Arm® Cortex®-M0 and Cortex-M0+ System Design Kit

Revision: r1p1

Example System Guide

Confidential

ARM DUI 0559D (ID110117)

Arm Cortex-M0 and Cortex-M0+ System Design Kit
Example System Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
17 March 2011	A	Non-Confidential	First release for r0p0
15 June 2011	B	Confidential	Second release for r0p0
19 April 2013	C	Confidential	First release for r1p0
31 October 2017	D	Confidential	First release for r1p1

Proprietary Notice

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

Confidentiality Status

This document is Confidential. This document may only be used and distributed 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 and Cortex-M0+ System Design Kit Example System Guide

Preface

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

About this book

This book is for the Cortex-M0 and Cortex-M0+ System 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 and Cortex-M0+ System 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 *Debug tester*

Read this for a description of the debug components in the design kit.

Appendix B *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

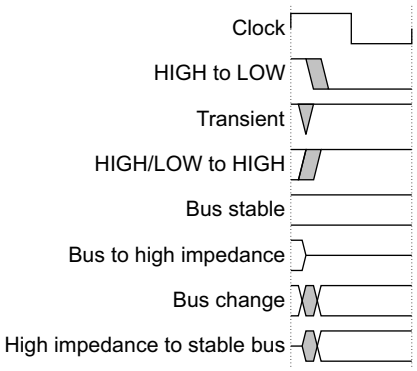
The following table describes the typographical conventions:

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.

Lowercase 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 Integration and Implementation Manual* (ARM DII 0238).
- *Arm® Cortex®-M0 User Guide Reference Material* (ARM DUI 0467).
- *Arm® Cortex®-M0 Technical Reference Manual* (ARM DDI 0432).
- *Arm® Cortex®-M0+ Devices Generic User Guide* (ARM DUI 0662).
- *Arm® Cortex®-M0+ Integration and Implementation Manual* (ARM DIT 0032).

- *Arm® Cortex®-M0+ User Guide Reference Material* (ARM DUI 0605).
- *Arm® Cortex®-M0+ Technical Reference Manual* (ARM DDI 0484).

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 0559D.
- 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 and Cortex-M0+ System Design Kit. It contains the following sections:

1.1 About the Cortex-M0 and Cortex-M0+ System Design Kit

The Cortex-M0 and Cortex-M0+ System Design Kit provides:

- An example system-level design for the Arm Cortex-M0 and Cortex-M0+ processors.
- Reusable AMBA components for system-level development.

For information on the AMBA components that the design kit uses, see the *Arm® Cortex®-M System Design Kit Technical Reference Manual*.

This document describes an example system for the Cortex-M0 and Cortex-M0+ processors. The example system is a simple microcontroller design.

1.2 Cortex-M0 and Cortex-M0+ System 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, and the AHB-Lite and APB protocol checkers.
`systems`	Design files, testbench files, and simulation setup files for the example system.
`implementation`	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. • The debug tester software for all the processors.
`document`	Documentation files.
`cores`	This is the default location for processor core RTL files. You can change this location by modifying the simulation and synthesis setup. The design kit does not include the processor RTL files.

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

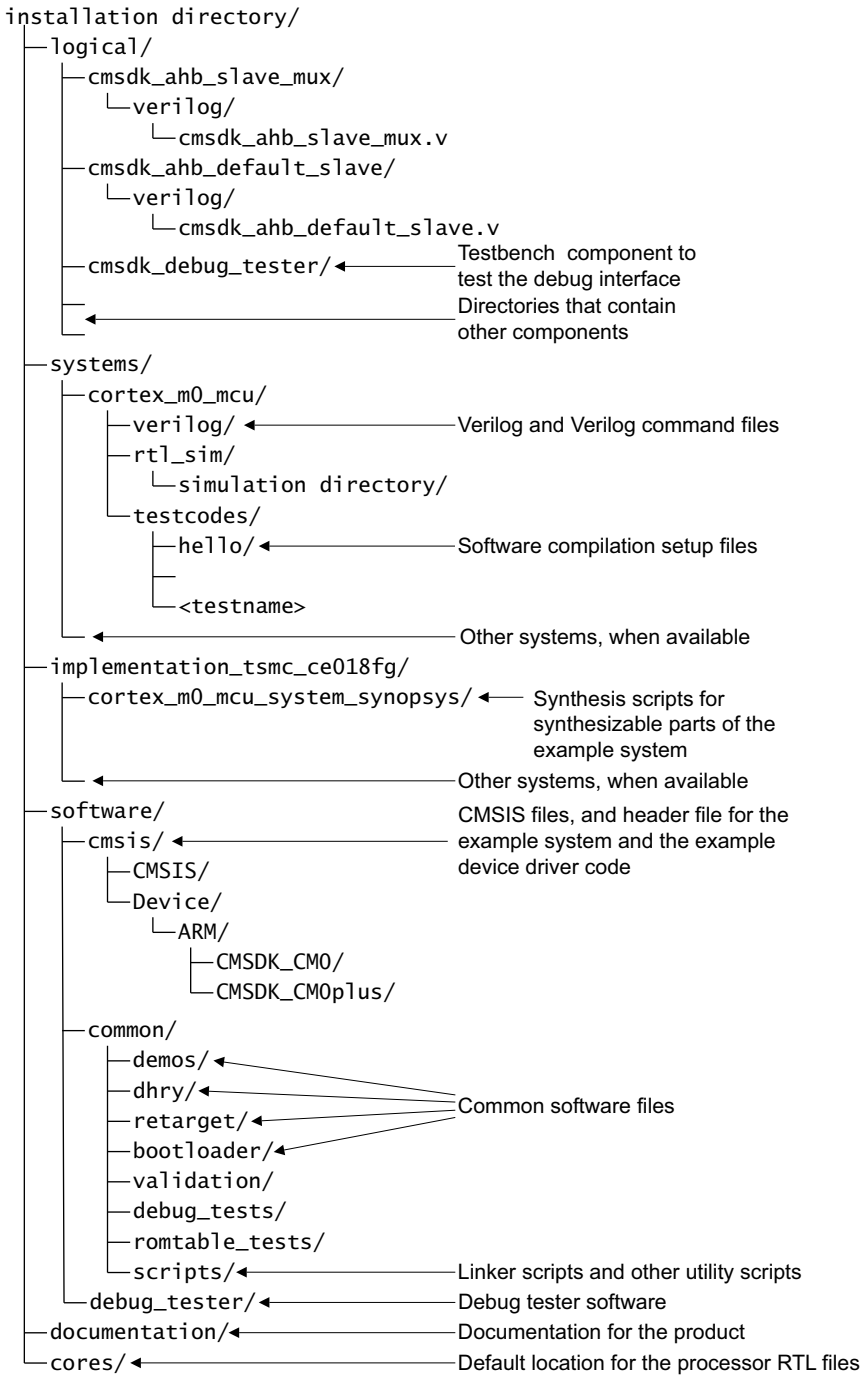


Figure 1-1 Directory structure

1.3 Limitations of the design kit

This section describes the limitations of the design kit.

1.3.1 Deliverables

The design kit does not include:

- Cortex-M0 processor RTL code.
- Cortex-M0+ processor RTL code.
- *Direct Memory Access* (DMA) Controller code.
- Software compilation tools.

You must license these products separately.

1.3.2 Processor support

The design kit supports the Cortex-M0 and Cortex-M0+ processors.

1.3.3 Endian support

The example system and its peripherals in the design kit are *little-endian*. The example system also supports limited *big-endian* (BE)-8 byte-invariant mode for the following components:

AHB infrastructure components except the AHB bit band wrapper

These components are endian-independent.

AHB bit band wrapper

This component contains a Verilog parameter to enable you to operate in a big-endian environment.

GPIO The AHB and I/O port GPIO support little-endian operation. They have a Verilog parameter that you can configure to enable you to use them and their existing device driver software in a big-endian environment. However, this increases the gate count of the design because it introduces additional logic to control the byte lane swapping.

Memory components

The behavioral models of the design kit components are designed to be little-endian. However, they can also work in a big-endian system if the system accesses each memory location with consistent transfer sizes.

APB peripherals

The APB peripherals are designed to be little-endian. The APB subsystem provides a Verilog parameter that introduces additional endian conversion logic to enable you to use these components and their device driver software in a big-endian environment. Arm recommends that you do not use this parameter because it adds extra hardware. To accomplish big-endian product design, Arm recommends that you modify the peripherals and device drivers to use a big-endian programmers model.

If you require a big-endian system design, Arm recommends that you redesign the peripherals and memory blocks to achieve the lowest hardware overhead.

1.3.4 Big-endian support with Arm GCC

By default, the GNU Tools for Arm embedded processors only support little-endian configurations. If you use the Arm GCC in a big-endian system, you must rebuild the runtime libraries. For information on rebuilding the toolchain, see Launchpad https://launchpad.net/gcc-arm-embedded. You can also contact Arm for additional help.

1.3.5 Platform

This release of the Cortex-M0 and Cortex-M0+ System 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 or 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.
- Debug connection.

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

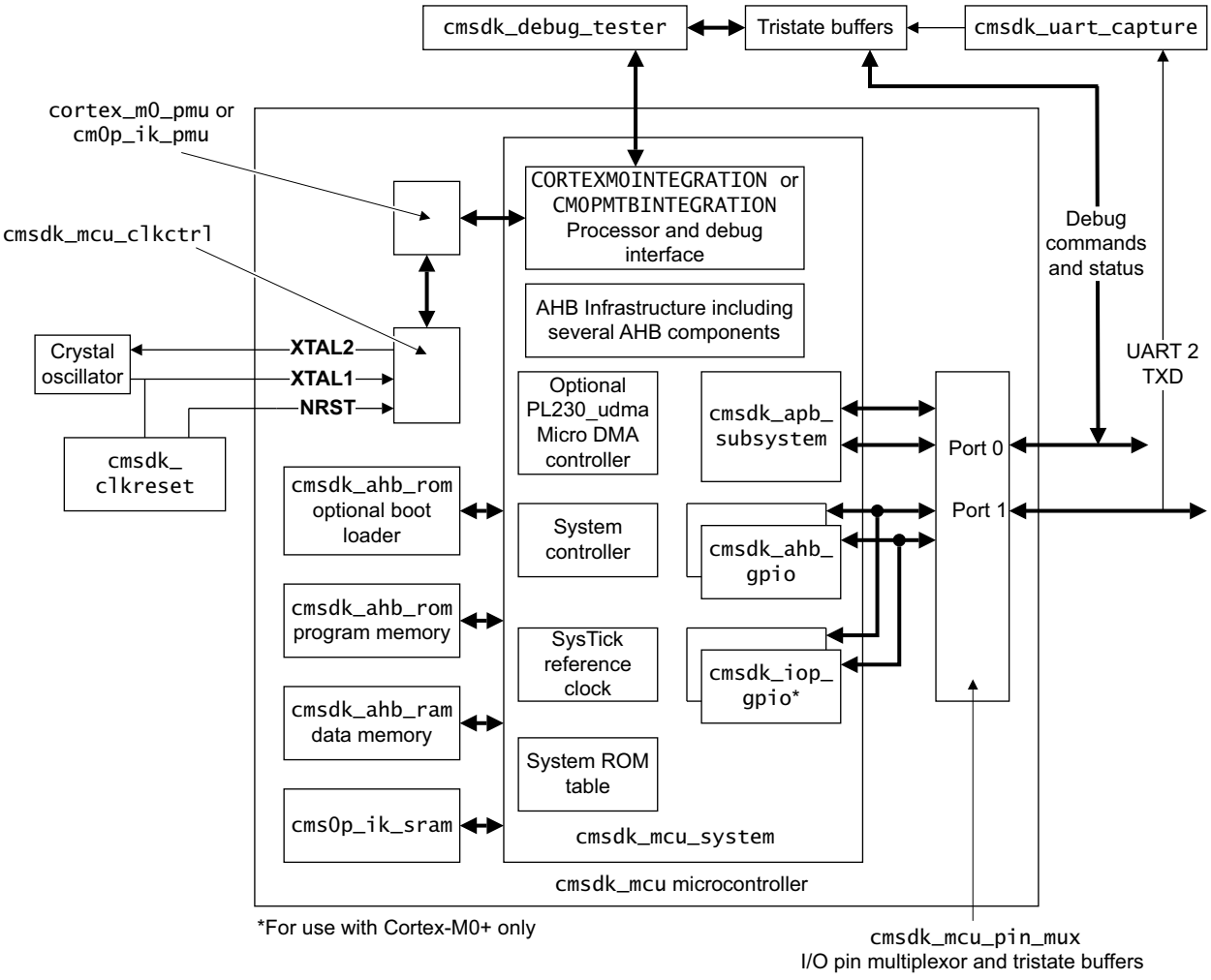


Figure 2-1 Example microcontroller system top-level view

For the Cortex-M0+ system, the following components from the Cortex-M0+ integration kit are reused:

- `cm0p_ik_rst_ctl`.
- `cm0p_ik_sram`.

- `cm0p_ik_pmu`.

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 or the Cortex-M0+ processor.
`CORTEXM0INTEGRATION`	The Cortex-M0 Integration Layer. This is part of the Cortex-M0 deliverable. It integrates the Cortex-M0 processor and the debug interface module.
`CM0PMTBINTEGRATION`	The Cortex-M0+ Integration Layer. This is part of the Cortex-M0+ deliverable. It integrates the Cortex-M0+ processor, CoreSight™ MTB-M0+ micro trace buffer, and the debug interface module. The CoreSight -MTB-M0+ is licensed separately from the Cortex-M0+ processor. It is not included in this deliverable.
`cmsdk_apb_subsystem`	A subsystem of APB peripherals and APB infrastructure.
System controller	Contains programmable registers for system control, for example memory remap and power management enable.
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_iop_gpio`	A low-latency GPIO. Each GPIO module provides 16 I/O pins. Only used in Cortex-M0+.
PL230 DMA Controller	An optional instantiation of the Arm CoreLink DMA-230 Micro DMA Controller. The DMA-230 is not included in this deliverable, and you must license it separately.
`cortex_m0_pmu`	An optional Cortex-M0 *Power Management Unit* (PMU). This is included in the Cortex-M0 deliverable. It is not included in this deliverable.
`cm0p_ik_pmu`	An optional Cortex-M0+ PMU. This is included in the Cortex-M0+ deliverable. It is not included in this deliverable.
`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 or Cortex-M0+ based system.
`cmsdk_mtb_sync.v`	Includes the synchronizers for the CoreSight M0+ MTB signals, that is, **TSTART** and **TSTOP**.
`cmsdk_mcu_addr_decode`	Generates the **HSELS** for each memory mapped component based on the CMSDK address map.
`cm0p_ik_sram`	A synchronous SRAM model for the CoreSight MTB-M0+. Only used in Cortex-M0+.

