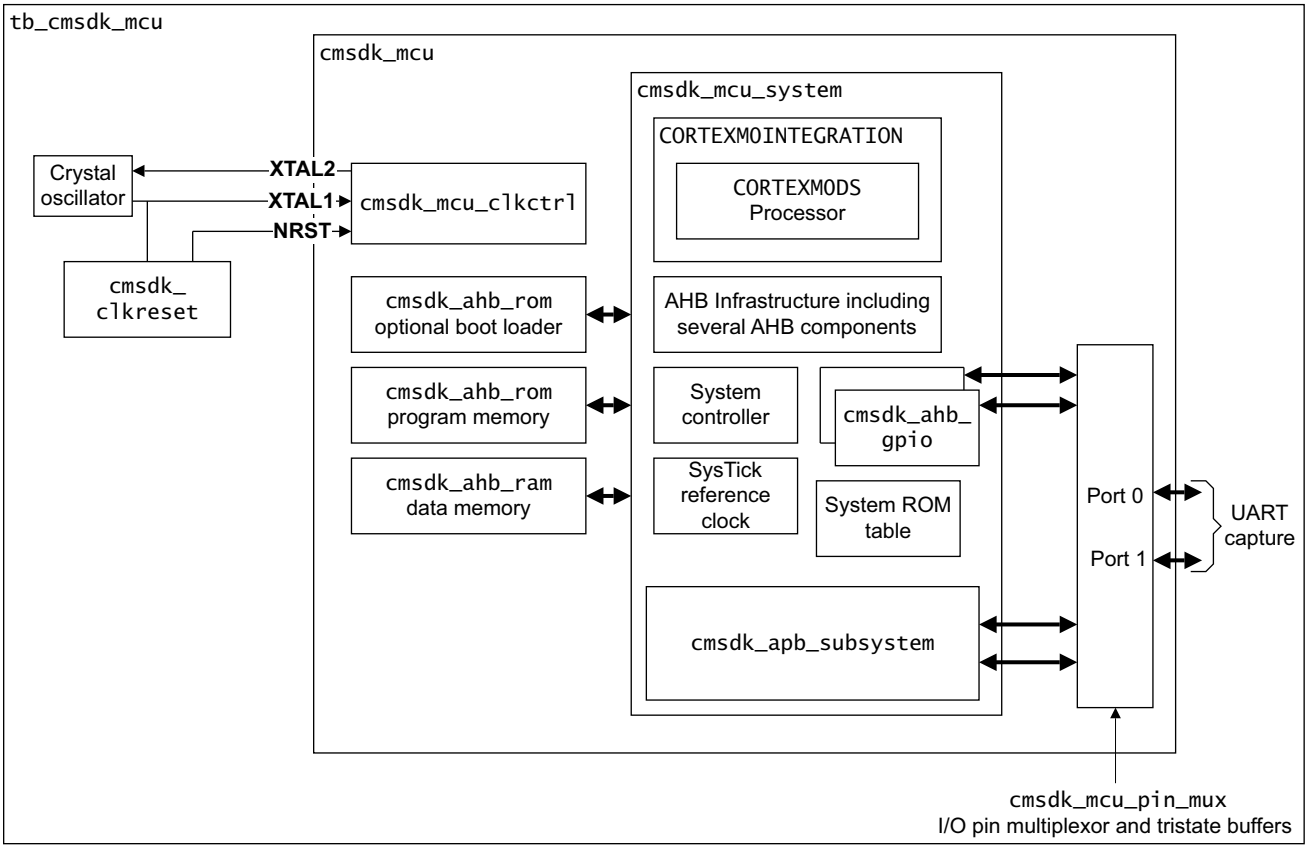


Figure 2-1 Example microcontroller system top level view

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`	Instantiates the `CORTEXM0DS` layer.

Table 2-1 Microcontroller items (continued)

Item	Description
CORTEXM0DS	The Cortex-M0 Integration layer. This is obfuscated 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 in the design kit 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 kit the DAP and WIC are not included. Empty files are included to indicate how the DAP and WIC are integrated in a full release.

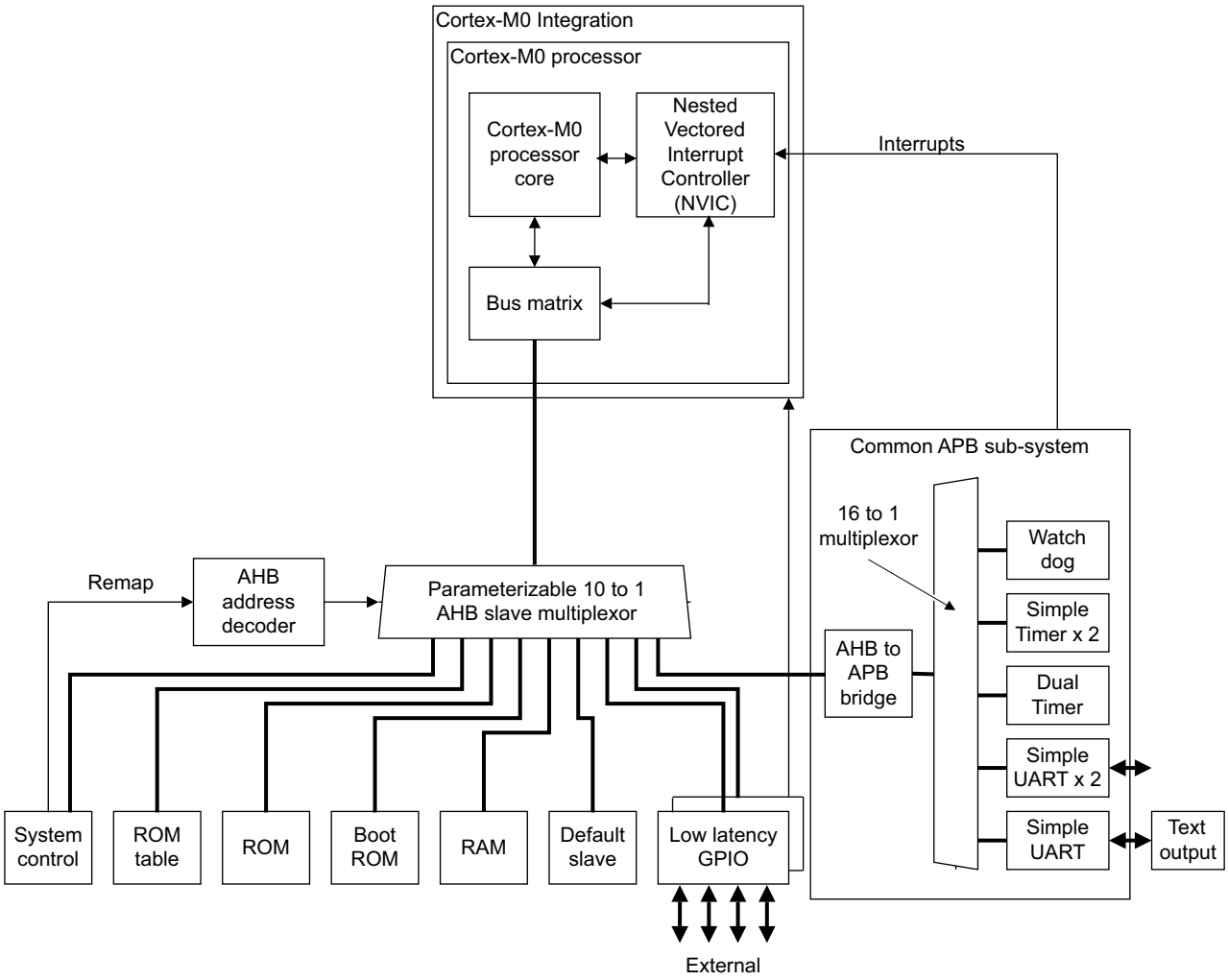


Figure 2-2 Cortex-M0 example system

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

Table 2-3 Design kit 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 Design files

This section describes the following design files that are included in the design kit:

- *Verilog files for the cmsdk_mcu example system*.
- *Verilog files for the cortex_m0_mcu testbench*.
- *Other files*.

2.2.1 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

2.2.2 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

2.2.3 Other files

See Chapter 4 *Using the Simulation Environment* for information on the simulation setup and the test codes to run simulations.

2.3 Processor file location

The location of the Verilog RTL files for the processors is a subdirectory called `cores`. For Cortex-M0 DesignStart RTL path is:

- `cores/cortexm0_designstart_r1p0/logical/.`

2.4 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 2-6 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*.

2.5 Memory map

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

- *AHB memory map*.
- *APB subsystem memory map* on page 2-9.

2.5.1 AHB memory map

The AHB memory map has a 4GB linear address range, but the system only uses 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 file `cmsdk_mcu_system.v` contains the address decoding logic. If you modify the memory map, you must also modify the address decoding logic in this file.

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 2-7 describes the AHB memory map with and without the remap function.