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`. This unit also controls the tristate buffers that pass debug commands and status information between the microcontroller and the debug tester. It does this by sending escape codes to the capture module.
`cmsdk_debug_tester`	The debug tester is a separate processor-based system that generates debug activities to test the debug connection. To run these tests, the microcontroller communicates with the debug tester through GPIO 0 and the I/O of port 0. By default, the testbench disables this communication with tristate buffers, so the microcontroller can test the I/O port functionalities without starting a debug test.

You can configure the system in a number of different ways. Figure 2-2 shows the interfaces of the Cortex-M0 example system that includes a DMA controller.

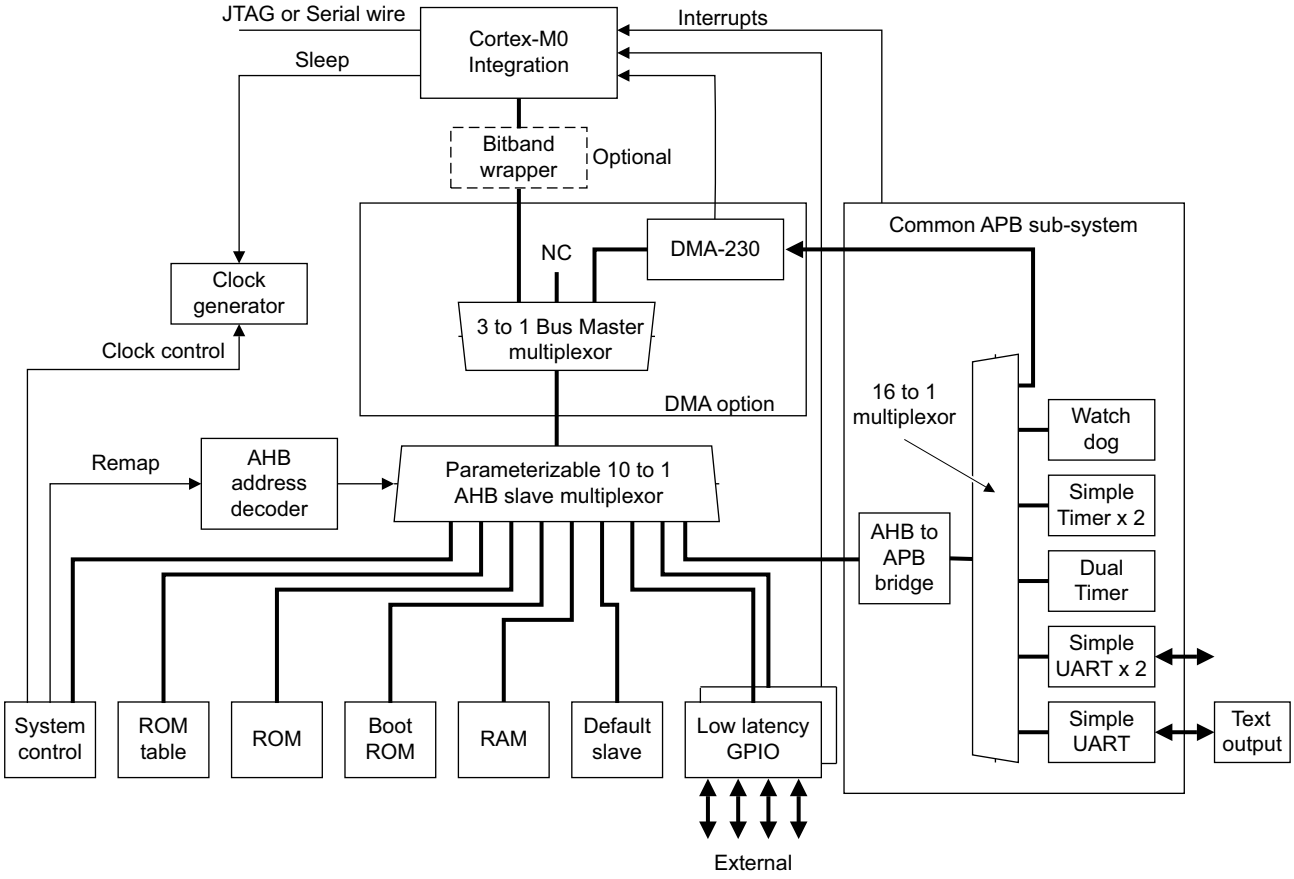


Figure 2-2 Cortex-M0 example system with DMA controller

Figure 2-3 on page 2-5 shows the interfaces of the Cortex-M0+ example system that includes a DMA controller.

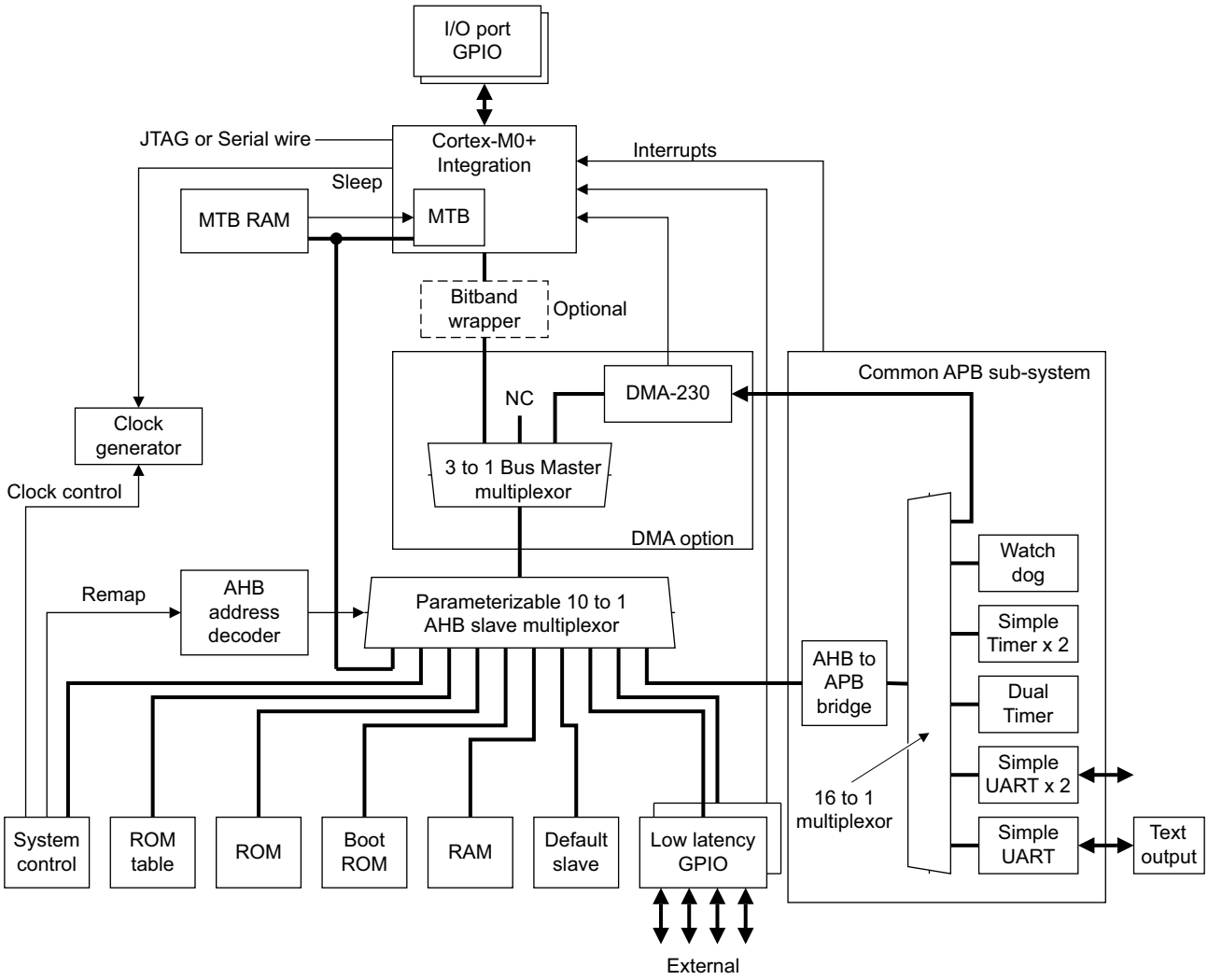


Figure 2-3 Cortex-M0+ example system with DMA controller

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

Figure 2-4 on page 2-6 shows the interfaces of the Cortex-M0 example system without a DMA controller.

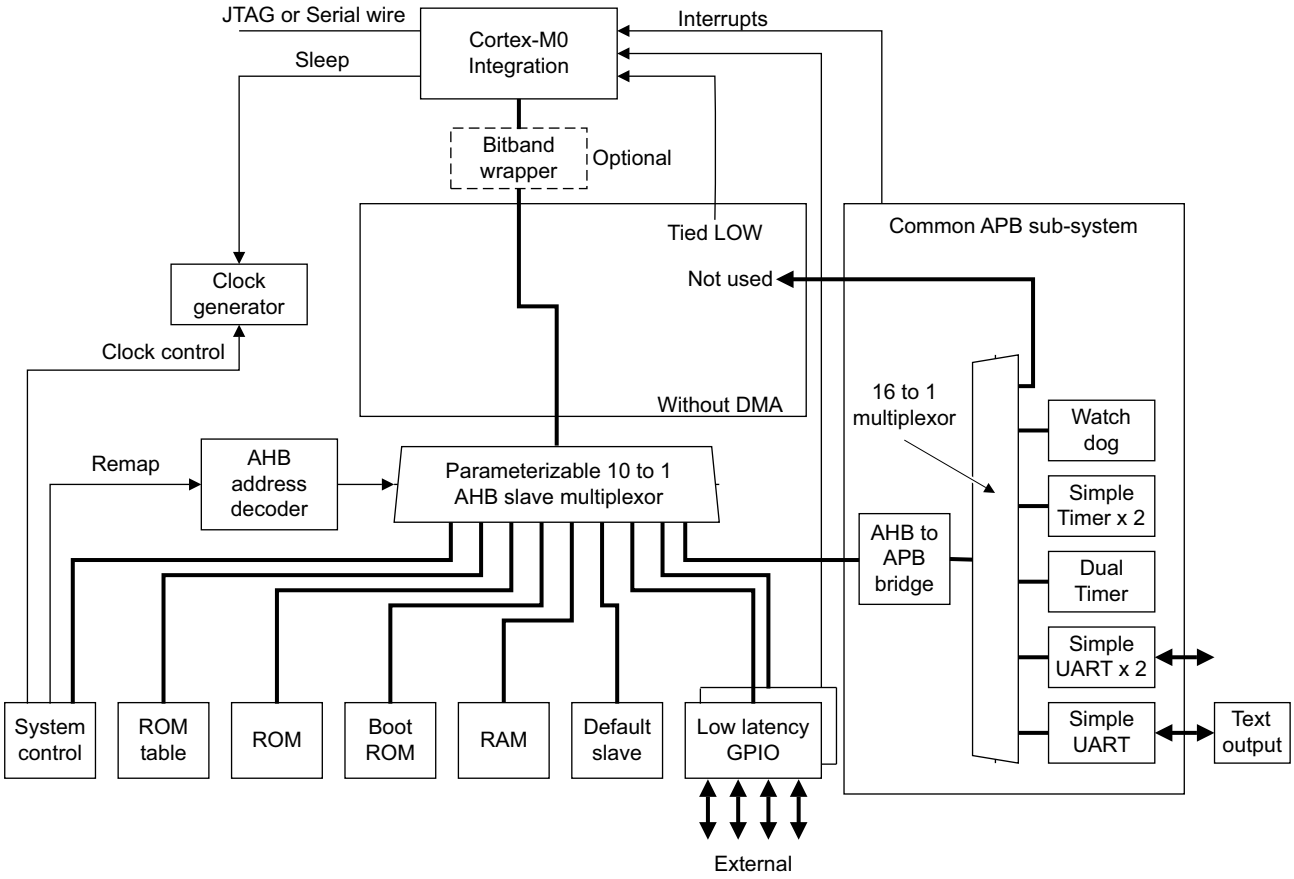


Figure 2-4 Cortex-M0 example system without DMA controller

Figure 2-5 on page 2-7 shows the interfaces of the Cortex-M0+ example system without a DMA controller.

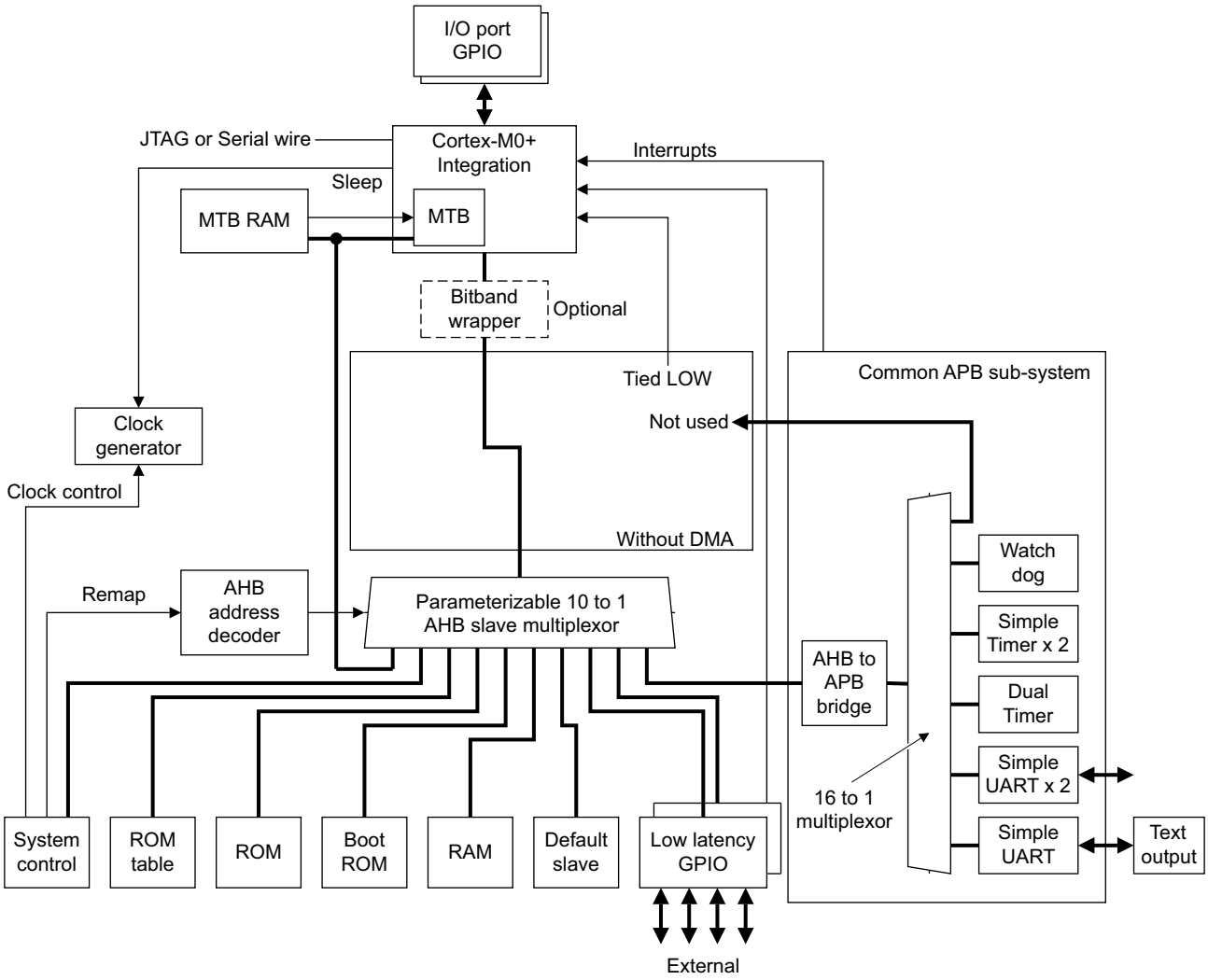


Figure 2-5 Cortex-M0+ example system without DMA controller

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* on page 2-9.