Table 2-7 AHB memory map

Address	Without remap
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.
0x40020000-0xEFFFFFFF	Unused, except for the private peripheral bus addresses in the Cortex-M0.
0x4001F000-0x4001FFFF (4KB)	System controller registers.
0x40012000-0x4001EFFF	Unused.
0x40011000-0x40011FFF (4KB)	Unused.
0x40010000-0x40010FFF (4KB)	Unused
0x40000000-0x4000FFFF (64KB)	APB subsystem peripherals.
0x20010000-0x3FFFFFFF	Unused.
0x20000000-0x2000FFFF (64KB)	RAM.
0x01010000-0x1FFFFFFF	Unused.
0x01000000-0x0100FFFF (64KB)	Optional boot loader memory. Actual size 4KB, access above 4KB wraps round.
0x00010000-0x00FFFFFF	Unused.
0x00000000-0x0000FFFF (64KB)	Program memory.

2.5.2 APB subsystem memory map

Table 2-8 describes the peripherals in the APB subsystem.

Table 2-8 APB subsystem peripherals

Address	Item	Notes
0x4000F000-0x4000FFFF	APB expansion port 15	Connected to 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
0x40009000-0x4000AFFF	Not used	Ports on APB slave multiplexer disabled
0x40008000-0x40008FFF	Watchdog	-
0x40007000-0x40007FFF	Not used	Port on APB slave multiplexer disabled
0x40006000-0x40006FFF	UART2	`Stdout` text message for simulations
0x40005000-0x40005FFF	UART1	-
0x40004000-0x40004FFF	UART0	-
0x40003000-0x40003FFF	Not used	Port on APB slave multiplexer disabled
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 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 2-11.

2.6.1 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.

2.6.2 System controller block diagram

Figure 2-3 shows the example system controller.

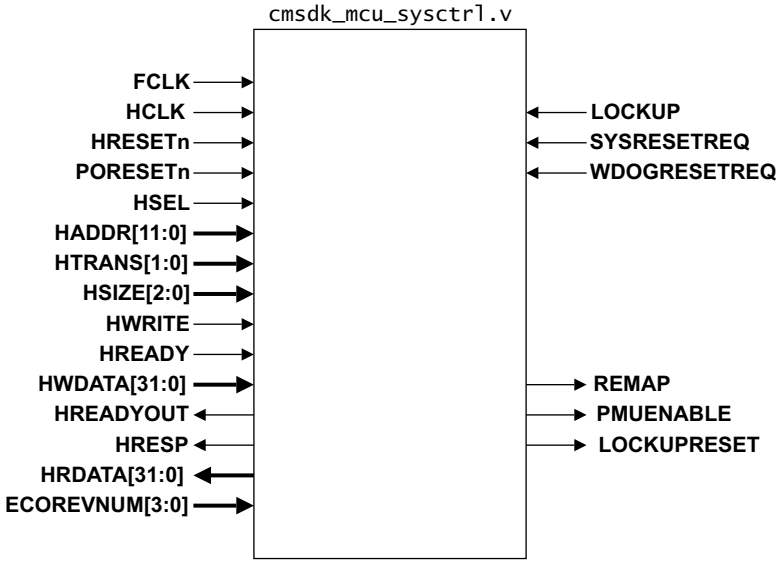


Figure 2-3 Example system controller

Table 2-9 shows the non-AHB signals of the system controller.

Table 2-9 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	Enable 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. It indicates the revision changes during an *Engineering Change Order* (ECO) of the chip design process. You can tie this signal LOW, or connect it to special tie-off cells so that you can change the ECO revision number in a silicon netlist, or at a lower level, for example the silicon mask.

2.6.3 Programmers model

Table 2-10 describes the system controller programmers model.

Table 2-10 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: **1** Enable PMU. If not present the value of this bit is ignored. **0** Disable PMU. Not available for 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].

Table 2-10 System controller programmers model (continued)

Address	Name	Type	Reset	Descriptions
0x4001FFE8	PID2	RO	0x1B	Peripheral ID 2. [7:4] revision. [3] jedec_used. [2:0] jep106_id_6_4.
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.

2.7 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. Figure 2-4 shows the interface of the example microcontroller.