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 and Cortex-M0+ microcontroller.

Table 2-4 Verilog files for the Cortex-M0 and Cortex-M0+ microcontroller

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_mtb_sync.v`	Synchronizer for Cortex-M0+ MTB trace start/stop control signals
`cmsdk_iop_interconnect.v`	Address decode and interconnection for Cortex-M0+ single cycle I/O port
`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 and 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 and control of debug tester communication
`tbench_M0.vc`	Verilog command file for Cortex-M0
`tbench_M0P.vc`	Verilog command file for Cortex-M0+

The testbench also contains a debug tester that tests the debug connectivity. Table 2-6 describes the Verilog files of the debug tester.

Table 2-6 Verilog files of the debug tester

File	Description
`cmsdk_debug_tester.v`	Top level of the debug tester.
`cmsdk_debug_tester_ahb_interconnect.v`	AHB interconnection inside the debug tester. The debug tester is a Cortex-M0 or Cortex-M0+ based system that requires its own AHB infrastructure.
`cmsdk_ahb_default_slave.v`	CMSDK AHB default slave that is used by the debug tester.
`cmsdk_ahb_rom.v`	Program memory for the debug tester.
`cmsdk_ahb_ram.v`	CMSDK AHB SRAM used by the debug tester.
`cmsdk_ahb_gpio.v`	CMSDK AHB GPIO

The debug tester Verilog files are located in the `logical/cmsdk_debug_tester/verilog/` directory.

The debug tester has its own program image. It is stored in `debug_tester_cm0.hex` for the Cortex-M0 processor or `debug_tester_cm0plus.hex` for the Cortex-M0+ processor. The file is located in the `software/debug_tester` directory. This directory also contains the source code for the debug tester program, and the compilation makefile of the program for the Arm *Development Studio* (DS-5) and Arm GCC, and the compile project setup for the Keil® *Microcontroller Development Kit* (MDK).

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 locations

The default location of the Verilog RTL files for the processors is a subdirectory called `cores`. For example:

- Cortex-M0 default RTL path is `cores/at510_cortexm0_r0p0_03rel2/logical/`
- Cortex-M0+ default RTL path is `cores/at590_cortexm0p_r0p1/logical/`

2.3.1 Modifying the location of the processor files for simulation

To modify the location of the Cortex-M0 default files for simulation, edit the Verilog command file `tbench_M0.vc`.

To modify the location of the Cortex-M0+ default files for simulation, edit the Verilog command file `tbench_M0P.vc`.

2.3.2 Modifying the location of the processor files for synthesis

Modify the synthesis script file `cmsdk_mcu_system_verilog.tcl` to match the location of the source files.

2.3.3 Tarmac

The logging of instructions is provided in the Cortex-M0 and Cortex-M0+ using a simulation model called Tarmac. This is enabled by default. You can disable it by editing the appropriate `vc` file and commenting out the `USE_TARMAC` define.

2.4 Configuration options

The example microcontroller system contains several configurable options. You can define these options in the following ways:

- *Preprocessing definitions*.
- *Verilog parameters for Cortex-M0* on page 2-13.
- *Verilog parameters for Cortex-M0+* on page 2-14.
- *Changing the processor type* on page 2-15.

2.4.1 Preprocessing definitions

The file `cortex_m0_mcu/verilog/cmsdk_mcu_defs.v` contains a number of Verilog preprocessing definitions. To remove a definition, comment-out the line of Verilog code that describes the preprocessing definitions. Table 2-7 describes the Verilog preprocessing definitions.

Table 2-7 Verilog preprocessing definitions

Preprocessing macro	Descriptions
`ARM_CMSDK_INCLUDE_BITBAND`	Include this macro to add a bitband wrapper module to the example Cortex-M0 or Cortex-M0+ system. It enables these processors to have the same bitband capability as the Cortex-M3 and Cortex-M4 processors. Comment this out to remove the bitband wrapper and reduce the gate count.
`ARM_CMSDK_INCLUDE_DEBUG_TESTER`	This macro includes the debug tester component that is in the simulation testbench. Remove this macro option if you use a processor that does not have a debug feature.
`ARM_CMSDK_INCLUDE_JTAG`	If the debug feature is available, the debug interface can either use the JTAG protocol or the Serial Wire protocol. Include this macro to specify the JTAG debugger protocol. Remove this macro to specify the Serial Wire debug protocol. If you select the Serial Wire option, the example system removes the redundant JTAG pins from the example microcontroller design.
`ARM_CMSDK_INCLUDE_DMA`	If you include this macro, the example system includes the instantiation of a DMA-230 Micro DMA Controller and an AHB master multiplexer. The DMA-230 is not included in this deliverable, and you must license it separately.
`ARM_CMSDK_INCLUDE_CLKGATE`	If you include this macro, the system: • Sets the `CLKGATE_PRESENT` Verilog parameter of the Cortex-M0 or Cortex-M0+ instantiation HIGH to enable the architectural clock gating feature. • Includes clock gating cells in the clock controller file `cortex_m0_mcu_clkctrl.v` for gated clock generation. • Uses the gated clock, that the example *Power Management Unit* (PMU) generates, for the connection of **HCLK**, **DCLK**, and **SCLK** of the Cortex-M0 or Cortex-M0+. When you select this option and enable the Cortex-M0 or Cortex-M0+ PMU, the SysTick stops during deep sleep mode because the PMU stops **SCLK** during deep sleep mode.
`ARM_CMSDK_SLOWSPEED_PCLK`	If you include this macro, the APB peripheral bus runs at half the speed of the AHB bus. **Note** This option is ignored when the clock gating option, `ARM_CMSDK_INCLUDE_CLKGATE`, or the `CLKGATE_PRESENT` parameter, is not set.
`ARM_CMSDK_BOOT_MEM_TYPE`	Defines the boot loader memory type. The options are: • `AHB_ROM_NONE` for memory not present. This disables the bootloader feature. • `AHB_ROM_BEH_MODEL` for behavioral ROM memory. • `AHB_ROM_FPGA_SRAM_MODEL` for a behavioral FPGA SRAM model with an SRAM wrapper. • `AHB_ROM_FLASH32_MODEL` for behavioral 32-bit flash memory. • `AHB_ROM_FLASH16_MODEL` for behavioral 16-bit flash memory, for use with the Cortex-M0+ processor.

Table 2-7 Verilog preprocessing definitions (continued)

Preprocessing macro	Descriptions
ARM_CMSDK_ROM_MEM_TYPE	Defines the program memory type. The options are: • AHB_ROM_BEH_MODEL for behavioral ROM memory. • AHB_ROM_FPGA_SRAM_MODEL for a behavioral FPGA SRAM model with an SRAM wrapper. • AHB_ROM_FLASH32_MODEL for behavioral 32-bit flash memory. • AHB_ROM_FLASH16_MODEL for behavioral 16-bit flash memory, for use with the Cortex-M0+ processor.
ARM_CMSDK_RAM_MEM_TYPE	Defines the RAM memory type. The options are: • AHB_RAM_BEH_MODEL for behavioral RAM memory. • AHB_RAM_FPGA_SRAM_MODEL for behavioral SRAM model with SRAM wrapper. • AHB_RAM_EXT_SRAM16_MODEL for benchmarking using 16-bit external asynchronous SRAM. • AHB_RAM_EXT_SRAM8_MODEL for benchmarking using 8-bit external asynchronous SRAM.
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*.
ARM_CMSDK_INCLUDE_IOP	Defines the inclusion of the I/O Port GPIO in place of the AHB GPIO because they are mutually exclusive. Only used in Cortex-M0+.
ARM_CMSDK_INCLUDE_MTB	Include this macro to instantiate the MTB and its associated RAM for trace storage. In this configuration, the AHB RAM is removed from the system and the MTB provides the access for data accesses. This enables the design to be kept small because only a single RAM is used for trace and data. Only used in Cortex-M0+. **Note** When the MTB is included, the RAM that is used for the data accesses is shared with the MTB. If you set the address width to be smaller than the default, that is, AWIDTH = 16, the addresses that some tests use to write data to will alias to lower addresses, therefore some tests will not work with the smaller memory. It is also possible that, with a smaller address space, test data accesses and MTB accesses overwrite each other within the shared memory space, so take care to limit each of these accesses to certain ranges of addresses.
ARM_CMSDK_INCLUDE_F16	Configures a Cortex-M0+ based system to use 16-bit Flash. In this configuration, a 32-bit to 16-bit converter, cm0p_32to16_dnsize, is used. This converter is supplied with the Cortex-M0+ integration kit, and does not include burst support.

2.4.2 Verilog parameters for Cortex-M0

If you are using the full version of the Cortex-M0 processor, its Verilog parameters are propagated from the testbench. You can edit file `cortex_m0_mcu/verilog/tb_cmsdk_mcu.v` to modify these Verilog parameters. Table 2-8 describes the parameters.

Table 2-8 Verilog parameters for Cortex-M0

Verilog parameters	Descriptions
`NUMIRQ`	Number of *Interrupt ReQuest* (IRQ) signals. The range is 1-32.
`SMUL`	Small multiplier. This can be one of the following values: **0** Single cycle multiplier. **1** Small multiplier with multicycle operation.
`SYST`	SysTick Timer, it can be one of the following values: **0** No timer. **1** Include the SysTick timer.
`WIC`	*Wake-up Interrupt Controller* (WIC) support. This can be one of the following values: **0** Remove WIC. **1** Enable WIC.
`WICLINES`	Number of interrupt lines that the WIC supports.
`DBG`	Debug configuration. This can be one of the following values: **0** Remove debug feature. **1** Include debug feature.
`BKPT`	Number of breakpoint comparators.
`WPT`	Number of watchpoint comparators.
`BE`	Big-endian. This can be one of the following values: **0** Little-endian. **1** Big-endian.
`RESET_ALL_REGS`	Specifies whether all synchronous states or only the architecturally required state is reset. **0** Only resets the architecturally required state. **1** Resets all synchronous states. **Note** Setting this parameter increases the size of the registers that are not reset by default, and therefore also increases the overall area of the implementation. When some of the registers in the register bank are not reset, Xs might be seen on **WDATA** and **RDATA** during simulation. This is normal behavior.

2.4.3 Verilog parameters for Cortex-M0+

If you are using the full version of the Cortex-M0+ processor, its Verilog parameters are propagated from the testbench. You can edit file `cortex_m0_mcu/verilog/tb_cmsdk_mcu.v` to modify these Verilog parameters. Table 2-9 describes the parameters.

Table 2-9 Verilog parameters for Cortex-M0+

Verilog parameters	Descriptions
NUMIRQ	Number of *Interrupt ReQuest* (IRQ) signals. The range is 0-32.
SMUL	Small multiplier. This can be one of the following values: **0** Single cycle multiplier. **1** Small multiplier with multicycle operation.
SYST	SysTick Timer. This can be one of the following values: **0** No timer. **1** Include the SysTick timer.
WIC	*Wake-up Interrupt Controller* (WIC) support. This can be one of the following values: **0** Remove WIC. **1** Enable WIC feature.
WICLINES	Number of interrupt lines that the WIC supports.
DBG	Debug configuration. This can be one of the following values: **0** Remove debug feature. **1** Include debug feature.
BKPT	Number of breakpoint comparators.
WPT	Number of watchpoint comparators.
BE	Big-endian. This can be one of the following values: **0** Little-endian. **1** Big-endian.
IOP	Single-cycle I/O port. This can be one of the following values: **0** Exclude single-cycle I/O port. **1** Include single-cycle I/O port.
MPU	*Memory Protection Unit* (MPU). This can be one of the following values: **0** No MPU. **8** 8-region MPU.
VTOR	*Vector Table Offset Register* (VTOR). This can be one of the following values: **0** Exclude VTOR. **1** Include VTOR.
IRQDIS	*IRQ Disable* (IRQDIS). IRQDIS[i] disables IRQ[i], for example: `32'h00000000` No IRQ disabled. `32'h0000FFFF` IRQ[15:0] disabled.
MTB	*Micro Trace Buffer* (MTB). This can be one of the following values: **0** Exclude MTB. **1** Include MTB.
AWIDTH	Specifies the MTB SRAM address width.

Table 2-9 Verilog parameters for Cortex-M0+ (continued)

Verilog parameters	Descriptions
USER	User/Privilege support. This can be one of the following values: **0** Exclude User/Privilege support. **1** Include User/Privilege support.
RAR	Specifies whether all synchronous states or only the architecturally required state is reset. **0** Only resets the architecturally required state. **1** Resets all synchronous states. **Note** Setting this parameter increases the size of the registers that are not reset by default, and therefore also increases the overall area of the implementation. When some of the registers in the register bank are not reset, Xs might be seen on **WDATA** and **RDATA** during simulation. This is normal behavior.
HWF	Halfword fetching. This can be one of the following values: **0** Fetch instructions using 32-bit AHB-Lite accesses whenever possible. **1** Fetch instructions using only 16-bit AHB-Lite accesses.

2.4.4 Changing the processor type

The Cortex-M0 and Cortex-M0+ example system supports the Cortex-M0 processor and the Cortex-M0+ processor. To support all designs, the simulation setup provides the following Verilog command files:

`tbench_M0.vc` For the full version of the Cortex-M0 processor, with the processor RTL path pointing to the default location of `cores/at510_cortexm0_r0p0_03rel2/logical/`.

`tbench_M0P.vc` For the full version of the Cortex-M0+ processor, with the processor RTL path pointing to the default location of `cores/at590_cortexm0p_r0p1/logical/`.

For simulation, the makefile `cortex_m0_mcu/rtl_sim/makefile` selects the correct Verilog command file. This makefile also enables you to select your required processor by setting the parameter `CPU_PRODUCT` to `CORTEX_M0`, or `CORTEX_M0PLUS`.

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 2-17.

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 powerup, 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 2-10 describes the AHB memory map with and without the remap function.