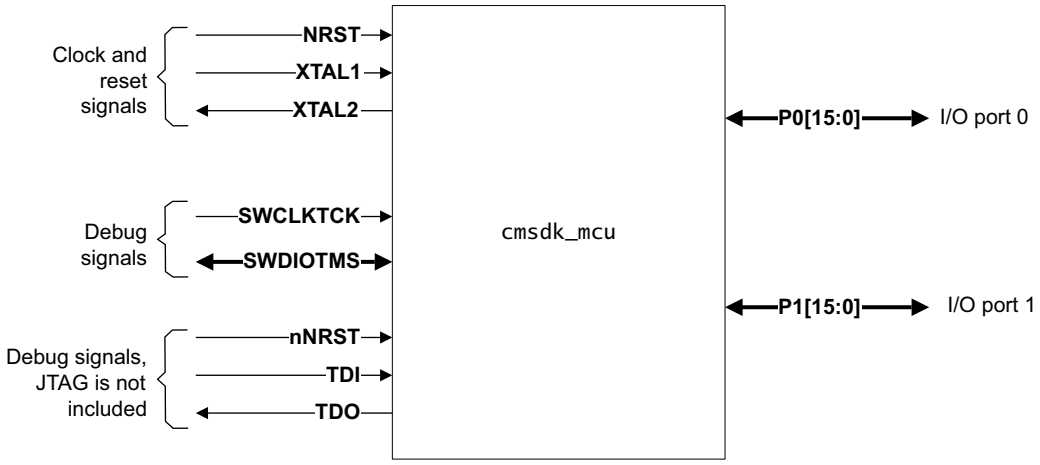


Figure 2-4 Example microcontroller interface

Table 2-11 describes the I/O of the example *MicroController Unit* (MCU).

Table 2-11 Example MCU I/O

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

a. This signal is always present and connects to a dummy DAP module. The dummy DAP is present to show how it is integrated in the full processor.

b. This signal is only present when you select the JTAG option.

Table 2-12 shows the alternate functions of the GPIO 1 ports that support pin multiplexing.

Table 2-12 GPIO alternate functions

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

2.8 Interrupts and event functions

The example system contains:

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

Note

The Cortex-M0 DesignStart only supports 32 interrupts.

2.8.1 Interrupt assignments

Table 2-13 describes the interrupt assignments.

Table 2-13 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	GPIO 0 combined interrupt for AHB GPIO and I/O port GPIO
7	GPIO 1 combined interrupt for AHB GPIO and I/O port GPIO
8	Timer 0
9	Timer 1
10	Dual timer
11	Not used
12	UART 0 overflow interrupt
13	UART 1 overflow interrupt
14	UART 2 overflow interrupt
15	Not used
16-31	GPIO 0 individual interrupts

2.8.2 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 2-5 on page 2-16 shows an example circuit that performs this synchronization.

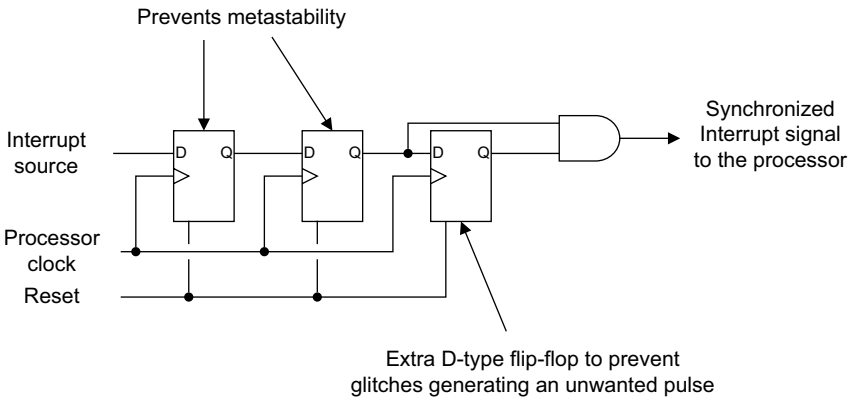


Figure 2-5 IRQ synchronizer

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.

2.8.3 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.

2.9 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.

Figure 2-6 shows the clock and reset operation of the example microcontroller.

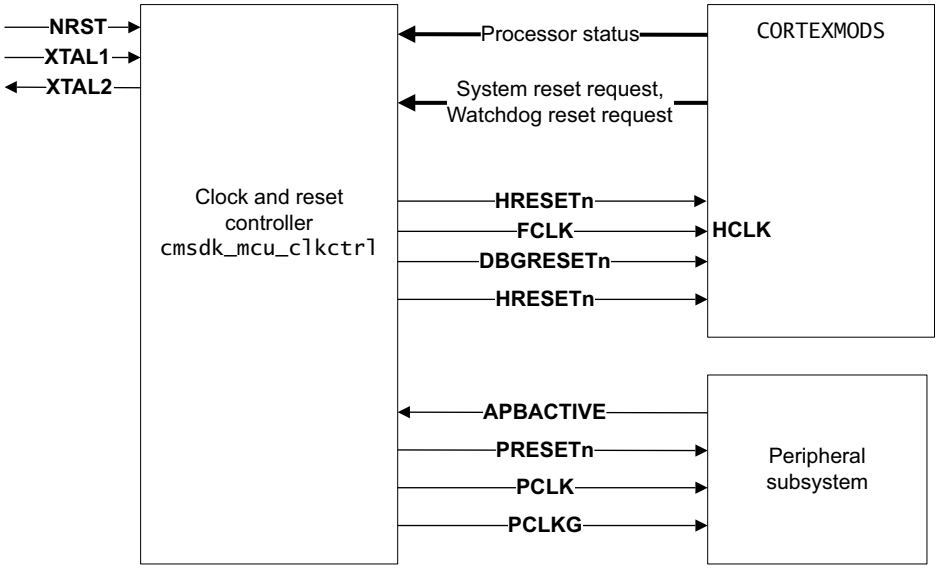


Figure 2-6 Example microcontroller clock and reset operation

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**.

2.10 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 2-14 describes the bit field values of the *SysTick Calibration Value Register* (SYST_CALIB).

Table 2-14 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 Testbench

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

3.1 About the testbench design

The example system includes a testbench to enable you to simulate the example microcontroller with a supported Verilog simulator.

The testbench includes:

- A loop back connection for UART testing.
- A clock and reset generator.
- Text message capture by the UART.

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

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

UART0 connects to UART1 in a crossover arrangement so you can test the serial communication. The serial output of UART2 is connected to a UART capture module that can generate text messages during simulation. See the `cmsdk_uart_capture.v` file.

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 UART2. 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 UART2. 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.

4.2 Files and directory structure

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

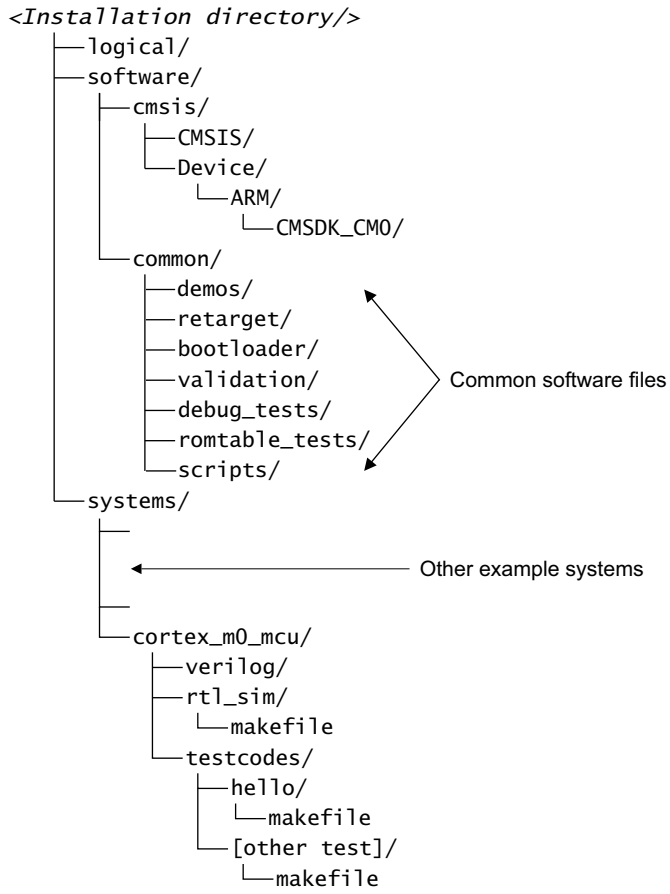


Figure 4-1 Directories for simulation

Table 4-1 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.
`software/common/demos`	C program codes for demonstration.
`software/common/validation`	C program codes for functional tests.

Table 4-1 Installation directory information (continued)

Directory name	Directory contents
`software/common/bootloader`	Example boot loader.
`software/common/dhry`	Dhrystone demonstration.
`software/common/retarget`	Support files to handle printing.
`software/common/scripts`	Linker scripts.

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*.

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/` 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.

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. The process uses a Verilog command file that the value of parameter `CPU_PRODUCT` determines. 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

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/cortex_m0_mcu/testcodes` directory. Inside each of these directories is a makefile for software compilation. The makefiles support ARM DS-5 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 Development Suite. `gcc` ARM GCC. `keil` Keil Microcontroller Development Suite. 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.

Table 4-3 Makefile settings (continued)

Variable	Descriptions
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*.

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 shows the interaction of the makefiles.

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.