Table 2-10 AHB memory map

Address	Without remap	With remap
0xF0220000-0xFFFFFFFF	Unused, except for the private peripheral bus addresses in the Cortex-M0 and Cortex-M0+.	Unused, except for the private peripheral bus addresses in the Cortex-M0 and Cortex-M0+.
0xF0210000-0xF021FFFF (64KB)	MTB RAM[a].	MTB RAM[a].
0xF0201000-0xF021FFFF	Unused.	Unused.
0xF0200000-0xF0200FFF (4KB)	MTB SFR[a].	MTB SFR[a].
0xF0000401-0xF01FFFFF	Unused.	Unused.
0xF0000000-0xF0000400 (4KB)	System ROM table.	System ROM table.
0x40020000-0xEFFFFFFF	Unused, except for the private peripheral bus addresses in the Cortex-M0 and Cortex-M0+.	Unused, except for the private peripheral bus addresses in the Cortex-M0 and Cortex-M0+.
0x4001F000-0x4001FFFF (4KB)	System controller registers.	System controller registers.
0x40012000-0x4001EFFF	Unused.	Unused.
0x40011000-0x40011FFF (4KB)	AHB GPIO 1 I/O port GPIO[a].	AHB GPIO 1 I/O port GPIO[a].
0x40010000-0x40010FFF (4KB)	AHB GPIO 0 I/O port GPIO[a].	AHB GPIO 0 I/O port GPIO[a]
0x40000000-0x4000FFFF (64KB)	APB subsystem peripherals.	APB subsystem peripherals.
0x20010000-0x3FFFFFFF	Unused.	Unused.

Table 2-10 AHB memory map (continued)

Address	Without remap	With remap
0x20000000-0x2000FFFF (64KB)	RAM.	RAM.
0x01010000-0x1FFFFFFF	Unused.	Unused.
0x01000000-0x0100FFFF (64KB)	Optional boot loader memory. Actual size 4KB, access above 4KB wraps round.	Optional boot loader memory. Actual size 4KB, access above 4KB wraps round.
0x00010000-0x00FFFFFF	Unused.	Unused.
0x00000000-0x0000FFFF (64KB)	Program memory.	Alias of optional boot loader memory. Actual size 4KB, access above 4KB wraps round.

a. For use with the Cortex-M0+ only.

If you enable the boot loader, the system executes the following sequence after reset:

1. The processor exits from reset with the remap feature on by default. The processor fetches the initial Program Counter and stack pointer from boot loader memory. The reset vector points to the reset handler at the boot loader address 0x0100XXXX.
2. The processor executes the boot loader from boot loader address 0x0100XXXX. The remap feature is still on so the boot loader memory is also visible from the boot loader alias address. This is necessary if the processor is required to handle interrupts, because the vector table is at the start of the address space.
3. When the boot loader is ready to switch to program memory, it turns off the remap feature by writing to a register in the System Controller. See *System controller* on page 2-19. The program memory is then visible at the start of the program memory. After the boot loader switches the memory map, the processor must execute the DSB instruction and then the ISB instruction to ensure it uses the new memory map.
4. The boot loader loads the initial stack pointer value from the program memory, and branches to the reset handler that the reset vector specifies in the program memory.

When the linking stage creates the boot loader image, it must specify the memory location of the boot loader actual address, not the alias. The example software includes an example boot loader in the software/common/bootloader/bootloader.c file.

2.5.2 APB subsystem memory map

Table 2-11 describes the peripherals in the APB subsystem.

Table 2-11 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

Table 2-11 APB subsystem peripherals (continued)

Address	Item	Notes
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 *Arm® 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 2-20.

2.6.1 About the system controller

The example system contains a simple system controller that provides:

- Control of the memory remap feature.
- The ability to enable or disable the operation of the *Power Management Unit* (PMU), if available.
- 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 2-6 shows the example system controller.

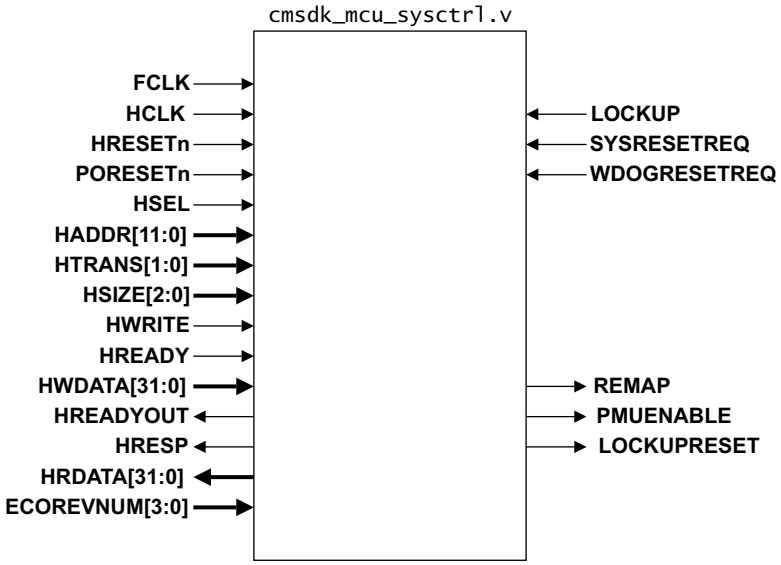


Figure 2-6 Example system controller

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

Table 2-12 Example system controller non-AHB signals (continued)

Signals	Descriptions
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 2-13 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: **1** Enable PMU. If not present, the value of this bit is ignored. **0** Disable PMU. 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.

Table 2-13 System controller programmers model (continued)

Address	Name	Type	Reset	Descriptions
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.
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 2-7 shows the interface of the example microcontroller.

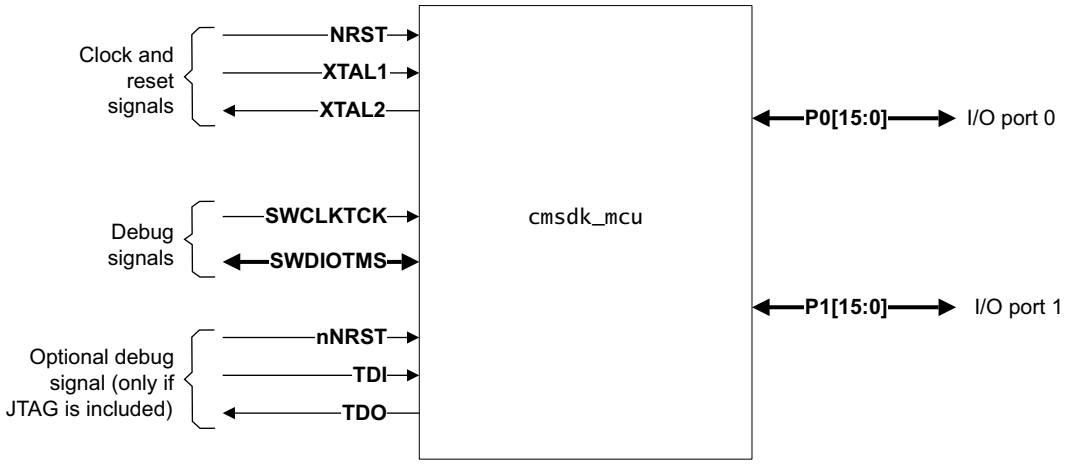


Figure 2-7 Example microcontroller interface

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]	Input	GPIO
P1[15:0]	Input	GPIO
nTRST	Input	JTAG reset, active LOW[a]
TDI	Input	JTAG data in[a]
SWDIOTMS	Input	Serial Wire Data or JTAG TMS
SWCLKTCK	Input	Serial Wire clock or JTAG clock
TDO	Output	JTAG data out[a]

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

Table 2-15 shows the alternate functions of the GPIO 1 ports that support pin multiplexing.

Table 2-15 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.

2.8.1 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	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	DMA done and DMA error
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 2-8 on page 2-25 shows an example circuit that performs this synchronization.

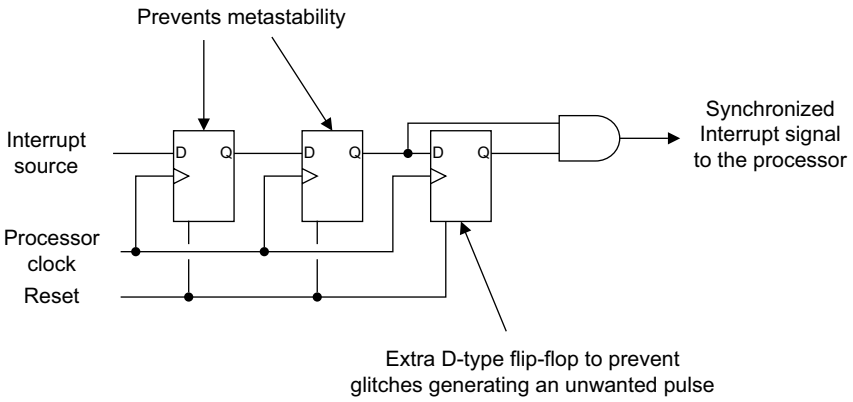


Figure 2-8 IRQ synchronizer

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 nonzero 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 and Cortex-M0+ have 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. In the example system, **RVEX** is connected to **dma_done** of the DMA-230 Micro DMA controller. This enables the processor to wake up from WFE sleep when the DMA process finishes.

2.9 AHB system ROM table

This module, `cmsdk_ahb_cs_rom_table`, is an example system level CoreSight ROM table. This enables a debugger to uniquely identify the Cortex-M0 or Cortex-M0+ based system, and enables debug components, such as the optional CoreSight MTB-M0+, to be discovered automatically.

The system ROM table is a 4K component, and is located at address `0xF0000000` in the memory map. This module includes parameters that you must modify with your own JEP106 manufacturer ID value and a part number that identifies your system. See the *Arm® CoreSight® Architecture Specification* for information about how to configure your ROM tables to ensure that they are correctly referenced.

The example system ROM table for the Cortex-M0 system has a single entry:

- A pointer to the Cortex-M0 ROM table.

The example system ROM table for the Cortex-M0+ system has two entries:

- A pointer to the Cortex-M0+ ROM table.
- A pointer to the optional CoreSight MTB-M0+.

The system ROM table is included in the Cortex-M0 and Cortex-M0+ systems. However by default it is not the first ROM table in Cortex-M0 based systems because the Cortex-M0 DAP points to the processor ROM table. You can modify the value of **baseaddr** in the Cortex-M0 integration level to make the Cortex-M0 DAP point to the system ROM table instead.

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.

The PMU is inside the microcontroller. It performs the clock gating and the wake-up of the system from deep-sleep.

Figure 2-9 shows the clock and reset operation of the example microcontroller.

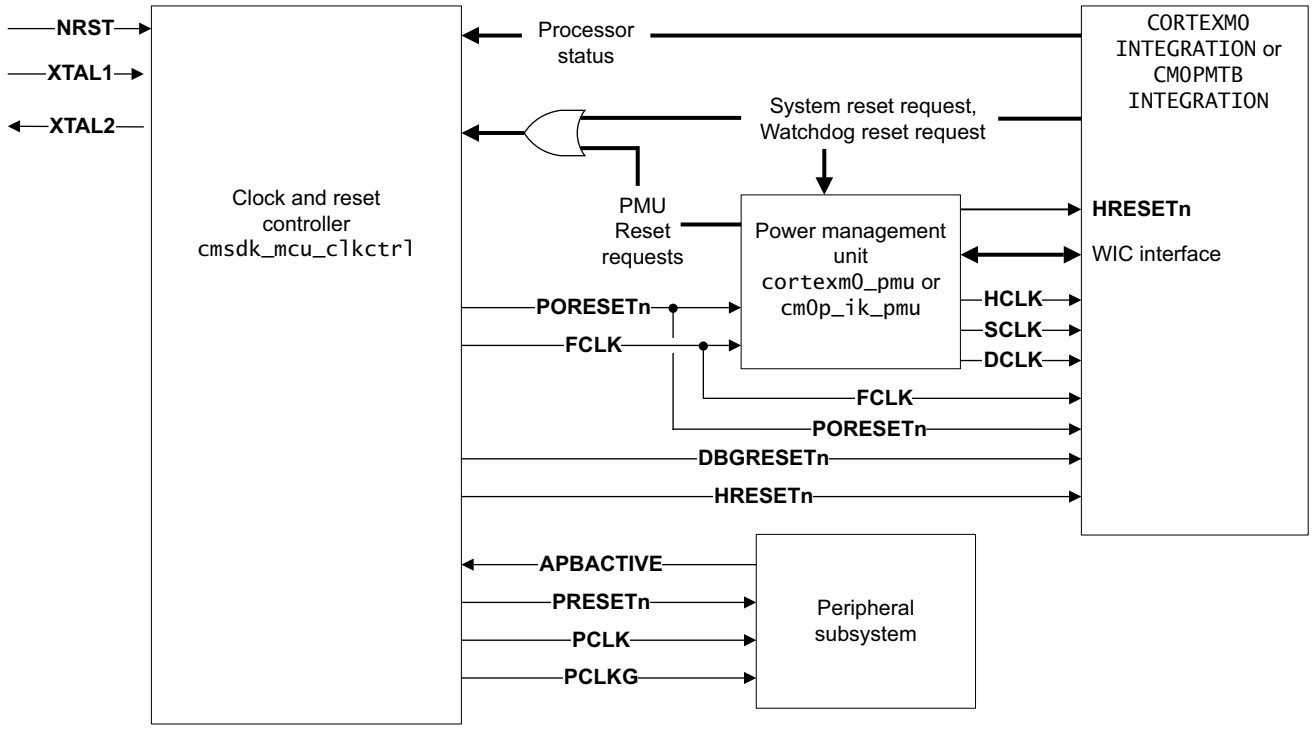


Figure 2-9 Example microcontroller clock and reset operation

The example only demonstrates a simple application scenario. Arm recommends that, for actual silicon projects, you modify the design of the PMU and clock controller 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 STCALIB register.

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 SysTick Calibration Value Register, SYST_CALIB in the *ARMv6-M Architecture Reference Manual* for more information.

2.12 Handling the *Engineering Change Order* (ECO) Revision Number in ID registers

The peripheral blocks in the Cortex-M0 and Cortex-M0+ System Design Kit provide a set of peripheral ID registers. These registers enable software to determine the identity of the peripheral and the revision of the design, so that you can perform software-based workaround solutions if a defect is detected in a peripheral.

One of the bit fields in the ID registers is connected to a 4-bit input called **ECOREVNUM**. This is the ECO Revision Number, and enables the revision information to be modified using metal fixes or ECO. To enable these modifications, you must do the following:

1. In the instantiation of the peripheral blocks in the example system Verilog files, you must connect each bit of the **ECOREVNUM** input to individual tie-off cells. This includes `cortex_m_mcu_system.v` as shown in Table 2-4 on page 2-8, and `apb_subsystem.v` which is described in the *Cortex-M System Design Kit Technical Reference Manual*.

   Arm recommends that the tie-off cells are instantiated with a separated level of hierarchy so that they can be located easily, and to enable the process of switching the design to a different technology to be made easier by just changing the tie-off cell files.

2. Ensure that the tie-off cells are not optimized away in each stage of the synthesis and layout. This is commonly achieved by setting the signal as `dont_touch`. If any stage of the implementation flow does not retain the `dont_touch` attribute, the revision value in the ID register might not be modifiable with metal fixes or ECO changes.