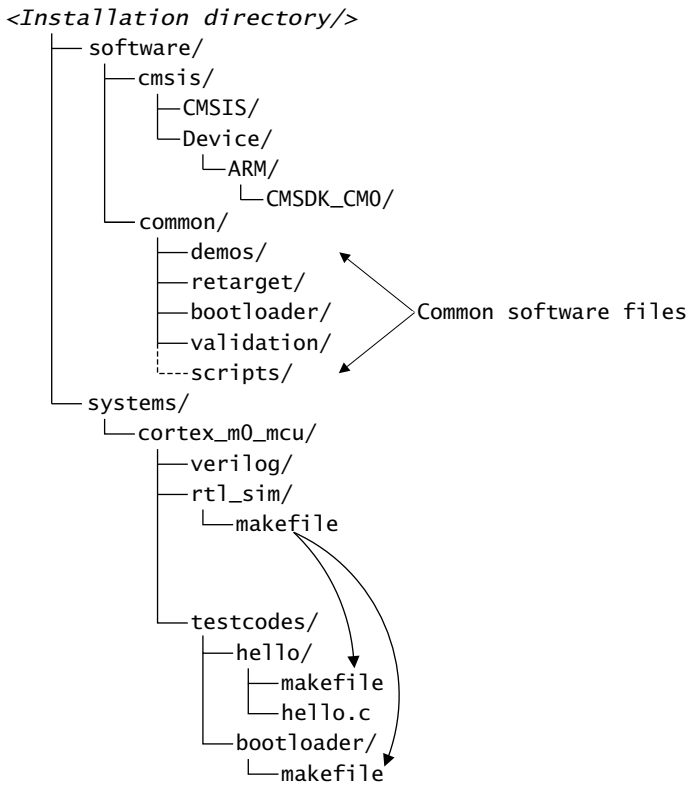


Figure 4-2 Interaction of makefiles

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 UART2. 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.