Note

Ensure that the synthesis tool does not merge multiple bits of the tie-off cells, or share the tie-off cell connection with other functional logic.

Figure 2-10 shows the recommended method of using separate hierarchy for tie-off cells.

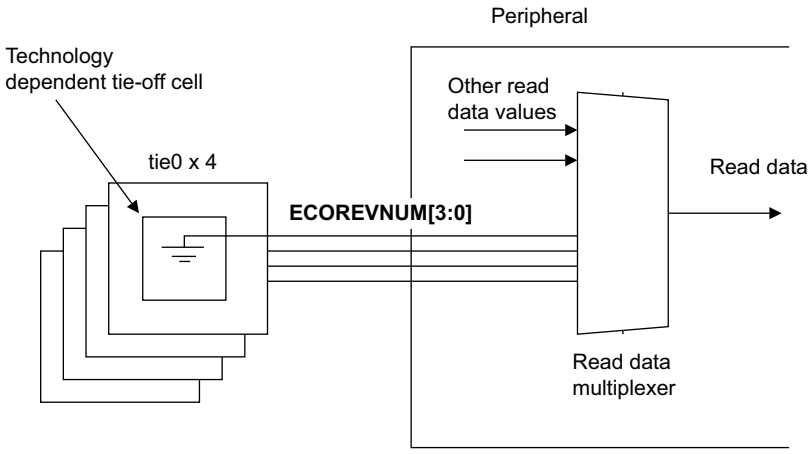


Figure 2-10 Separate hierarchy for tie-off cells

The example synthesis scripts contain the `dont_touch` attribute for the **ECOREVNUM** input for the Cortex-M0 or Cortex-M0+ processor, but not the peripheral blocks. If required, see the processor example using **ECOREVNUM** to implement the same arrangement for the peripheral blocks.

The behavior of the synthesis tool might change between different compile options and tool versions. This can have an impact to the resulting connection for **ECOREVNUM**. Arm recommends that you manually check the correct implementation of **ECOREVNUM** each time after synthesis or layout implementation is carried out.

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 debug tester to test for debug connectivity.
- A clock and reset generator.
- Text message capture by the UART.

Figure 3-1 shows a block diagram of the testbench.

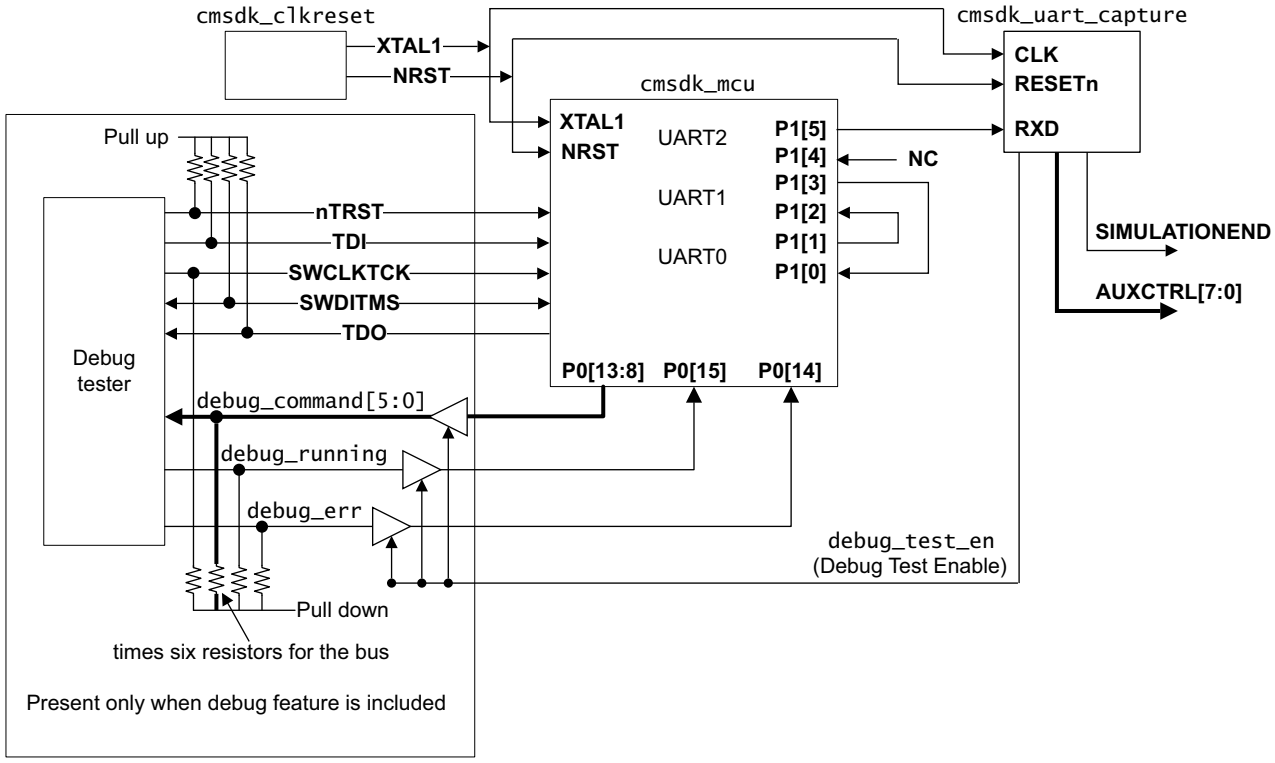


Figure 3-1 Testbench block diagram

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.

The UART capture module also supports the following features when you use the escape code:

- Control of the debug test enable signal **debug_test_en**.
- Provides an 8-bit auxiliary output **AUXCTRL**.

It implements the following escape codes:

ESC, 0x10, N Capture value of N to **AUXCTRL[7:0]**.

ESC, 0x11 Set **debug_test_en** HIGH.

ESC, 0x12 Set **debug_test_en** LOW.

The system uses the **debug_test_en** signal to control tristate buffers that enable or disable information passing between the example microcontroller and the debug tester.

3.3 Debug tester

The debug tester is a testbench component that tests debug and trace connectivity. Other tests do not require the debug tester. This section describes the following:

- *About the debug tester*.
- *Operation*.
- *Information passing* on page 3-6.
- *Debug tester program* on page 3-7.
- *Debug tester test function information* on page 3-8.

3.3.1 About the debug tester

The debug tester is a separate processor system that uses a Cortex-M0 or Cortex-M0+ processor. It is included only when you define the debug option `ARM_CMSDK_INCLUDE_DEBUG_TESTER` in the Verilog file `cmsdk_mcu_defs.v`. By default, the system disables the debug tester by setting the debug command input LOW at the start of the simulation. When the example microcontroller is required to carry out a debug test, it sends the escape code `ESC-0x11` to UART2 to enable the communication between the example microcontroller and the debug tester.

3.3.2 Operation

Table 3-1 describes the set of signals that the debug tester uses to communicate with the example microcontroller:

Table 3-1 Debug tester interface signals

Signal	Direction	Description
DBGCMD	Input from microcontroller	Debug command, 6-bit signal to indicate what task to perform
DBGRUNNING	Output to microcontroller	Debug tester acknowledges the debug command, or debug tester is running a debug task
DBGERROR	Output to microcontroller	Debug tester has encountered an error during the debug operation

Figure 3-2 on page 3-5 shows the handshaking sequence of the communication between the debug tester and the microcontroller.

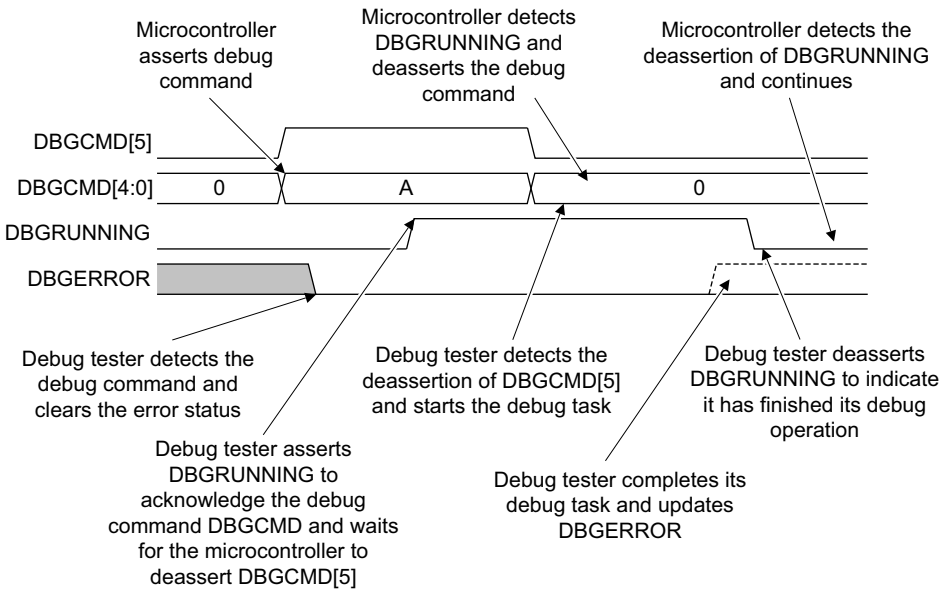


Figure 3-2 Debug command handshake sequence

Figure 3-3 on page 3-6 shows the program flow chart of the debug tester.

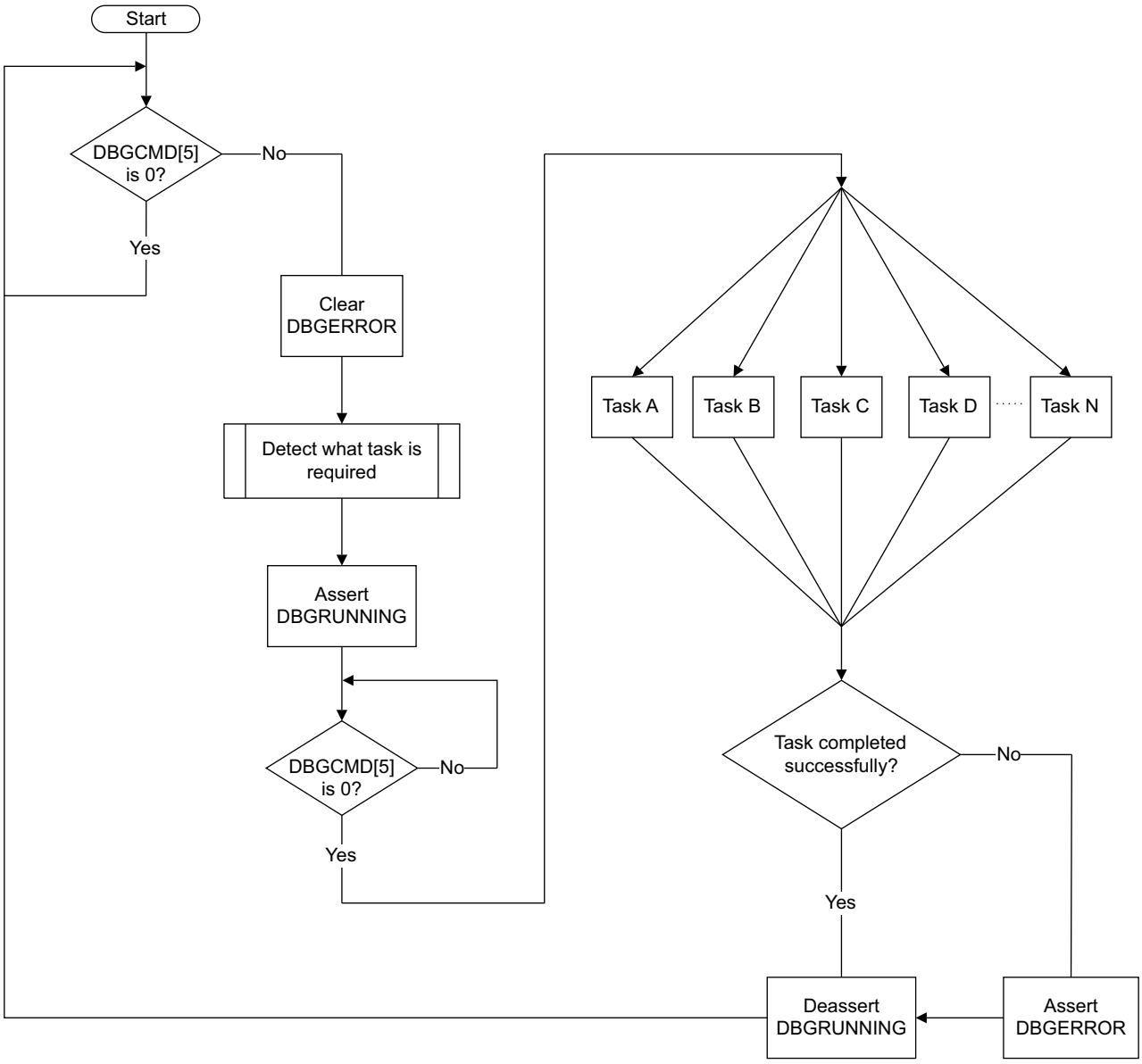


Figure 3-3 Program flow chart of debug tester

The debug tester also has its own test functions. See *Debug tester test function information* on page 3-8.

3.3.3 Information passing

The handshaking mechanism between the debug tester and the microcontroller can only pass information about the test start and test result. To enable the debug tester to transfer more information, it uses four words of SRAM at a memory address above the stack memory.

By allocating four words of RAM space above the top of stack memory, the debug tester locates the data variables by accessing the *Main Stack Pointer* (MSP) initial value, that is always at address `0x0`. The debug tester then executes the data transfer. Figure 3-4 on page 3-7 shows the memory space that the debug tester uses for its communication.

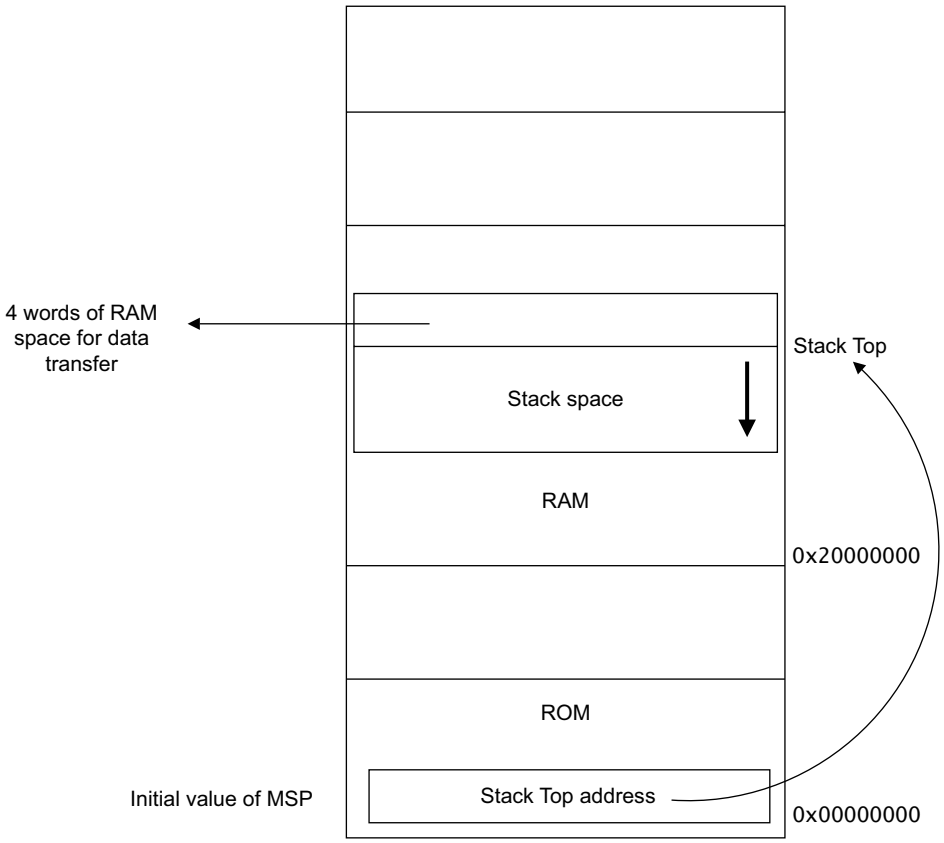


Figure 3-4 Memory space for debug tester communication

Note

If a simulation test uses the debug tester, the top of the stack memory must be at least four words lower than the ending address of the valid SRAM memory range.

3.3.4 Debug tester program

The debug tester is a processor system that has its own program code. The design kit includes a precompiled version of the debug tester program in the `software/debug_tester/` directory, either `debugtester_le.hex` for little endian and `debugtester_be.hex` for big endian. You can change the configuration of the processor, for example, because one of the following has changed:

- Expected value of an ID register.
- Expected number of comparators in breakpoint or watchpoint units.
- Endianness.

If you change the configuration of the processor, you might have to update the debug tester program. You can locate the source code of the debug tester program in the `software/debug_tester/` directory. This directory includes the files `debugtester.h` and debug `cmsdk_debugtester.h`, which contain generic values and peripheral definitions common across all Cortex-M series processors.

Additional header files can be found the `systems/cortex_m0_mcu/testcodes/generic` directory. This directory includes the file `config_id.h`, which stores most of the expected values.

The design kit includes the compile setup for Arm DS-5, GCC, and Keil MDK-ARM so you can regenerate the test program with your new configuration.

3.3.5 Debug tester test function information

Table 3-2 describes the test functions that the debug tester provides.

Table 3-2 Debug tester test functions

Function name	Input data	Return data	Return status	Description
`FnSetInterfaceJTAG` (**DBGCMD [4:0]** = 0)	None.	None.	FAIL if DAP was already on. PASS otherwise.	Set debug tester to use JTAG protocol.
`FnSetInterfaceSW` (**DBGCMD [4:0]** = 1)	None.	None.	FAIL if DAP was already on or DAP reported errors. PASS if DPIDR was successfully read from the DP.	Set debug tester to use Serial Wire protocol and read DPIDR from DAP.
`FnDAPPowerUp` (**DBGCMD [4:0]** = 2)	None.	None.	FAIL if DAP was already on or DAP reported errors. PASS when system and debug power domain power-up-requests have been acknowledged.	Power up the system and debug power domains.
`FnDAPPowerDown` (**DBGCMD [4:0]** = 3)	None.	None.	FAIL if DAP was already off or DAP reported errors. PASS when system and debug power domain power-down-requests have been acknowledged.	Power down the system and debug power domains.
`FnGetTAPID` (**DBGCMD [4:0]** = 4)	None.	`Memory[StackTop]` = JTAG TAPID	FAIL if DAP was already on or DAP reported errors. PASS when TAPID has been written to the MCU memory.	Read the DAP JTAG TAP ID and return using the reserved SRAM space above the stack.
`FnGetDPReg` (**DBGCMD [4:0]** = 5)	`Memory[StackTop]` = DP Register address.	`Memory[StackTop]` = DP Register value.	FAIL if DAP was already on or DAP reported errors. PASS when DP register value has been written to the MCU memory.	Read the value of a DP register.
`FnGetAPReg` (**DBGCMD [4:0]** = 6)	`Memory[StackTop]` = AP Register address.	`Memory[StackTop]` = AP Register value.	FAIL if DAP was already on or DAP reported errors. PASS when AP register value has been written to the MCU memory.	Read the value of an AP register.
`FnGetAPMem` (**DBGCMD [4:0]** = 7)	`Memory[StackTop]` = AP Memory address.	`Memory[StackTop]` = AP Memory value.	FAIL if DAP was already on or DAP reported errors. PASS when word-aligned AP Memory value has been written to the MCU memory.	Read a word of memory.

Table 3-2 Debug tester test functions (continued)

Function name	Input data	Return data	Return status	Description
`FnSetAPMem` (**DBGCMD [4:0]** = 8)	`Memory[StackTop]` = AP Memory address. `Memory[StackTop + 4]` = Data value.	None	FAIL if DAP was already on or DAP reported errors. PASS when data value has been written to the word-aligned location in the MCU memory.	Write a word of memory.
`FnGetAemCSIDs` (**DBGCMD [4:0]** = 9)	`Memory[StackTop]` = CoreSight Component Base Address.	`Memory[StackTop]` = Merged ID value ({CID3, CID2, CID1, CID0}). `Memory[StackTop+4]` = Merged ID value ({PID3, PID2, PID1, PID0}). `Memory[StackTop+8]` = Merged ID value ({PID7, PID6, PID5, PID4}).	FAIL if DAP was already on or DAP reported errors. PASS when CoreSight Component and Peripheral ID values have been written to the MCU memory.	Read the CoreSight Component ID0 to ID3 (address offset `0xFF0` to `0xFFC`) and Peripheral ID0 to ID7 of a peripheral (address offset `0xFD0` to `0xFEC`).
`FnConnectWakeUnhalt` (**DBGCMD [4:0]** = 10)	None.	`Memory[StackTop]` = 1.	FAIL if DAP was already on or DAP reported errors. PASS if MCU was seen sleeping and was successfully halted, the software flag written and core un-halted.	If the processor is sleeping, halt the processor and write to a memory location that contains a software flag variable, and then unhalt the processor.
`FnConnectCheckUnlockup` (**DBGCMD [4:0]** = 11)	`Memory[StackTop + 8]` = Non Faulting HardFault Handler (NOP, BX LR). `Memory[StackTop + 0xC]` = Fault generating HardFault Handler (`0xf123`, BX LR).	None.	FAIL if DAP was already on or DAP reported errors. PASS if MCU was seen in Lockup State and was successfully put back into normal running state.	If the processor is locked up, halt the processor, modify HardFault handler in SRAM, change Program Counter and then permit the processor to resume its operation.
`FnEnableHaltingDebug` (**DBGCMD [4:0]** = 12)	None.	None.	FAIL if DAP was already on or DAP reported errors. PASS if write to DHCSR.C_DEBUGEN was successful.	Enable debugging by setting the C_DEBUGEN bit in the DHCSR.
`FnDAPAccess` (**DBGCMD [4:0]** = 13)	None	`Memory[StackTop]` = 7: Bit 0 = word RW success. Bit 1 = halfword RW success. Bit 2 = byte RW success).	FAIL if DAP was already on or DAP reported errors. PASS if test was successful.	Perform simple read and write operations into the MCU memory at word, halfword, and byte sizes.

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 very quickly. The simulation environment includes software files and simulation setup makefiles.

The simulation environment supports the following Verilog simulators:

- Mentor ModelSim (6.0 or above).
- 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).
- Arm RealView Development Suite.
- 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.

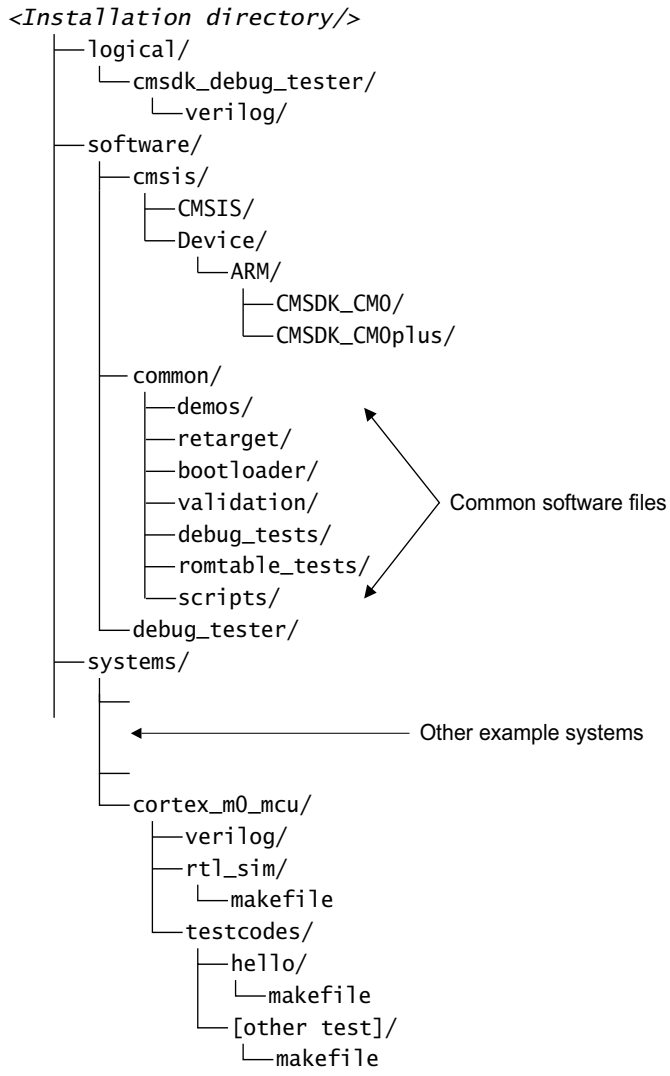


Figure 4-1 Directories for simulation

The Cortex-M0 and Cortex-M0+ System Design Kit contains example systems for the Arm Cortex-M0 and Cortex-M0+ processors. The Cortex-M System Design Kit contains more example systems. The files for the additional systems are located in sub-directories below the `systems` directory.

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 and CMSDK_CM0plus 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.
logical/cmsdk_debug_tester/verilog/	Design of the debug tester.
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 demonstation.
software/common/retarget	Support files to handle printing.
software/common/debug_tests	Debug test.
software/common/romtable_tests	Reads the system ROM table to determine what the system contains.
software/common/scripts	Linker scripts.
software/debug_tester	Design of the debug tester for the Cortex-M Series processor supported. This contains all necessary files to generate the .hex file software that the debug tester runs. This directory also includes a precompiled image, source code, and a makefile for the Arm DS-5 and Arm GCC.

4.3 Setting up the simulation environment

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

- *Installing the processor core*.
- *Modifying the rtl_sim/makefile*.
- *Modifying configuration files* on page 4-6.
- *Enabling assertions checking* on page 4-6.
- *Setting up tools* on page 4-7.

4.3.1 Installing the processor core

This design kit does not include the processor RTL. After you unpack the design kit deliverables, you must also unpack and install the processor RTL. The Verilog command files that are located in directory `systems/cortex_m0_mcu/verilog/` assume that the processor RTL files are stored in the `cores` directory. See Figure 1-1 on page 1-4. The directories are as follows:

- `cores/at510_cortexm0_r0p0_03rel2/logical/` for Cortex-M0. The path is specified in file `tbench_M0.vc`.
- `cores/at590_cortexm0p_r0p1/logical/` for Cortex-M0+. The path is specified in file `tbench_M0P.vc`. In addition, a number of components from the Cortex-M0+ integration kit are also required. The integration kit is expected to be located in `cores/at590_cortexm0p_r0p1/integration_kit`.

If you store the processor RTL in another location you must also update the corresponding Verilog command file.

You have the option to include the DMA-230 Micro DMA Controller in your simulation environment. The DMA-230 is not included in the design kit, and you must license it separately. The default location for the DMA-230 is in directory `logical/pl230_udma/logical/`. The Verilog command files specify the location.

4.3.2 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.
`CPU_PRODUCT`	The type of CPU core product. This can be `CORTEX_M0` or `CORTEX_M0PLUS`.
`SIMULATOR`	The Verilog simulator product. This can be one of the following: **mti** Mentor Graphics Questasim. **vcs** Synopsys VCS. **nc** Cadence IUS/Incisive.
`MTI_OPTIONS`	Simulator tool specific command line options for Mentor Graphics Questasim.

Table 4-2 Makefile variables (continued)

Variable	Descriptions
VCS_OPTIONS	Simulator tool specific command line options for Synopsys VCS.
NC_OPTIONS	Simulator tool specific command line options for Cadence IUS/Incisive.
BOOTLOADER	Name of boot loader software directory. This name must match the software directory name inside systems/cortex_m0_mcu/testcodes/ directory.
SW_MAKE_OPTIONS	Options that pass to the software makefile.

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 4-10 for example test programs.

4.3.3 Modifying configuration files

The systems/cortex_m0_mcu/verilog/cmsdk_mcu_defs.v file specifies most of the configurations of the example system.

For more information see *Preprocessing definitions* on page 2-11. If you do not have a DMA Controller installed, you must comment out the line `include ARM_CMSDK_INCLUDE_DMA.

If you want to change the configuration of the processors, for example the number of **IRQ** lines, you can edit the Verilog parameters in the systems/cortex_m0_mcu/verilog/tb_cmsdk_mcu.v file.

This propagates the settings to the processor instantiation. See *Verilog parameters for Cortex-M0* on page 2-13 and *Verilog parameters for Cortex-M0+* on page 2-14.

Note

Ensure that the relevant CMSIS header file for your chosen processor agrees with the configuration performed for the presence or not of the MPU and VTOR, as applicable. See systems/cortex_m0_mcu/rtl_sim/makefile for more information.

4.3.4 Enabling assertions checking

To enable assertions checking you must download the OVL Verilog library from Accellera, and add the OVL library path in the include and search paths of your simulator setup using the Verilog command file, tbench_M0.vc, or tbench_M0P.vc.

The Verilog command file also contains preprocessing definitions that control the inclusion of OVL assertions in the example system. Define:

- ARM_AHB_ASSERT_ON to include AHB-Lite protocol checkers and OVL assertions in AHB-Lite components.
- ARM_APB_ASSERT_ON to include APB protocol checkers and OVL assertions in APB components.
- ARM_IOP_ASSERT_ON to include Cortex-M0+ I/O Port protocol checkers.

4.3.5 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 4-10.

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.

If an error occurs at this time, it might be caused by an incorrect RTL path setting. Check the RTL path settings in the Verilog command file `tbench_M0.vc` or `tbench_M0P.vc` and the `CPU_PRODUCT` variable in the makefile located in the `rtl_sim` directory.

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.