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
```

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.
`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.
`default_slaves_tests`	Tests the default slave activation. It accesses invalid memory locations.
`gpio_driver_tests`	Simple test for the GPIO device driver functions.
`timer_driver_tests`	Simple test for the simple timer device driver functions.
`uart_driver_tests`	Simple test for the UART device driver functions.
`apb_mux_tests`	Simple test for the APB slave multiplexer.
`memory_tests`	Simple test for the system memory map.

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 the Keil MDK-ARM. The example software package includes a precompiled hex file of the RTX demonstration test, therefore you can simulate this test without a Keil MDK-ARM 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/cortex_m0_mcu/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 UART2. The `tb_uart_capture` device in the testbench captures the text and outputs it to the simulation console during the simulation.

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 UART2 and GPIO1 (for pin multiplexing) for text message output.
`char UartPutc(unsigned char my_ch)`	Output a single character to UART 2.
`char UartGetc(void)`	Read a character from UART.
`char UartEndSimulation(void)`	Terminate the simulation by sending value `0x4` to UART 2. When `tb_uart_capture` receives this data it stops the simulation.

Chapter 6
Synthesis

This chapter describes how to run synthesis for the example system. It contains the following sections:

6.1 Implementation overview

The design kit includes a set of example scripts to enable synthesis of the `cmsdk_mcu_system`. The example scripts support Synopsys Design Compiler. The scripts in the design kit provide a simple setup to enable you to quickly obtain area and timing information from the design:

- They do not include all required steps for a complete implementation flow.
- They do not support SRPG implementation and handle the whole design as one single power domain.
- They must not be used for design signoff.

Three main steps are provided in the scripts:

1. Synthesis, which is handled by `cmsdk_mcu_system_syn.tcl`.
2. DFT scan insertion, which is handled by `cmsdk_mcu_system_dft.tcl`.
3. *Logical Equivalence Checking* (LEC) of the netlist against the Verilog model, which is handled by `cmsdk_mcu_system_fm.tcl`.

Note

This script does not handle *Automatic Test Pattern Generation* (ATPG).

The design kit uses the design level `cmsdk_mcu_system` because it contains all parts of the systems except the blocks that might affect DFT, for example, reset and clock generation logic, and memories. You must synthesize the clock and reset generation logic separately to avoid issues with scan test control. In addition, you might want to modify the clock and reset logic, and the PMU to your own requirements for system-level power management.

The memory blocks included in the design kit are behavioral model, and therefore not suitable for synthesis.

6.2 Directory structure and files

Figure 6-1 shows the file directories in the synthesis environment.

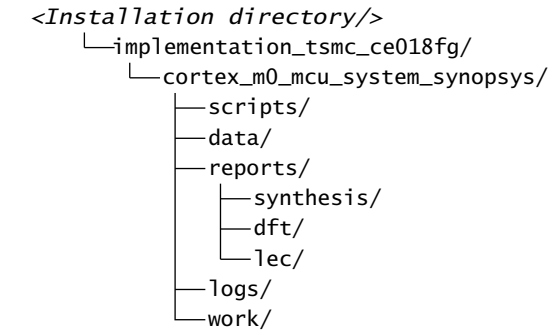


Figure 6-1 Implementation directories

Table 6-1 shows the directories in the synthesis environment.

Table 6-1 Implementation directories

Directories	Descriptions
`scripts/`	Location of the synthesis scripts.
`data/`	Location of the data generated from the synthesis process.
`reports/`	Location of the report files.
`logs/`	Location of the synthesis log file.
`work/`	Location of the temporary files generated from the elaboration of the design. ARM recommends that you clear the contents of this directory before you run a new synthesis.

Table 6-2 shows the contents of the `scripts/` directory.

Table 6-2 Contents of the scripts/ directory

Files	Descriptions
`cmsdk_mcu_system_syn.tcl`	Main script for the synthesis process
`cmsdk_mcu_system_dft.tcl`	Main script for the DFT scan chain insertion process
`cmsdk_mcu_system_tech.tcl`	Setup for the cell library and technology specific parameters
`cmsdk_mcu_system_verilog.tcl`	Specifies the Verilog files for synthesis, including the library-specific clock gate model
`cmsdk_mcu_system_verilog-rtl.tcl`	Specifies the Verilog files for LEC, including the generic RTL clock gate model
`design_config.tcl`	Configuration of the synthesis
`cmsdk_mcu_system_clocks.tcl`	Clock generation script
`cmsdk_mcu_system_constraints.tcl`	Constraints of the design, for example, timing
`cmsdk_mcu_system_fm.tcl`	Main script for *Logical Equivalence Checking* (LEC)
`cmsdk_mcu_system_reports.tcl`	Main script for the design report generation

6.3 Implementation flow

The implementation flow includes the following steps:

1. Customize the synthesis script for the targeted cell library.
2. Customize the synthesis script for the timing constraints and configuration.