`CPU_TYPE`	CPU type. The default setting is one of the following:
	`--cpu Cortex-M0`
	If the `TOOL_CHAIN` is `ds5`.
	`-mcpu=cortex-m0`
	If the `TOOL_CHAIN` is `gcc`.

`COMPILE_BIGEND`	Endianness. This can be set to one of the following: **0** Little-endian. This is the default value. **1** Big-endian.

Table 4-3 Makefile settings (continued)

Variable	Descriptions
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.
COMPILE_SMALLMUL	Use only for the DS-5 option. **0** Use single cycle multiplier. This is the default value. **1** Use the small multiplier that has a multiple clock cycle operation.
USER_DEFINE	A user defined C preprocessing macros. Set to -DCORTEX_M0 or -DCORTEX_M0P 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 additional 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 4-10.

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 4-11 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.

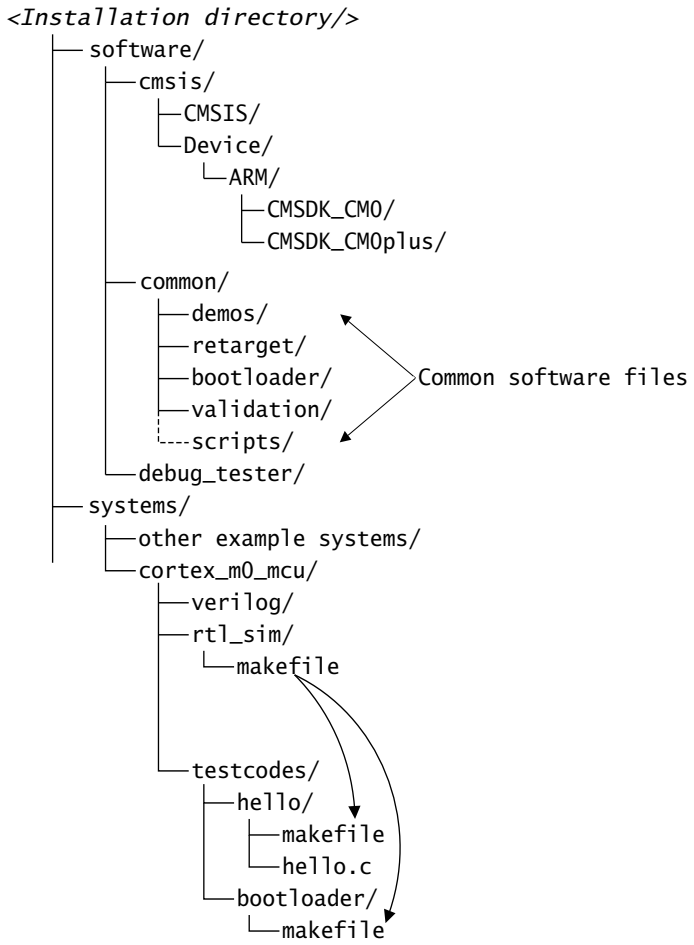


Figure 4-2 Interaction of makefiles

The Debug tester software is precompiled. You have to only compile this software if you change the processor configuration.

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:

```
#    Time: 0 ps  Iteration: 1  Instance: /tb_cmsdk_mcu/u_cmsdk_debug_tester/u_ram#
30160 ns UART: Hello world# 31580 ns UART: Test Ended# Break in NamedBeginStat
p_sim_end at ../verilog/cmsdk_uart_capture.v line 219# Stopped at
../verilog/cmsdk_uart_capture.v line 219# 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
```

Note

A warning appears when using Questasim with a Cortex-M0 based system because of the following modules being referenced but uncompiled:

- `cm0p_32to16_dnsize`.
- `cmsdk_ahb_to_flash16`.

You can ignore this warning.

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.
`bitband_tests`	Tests the bitband wrapper.
`debug_tests`[a]	Debug connectivity test. You can only use this test with the full version of the Cortex-M0 or Cortex-M0+, and if you configure the system to include the debug features.
`dma_tests`	Tests the DMA-230 connections. For more information, see `config_id.h` in the `generic` directory.
`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.
`romtable_tests`[b]	Checks that it is possible for a debugger to autodetect the Cortex-M0 or Cortex-M0+ processor.
`vtor_tests`[c]	Simple test for the *Vector Table Offset Register* (VTOR).
`mpu_tests`[c]	Sets up a number of MPU regions and demonstrates the generation of faults from the MPU.
`user_tests`[c]	Simple test for user and privilege modes in the Cortex-M0+ processor.
`mtb_tests`[c]	Simple test for the *Micro Trace Buffer* (MTB).

a. These tests use a `confg_id.h` file in the `cortex_m0_mcu/testcodes/generic` directory to configure the tests to run with the same setup as the processor.

b. This test prints a failure message by default on the Cortex-M0+ processor because the first ROM table is the system ROM table, which contains zero values by default. You must change the values in the system ROM table. See *AHB system ROM table* on page 2-26. This test passes on Cortex-M0 because the first ROM table the test finds is the processor ROM table.

c. Only applicable in Cortex-M0+. This test is skipped in Cortex-M0.

debug_tests is the only test that uses the debug tester.

Many of these tests are shared with multiple example systems that are only available in the full version of the design kit. Therefore, the source code for these tests is located in the software/common/ directory, see *Files and directory structure* on page 4-3. This directory also contains shared header files and example device driver files. See *Example header files and device driver files* on page 5-6.

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.

Note

The precompiled RTX OS image is built for a little-endian system, and therefore does not work if you choose a big-endian configuration.

The config_id.h header file in the testcodes/generic directory contains the defines for each of the available functions in the Cortex-M0 or Cortex-M0+ processor. You must update this file to reflect the configuration of the system so that the tests pass correctly. For example, if the MPU is enabled in the Cortex-M0+ processor, set EXPECTED_MPU to 8.

5.1.1 Generic header file for tests

In the same way that you can use the parameters in the RTL to configure the hardware, there is a header file for the tests, testcodes/generic/config_id.h, which includes defines that must match the hardware so the tests run correctly. Table 5-2 shows the defines.

Table 5-2 Defines for tests

Define	Description
EXPECTED_BE	Set to 1 for big endian and 0 for little endian.
EXPECTED_BKPT	Shows the number of breakpoints, 0,2, or 4.
EXPECTED_DBG	Set to 1 when the debug is enabled.
EXPECTED_NUMIRQ	Provides the number of interrupts, up to 32.
EXPECTED_SMUL	Set to 1 when the small multiplier is selected, otherwise the fast multiplier is present.
EXPECTED_SYST	Set to 1 when the SysTick extension is selected.
EXPECTED_WPT	Provides the number of watchpoints, 0, 1, or 2.
EXPECTED_JTAGnSW	Set 1 for JTAG and 0 for Serial Wire.
EXPECTED_IOP	Set to 1 when the I/O Port interface is present. Only applicable for Cortex-M0+.
EXPECTED_MPU	Set to the number of MPU regions, that is, 0 or 8. Only applicable for Cortex-M0+.
EXPECTED_USER	Set to 1 when user and privilege mode is available, otherwise only privilege mode is present. Only applicable for Cortex-M0+.
EXPECTED_VTOR	Set 1 when there is VTOR in the system. Only applicable for Cortex-M0+.

For the Cortex-M0+, you can select the MTB to perform instruction trace. Table 5-3 shows the defines.

Table 5-3 Defines for MTB tests

Define	Description
EXPECTED_MTB	Expected CoreSight MTB-M0+ configuration, 0-1
EXPECTED_MTB_BASEADDR	Expected CoreSight MTB-M0+ BASEADDR, 0-0xFFFFFFFF
EXPECTED_MTB_AWIDTH	Expected CoreSight MTB-M0+ address width, 5-32

Note

When the MTB is included, that is, EXPECTED_MTB = 1, the RAM that is used for the data accesses is shared with the MTB. If you set the address width to be smaller than the default, that is, EXPECTED_MTB_AWIDTH = 16, the addresses that some tests use to write data to will alias to lower addresses, therefore some tests do not work with the smaller memory.

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 and 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/` or `software/cmsis/Device/ARM/CMSDK_CM0plus/`.
- Device-specific startup codes, located in directory `cmsis/Device/ARM/CMSDK_CM0/Source/` or `cmsis/Device/ARM/CMSDK_CM0plus/Source/`.
- Device-specific example device drivers, located in directory `cmsis/Device/ARM/CMSDK_CM0/` or `cmsis/Device/ARM/CMSDK_CM0plus/`.

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 5-4 shows the generic Cortex-M0 and Cortex-M0+ processor support files.

Table 5-4 Generic Cortex-M0 and Cortex-M0+ processor support files

Filename	Descriptions
`core_cm0.h` `core_cm0plus.h`	CMSIS 3.20 compatible header file for processor peripheral registers definitions.
`core_cmInstr.h`	CMSIS 3.20 compatible header file for accessing special instructions.
`core_cmFunc.h`	CMSIS 3.20 compatible header file for accessing special registers.

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

Table 5-5 Device-specific header files

Filename	Descriptions
`CMSDK_CM0.h` `CMSDK_CM0plus.h`	CMSIS compatible device header file including register definitions
`system_CMSDK_CM0.h` or `system_CMSDK_CM0plus.h`	CMSIS compatible header file for system functions
`system_CMSDK_CM0.c` or `system_CMSDK_CM0plus.c`	CMSIS compatible program file for system functions

Table 5-6 shows the device-specific startup codes.

Table 5-6 Device-specific startup codes

Filename	Descriptions
cmsis/Device/ARM/CMSDK_CM0/Source/ARM/startup_CMSDK_CM0.s cmsis/Device/ARM/CMSDK_CM0plus/Source/ARM/startup_CMSDK_CM0plus.s	CMSIS compatible startup code for Arm DS-5 or Keil MDK
cmsis/Device/ARM/CMSDK_CM0/Source/GCC/startup_CMSDK_CM0.s cmsis/Device/ARM/CMSDK_CM0plus/Source/GCC/startup_CMSDK_CM0plus.s	CMSIS compatible startup code for Arm GCC

Table 5-7 shows the device-specific example device drivers.

Table 5-7 Device-specific example device drivers

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

You can use the config_id.h file in the testcodes/generic directory to configure the test switches to run differently, depending on whether USER or MPU is present.

To use these header files, you only have to include the device-specific header file CMSDK_CM0.h or CMSDK_CM0plus.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#ifdef CORTEX_M0PLUS#include
"CMSDK_CM0plus.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 or CORTEX_M0PLUS. 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 5-8 shows the files that are required for retargeting support.

Table 5-8 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 5-9 shows the UART functions.

Table 5-9 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.

5.5 Example device driver software

The design kit contains several driver functions for:

- The simple APB timer.
- The APB UART.
- The AHB GPIO.

The files `CMSDK_driver.c` and `CMSDK_driver.h` contain these functions.

5.5.1 UART driver functions

Table 5-10 to Table 5-21 on page 5-11 describe the UART driver functions.

Table 5-10 shows the initialize UART function.

Table 5-10 Initialize UART function

Description	Initialize UART	
Process	Initialize UART. Detect error status and return 1 if any error is detected.	
Function	`uint32_t CMSDK_uart_init(CMSDK_UART_TypeDef *CMSDK_UART, uint32_t divider, uint32_t_tx_en, uint32_t_rx_en, uint32_t tx_irq_en, uint32_t rx_irq_en, uint32_t tx_ovrirq_en, uint32_t rx_ovrirq_en,)`	
Return	**0**	Success.
	1	Error flag.

Table 5-11 shows the get UART RX buffer full status function.

Table 5-11 Get UART RX buffer full status function

Description	Get UART RX buffer full status	
Function	`uint32_t CMSDK_uart_GetRxBufferFull(CMSDK_UART_TypeDef *CMSDK_UART).`	
Return	**0**	Receive buffer empty.
	1	Receive buffer full.

Table 5-12 shows the get UART TX buffer full status function.

Table 5-12 Get UART TX buffer full status function

Description	Get UART TX buffer full status	
Function	`uint32_t CMSDK_uart_GetTxBufferFull(CMSDK_UART_TypeDef *CMSDK_UART).`	
Return	**0**	Transmit buffer empty.
	1	Transmit buffer full.

Table 5-13 shows the send a character to UART TX buffer function.

Table 5-13 Send a character to UART TX buffer function

Description	Send a character to UART TX buffer
Process	If TX buffer full, wait. Send a character to TX buffer.
Function	`void CMSDK_uart_SendChar(CMSDK_UART_TypeDef *CMSDK_UART, char txchar).`
Return	None.

Table 5-14 shows the receive a character from UART RX buffer function.

Table 5-14 Receive a character from UART RX buffer function

Description	Receive a character from UART RX buffer
Process	If RX buffer empty, wait. Receive a character from RX buffer.
Function	`char CMSDK_uart_ReceiveChar(CMSDK_UART_TypeDef *CMSDK_UART).`
Return	Received character.

Table 5-15 shows the get UART overrun status function.

Table 5-15 Get UART overrun status function

Description	Get UART overrun status	
Function	`uint32_t CMSDK_uart_GetOverrunStatus(CMSDK_UART_TypeDef *CMSDK_UART).`	
Return	**0**	No overrun.
	1	Transmit overrun.
	2	Receive overrun.
	3	Both transmit and receive overrun.

Table 5-16 shows the clear UART overrun status function.

Table 5-16 Clear UART overrun status function

Description	Clear UART overrun status	
Process	Clear overrun status. Read overrun status again and return new status.	
Function	`uint32_t CMSDK_uart_GetOverrunStatus(CMSDK_UART_TypeDef *CMSDK_UART).`	
Return	**0**	No overrun.
	1	Transmit overrun.
	2	Receive overrun.
	3	Both transmit and receive overrun.

Table 5-17 shows the get UART baud rate divider value function.

Table 5-17 Get UART baud rate divider value function

Description	Get UART baud rate divider value
Function	`uint32_t CMSDK_uart_GetBaudDivider(CMSDK_UART_TypeDef *CMSDK_UART).`
Return	Baud rate divider value.

Table 5-18 shows the get UART transmit IRQ status function.

Table 5-18 Get UART transmit IRQ status function

Description	Get UART transmit IRQ status	
Function	`uint32_t CMSDK_uart_GetTxIRQStatus(CMSDK_UART_TypeDef *CMSDK_UART).`	
Return	**0**	IRQ request inactive.
	1	IRQ request active.

Table 5-19 shows the get UART receive IRQ status function.

Table 5-19 Get UART receive IRQ status function

Description	Get UART receive IRQ status	
Function	`uint32_t CMSDK_uart_GetRxIRQStatus(CMSDK_UART_TypeDef *CMSDK_UART).`	
Return	**0**	IRQ request inactive.
	1	IRQ request active.

Table 5-20 shows the UART TX interrupt clear function.

Table 5-20 UART TX interrupt clear function

Description	UART TX interrupt clear
Function	`void CMSDK_uart_ClearTxIRQ(CMSDK_UART_TypeDef *CMSDK_UART).`
Return	None.

Table 5-21 shows the UART RX interrupt clear function.