3. Perform the synthesis.
4. Review the result.

You must customize the synthesis scripts before starting synthesis, as Table 6-3 shows.

Table 6-3 Synthesis script descriptions

Script name	Description
`cmsdk_mcu_system_tech.tcl`	Update this file to match the cell library installation path in your environment. Also, update this file if you use a different semiconductor process or use different process-related constraints.
`cmsdk_mcu_system_verilog.tcl`	Update this file to define the design files you want to synthesize.
`cmsdk_mcu_system_constraints.tcl`	Update this file if your timing requirements of the design are different from the default settings, for example the input and output constraints.
`design_config.tcl`	Update this file if you change the configuration of your design.

6.4 Timing constraints

This section describes the following timing issues of a design:

- *Maximum frequency*.
- *Input and output delay*
- *Combinatorial paths* on page 6-7.
- *Running the makefile* on page 6-7.

6.4.1 Maximum frequency

The default synthesis scripts target a frequency of 50MHz on a 0.18µm *Ultra Low Leakage* (ULL) process. Although the Cortex-M0 processor is capable of running at higher frequencies, the extra bus infrastructure results in longer timing paths.

You can also increase the maximum clock frequency if you relax the timing constraints of the memory interface.

6.4.2 Input and output delay

The input delay and output delay of the interface ports define their timing constraints. Figure 6-2 shows the input delays and output delays of the interface ports.

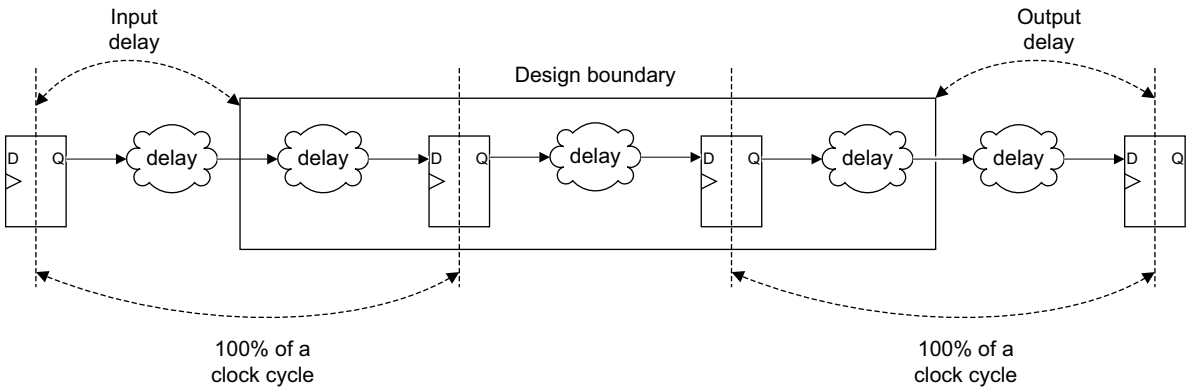


Figure 6-2 Input and output delays

Table 6-4 on page 6-6 describe the default input and output delay setting in the example synthesis script. The higher the percentage value, the tighter the timing requirement.

Table 6-4 shows the timing constraints for the **FCLK** domain and **HCLK** domain interface signals.

Table 6-4 Timing constraints for FCLK and HCLK

Interface	Signals	Type	Descriptions	Input delay	Output delay
Memory AHB	**HADDR[31:0]**	Output	Address		50%
	HTRANS[1:0]	Output	Transfer type		50%
	HSIZE[2:0]	Output	Transfer size		50%
	HWRITE	Output	Transfer direction		50%
	HWDATA[31:0]	Output	Write data		60%
	HREADY	Output	Transfer ready		30%
SRAM AHB	**sram_hsel**	Output	Device select for SRAM		50%
	sram_hreadyout	Input	SRAM ready	40%	
	sram_hrdata[31:0]	Input	SRAM read data	60%	
	sram_hresp	Input	SRAM response	60%	
CPU status	**SYSRESETREQ**	Output	System reset request		50%
	LOCKUP	Output	Core is locked up		50%
		Output			
		Output			
Reset	**WDOGRESETREQ**	Output	Watchdog reset request		50%
	LOCKUPRESET	Output	Reset system when locked up		50%
	HRESETn	Input	AHB reset	60%	
	PORESETn	Input	Power on reset	60%	
	DBGRESETn	Input	Debug reset	60%	
	PRESETn	Input	APB reset	60%	
UART	**uart0_rxd**	Input	UART0 receive data	60%	
	uart0_txd	Output	UART0 transmit data		60%
	uart0_txen	Output	UART0 transmit enable		60%
	uart1_rxd	Input	UART1 receive data	60%	
	uart1_txd	Output	UART1 transmit data		60%
	uart1_txen	Output	UART1 transmit enable		60%
	uart2_rxd	Input	UART2 receive data	60%	
	uart2_txd	Output	UART2 transmit data		60%
	uart2_txen	Output	UART2 transmit enable		60%
Timer	**timer0_extin**	Input	Timer0 external input	60%	
	timer1_extin	Input	Timer1 external input	60%	
GPIO	**p0_in[15:0]**	Input	GPIO port0 input	60%	
	p0_out[15:0]	Output	GPIO port0 output		60%
	p0_outend[15:0]	Output	GPIO port0 output enable		60%
	p0_altfunc[15:0]	Output	GPIO port0 alternate function		60%
	p1_in[15:0]	Input	GPIO port1 input	60%	
	p1_out[15:0]	Output	GPIO port1 output		60%
	p1_outend[15:0]	Output	GPIO port1 output enable		60%
	p1_altfunc[15:0]	Output	GPIO port1 alternate function		60%

6.4.3 Combinatorial paths

Figure 6-3 shows a combinatorial path in the design that affects the input and output constraints.

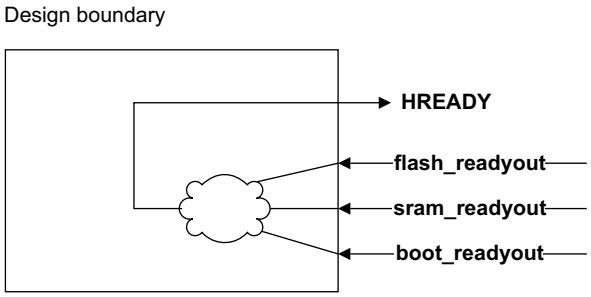


Figure 6-3 HREADY path

Because of this signal path, the example script sets the **HREADY** output delay value to only 30%.

6.4.4 Running the makefile

The makefile is located in `implementation_tsmc_ce018fg/cortex_m0_mcu_system_synopsys`. This makefile performs synthesis and *Design For Test* (DFT), and generates a log file. Use the makefile as follows:

`make synthesis` Performs synthesis using the topological features.

`make dft` Inserts DFT into the netlist after synthesis.

`make lec_synthesis` Performs LEC on the synthesized netlist.

`make lec_dft` Performs LEC on the DFT netlist.

`make front` Performs both the synthesis and DFT steps.

`make analysis` Performs both the LEC steps on the synthesized and DFT netlists.

`make all` Performs all steps, that is, synthesis, DFT, LEC synthesis, and LEC DFT.

`make clean` Cleans all the report directories, log files, and database information to be removed, ready for a new run.

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	-	-