Table 5-21 UART RX interrupt clear function

Description	UART RX interrupt clear
Function	`void CMSDK_uart_ClearRxIRQ(CMSDK_UART_TypeDef *CMSDK_UART).`
Return	None.

5.5.2 Timer driver functions

Table 5-22 on page 5-12 to Table 5-33 on page 5-14 describe the timer driver functions.

Table 5-22 shows the set timer for multi-shoot mode with internal clock function.

Table 5-22 Set Timer for multi-shoot mode with internal clock function

Description	Set Timer for multi-shoot mode with internal clock
Function	`void CMSDK_timer_Init_IntClock(CMSDK_TIMER_TypeDef *CMSDK_TIMER, uint32_t reload, uint32_t irq_en).`
Return	None.

Table 5-23 shows the set timer for multi-shoot mode with external clock function.

Table 5-23 Set Timer for multi-shoot mode with external clock function

Description	Set Timer for multi-shoot mode with external clock
Function	`void CMSDK_timer_Init_ExtClock(CMSDK_TIMER_TypeDef *CMSDK_TIMER, uint32_t reload, uint32_t irq_en).`
Return	None.

Table 5-24 shows the set timer for multi-shoot mode with external input as enable function.

Table 5-24 Set Timer for multi-shoot mode with external input as enable function

Description	Set Timer for multi-shoot mode with external input as enable
Function	`void CMSDK_timer_Init_ExtEnable(CMSDK_TIMER_TypeDef *CMSDK_TIMER, uint32_t reload, uint32_t irq_en).`
Return	None.

Table 5-25 shows the timer interrupt clear function.

Table 5-25 Timer interrupt clear function

Description	Timer interrupt clear
Function	`void CMSDK_timer_ClearIRQ(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	None.

Table 5-26 shows the get timer reload value function.

Table 5-26 Get Timer reload value function

Description	Get Timer reload value
Function	`uint32_t CMSDK_timer_GetReload(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	Reload register value.

Table 5-27 shows the timer reload value function.

Table 5-27 Set Timer reload value function

Description	Set Timer reload value
Function	`void CMSDK_timer_SetReload(CMSDK_TIMER_TypeDef *CMSDK_TIMER, uint_32 value).`
Return	None.

Table 5-28 shows the get timer current value function.

Table 5-28 Get Timer current value function

Description	Get Timer current value
Function	`uint32_t CMSDK_timer_GetValue(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	Timer current value.

Table 5-29 shows the set timer current value function.

Table 5-29 Set Timer current value function

Description	Set Timer current value
Function	`void CMSDK_timer_SetValue(CMSDK_TIMER_TypeDef *CMSDK_TIMER, uint_32 value).`
Return	None.

Table 5-30 shows the stop timer function.

Table 5-30 Stop Timer function

Description	Stop Timer
Process	Clear timer enable bit to 0.
Function	`void CMSDK_timer_StopTimer(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	None.

Table 5-31 shows the start timer function.

Table 5-31 Start Timer function

Description	Start Timer
Process	Set timer enable bit to 1.
Function	`void CMSDK_timer_StartTimer(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	None.

Table 5-32 shows the enable timer interrupt function.

Table 5-32 Enable Timer interrupt function

Description	Enable Timer interrupt
Process	Clear timer IRQ enable bit to 0.
Function	`void CMSDK_timer_EnableIRQ(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	None.

Table 5-33 shows the disable timer interrupt function.

Table 5-33 Disable Timer interrupt function

Description	Disable Timer interrupt
Process	Clear timer IRQ enable bit to 0.
Function	`void CMSDK_timer_DisableIRQ(CMSDK_TIMER_TypeDef *CMSDK_TIMER).`
Return	None.

5.5.3 GPIO driver functions

Table 5-34 to Table 5-47 on page 5-18 describe the GPIO driver functions.

Table 5-34 shows the set GPIO output enable function.

Table 5-34 Set GPIO output enable function

Description	Set GPIO output enable
Process	Read current Output Enable setting. Modify value (OR). Write back Output Enable.
Function	`void CMSDK_gpio_SetOutEnable(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 outenable).`
Return	None.

Table 5-35 shows the clear GPIO output enable function.

Table 5-35 Clear GPIO output enable function

Description	Clear GPIO output enable
Process	Read current Output Enable setting. Modify value (AND NOT). Write back Output Enable.
Function	`void CMSDK_gpio_ClrOutEnable(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 outenable).`
Return	None.

Table 5-36 shows the get GPIO output enable function.

Table 5-36 Get GPIO output enable function

Description	Get GPIO output enable
Process	Read current Output Enable setting.
Function	`uint_32 CMSDK_gpio_GetOutEnable(CMSDK_GPIO_TypeDef *CMSDK_GPIO).`
Return	Current settings of output enable.

Table 5-37 shows the set GPIO alternative function enable function.

Table 5-37 Set GPIO Alternate function enable function

Description	Set GPIO Alternate function enable
Process	Read current Alternate Function setting. Modify value (OR). Write back Output Enable.
Function	`void CMSDK_gpio_SetAltFunc(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 AltFunc).`
Return	None.

Table 5-38 shows the clear GPIO alternative function enable function.

Table 5-38 Clear GPIO Alternate function enable function

Description	Clear GPIO Alternate function enable
Process	Read current Alternate function setting. Modify value (AND NOT). Write back Alternate function setting.
Function	`void CMSDK_gpio_ClrAltFunc(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 AltFunc).`
Return	None.

Table 5-39 shows the get GPIO alternative function setting function.

Table 5-39 Get GPIO Alternate function setting function

Description	Get GPIO Alternate function setting
Process	Read current Alternate function setting.
Function	`uint_32 CMSDK_gpio_GetAltFunc(CMSDK_GPIO_TypeDef *CMSDK_GPIO).`
Return	Current settings of Alternate function.

Table 5-40 shows the clear GPIO interrupt function.

Table 5-40 Clear GPIO interrupt function

Description	Clear GPIO interrupt
Process	Shift 1 << Num to get bit pattern. Write to INTCLEAR.
Function	`uint_32 CMSDK_gpio_IntClear(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	Current settings of INTSTATUS.

Table 5-41 shows the enable GPIO interrupt function.

Table 5-41 Enable GPIO interrupt function

Description	Enable GPIO interrupt
Process	Shift 1 << Num to get bit pattern. Read the current value of INTEN. Modify value (OR). Write back to INTEN.
Function	`uint_32 CMSDK_gpio_SetIntEnable(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	Current settings of INTEN.

Table 5-42 shows the disable GPIO interrupt function.

Table 5-42 Disable GPIO interrupt function

Description	Disable GPIO interrupt
Process	Shift 1 << Num to get bit pattern. Read the current value of INTEN. Modify value (OR). Write back to INTEN.
Function	`uint_32 CMSDK_gpio_ClrIntEnable(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	Current settings of INTEN.

Table 5-43 shows the set up GPIO interrupt as high-level function.

Table 5-43 Set up GPIO interrupt as high-level function

Description	Set up GPIO interrupt as high level
Process	Shift 1 << Num to get bit pattern. Read current value of INTTYPE, INTPOL. Mask the bits to be changed (clear to 0). Modify the value (OR). Write to INTTYPE, INTPOL.
Function	`void CMSDK_gpio_SetIntHighLevel(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	None.

Table 5-44 shows the set up GPIO interrupt as rising edge function.

Table 5-44 Set up GPIO interrupt as rising edge function

Description	Set up GPIO interrupt as rising edge
Process	Shift 1 << Num to get bit pattern. Read current value of INTTYPE, INTPOL. Mask the bits to be changed (clear to 0). Modify the value (OR). Write to INTTYPE, INTPOL.
Function	`void CMSDK_gpio_SetIntRisingEdge(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	None.

Table 5-45 shows the set up GPIO interrupt as low-level function.

Table 5-45 Set up GPIO interrupt as low-level function

Description	Set up GPIO interrupt as low level
Process	Shift 1 << Num to get bit pattern. Read current value of INTTYPE, INTPOL. Mask the bits to be changed (clear to 0). Modify the value (OR). Write to INTTYPE, INTPOL.
Function	`void CMSDK_gpio_SetIntLowLevel(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	None.

Table 5-46 shows the set up GPIO interrupt as falling edge function.

Table 5-46 Set up GPIO interrupt as falling edge function

Description	Set up GPIO interrupt as falling edge
Process	Shift 1 << Num to get bit pattern. Read current value of INTTYPE, INTPOL. Mask the bits to be changed (clear to 0). Modify the value (OR). Write to INTTYPE, INTPOL.
Function	`void CMSDK_gpio_SetIntFallingEdge(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 Num).`
Return	None.

Table 5-47 shows the set up GPIO output value using the Masked access function.

Table 5-47 Set up GPIO output value using Masked access function

Description	Set up GPIO output value using Masked access
Process	Write to MASKLOWBYTE[mask & `0x00FF`] = value. Write to MASKHIGHBYTE[(mask & `0xFF00`) >> 8] = value [15:8].
Function	`void CMSDK_gpio_MaskedWrite(CMSDK_GPIO_TypeDef *CMSDK_GPIO, uint_32 value, uint_32 mask).`
Return	None.

5.5.4 Linker script for Arm GCC

The directory `software/common/scripts/` contains the linker script `cmsdk_cm0.ld` for Arm GCC. It contains symbols that are specific to the Arm GCC startup code. If you use another gcc toolchain, you must modify this linker script accordingly.

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.

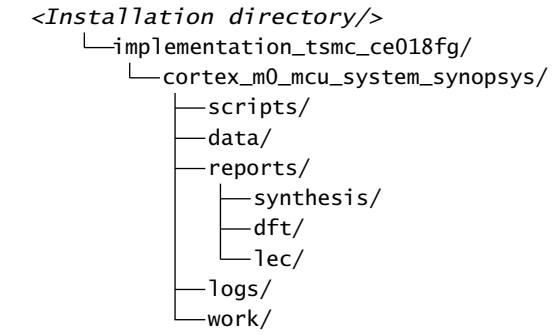


Figure 6-1 Implementation directories

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 that is generated from the synthesis process.
`reports/`	Location of the report files.
`logs/`	Location of the synthesis log file.
`work/`	Location of the temporary files that are 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. Configure the design. This includes the following:
   - Edit `cmsdk_mcu_defs.v` to match your system-level configuration requirements, including which processor to use.
   - Edit `cmsdk_mcu_system.v` to define the requirements, such as MPU and number of interrupts, using the Verilog parameter for the Cortex-M0 or Cortex-M0+ Integration level.
2. Select the appropriate library cells for clock gating and *Clock-Domain Crossing* (CDC) purposes. See the *Arm® Cortex®-M0 Integration and Implementation Manual* and the *Arm® Cortex®-M0+ Integration and Implementation Manual* for more information.
3. Customize the synthesis script for the targeted cell library.
4. Customize the synthesis script for the timing constraints and configuration.
5. Perform the synthesis.
6. Review the result.

Note

The following clock gating cells in files can only be used for behavioral simulation:

- `<your_cortex_m0_dir>/logical/models/cells/cm0_acg.v`
- `<your_cortex_m0_dir>/logical/models/cells/cm0_pmu_acg.v`
- `<your_cortex_m0p_dir>/logical/models/cells/cm0p_acg.v`
- `<your_cortex_m0p_dir>/logical/models/cells/cm0p_pmu_acg.v`

You must not use these files for synthesis, although they are used in the LEC as part of the reference system.

For synthesis, the directory `logical/models/clkgate/` contains the corresponding demonstration files for using the TSMC 0.18µm *Ultra Low Leakage* (ULL) process clock gating library cells.

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 6-8.
- *Running the makefile* on page 6-8.

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 and Cortex-M0+ processors are capable of running at higher frequencies, the extra bus infrastructure results in longer timing paths. You can increase the maximum frequency if you simplify the design configuration by doing one or more of the following:

- Remove the DMA option.
- Remove the bitband wrapper option.

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

If you have changed the design configuration, for example, by adding a DMA controller, you might also have to modify the timing constraints to meet the timing budget.

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.

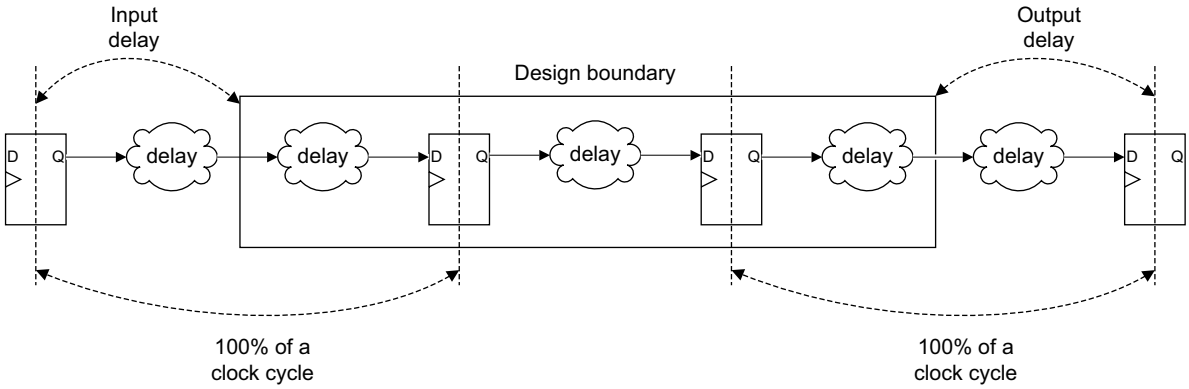


Figure 6-2 Input and output delays

Table 6-4 on page 6-6 and Table 6-5 on page 6-8 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%
Flash memory ROM AHB	**flash_hsel**	Output	Device select for flash/ROM		50%
	flash_hreadyout	Input	Flash/ROM ready	40%	
	flash_hrdata[31:0]	Input	Flash/ROM read data	60%	
	flash_hresp	Input	Flash/ROM response	60%	
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%	
Boot loader ROM AHB	**boot_hsel**	Output	Device select for boot ROM		50%
	boot_hreadyout	Input	Boot ROM ready	40%	
	boot_hrdata[[31:0]	Input	Boot ROM read data	60%	
	boot_hresp	Input	Boot ROM response	60%	
CPU status	**SLEEPING**	Output	Core is in sleep mode		50%
	SLEEPDEEP	Output	Core is in deep sleep mode		50%
	SYSRESETREQ	Output	System reset request		50%
	LOCKUP	Output	Core is locked up		50%
Power management	**PMUENABLE**	Output	PMU enable		50%
	GATEHCLK	Output	Safe to gate off **HCLK**		60%
	WAKEUP	Output	WIC request for wake up		60%
	APBACTIVE	Output	**PCLKG** enable/gating control		40%
	WICENREQ	Input	PMU request WIC feature	60%	
	WICENACK	Output	WIC acknowledge to PMU		60%
	CDBGPWRUPREQ	Output	Debug power up request		60%
	CDBGPWRUPACK	Input	Debug power up acknowledge	30%	
	SLEEPHOLDREQn	Input	Sleep extend request from PMU	60%	
	SLEEPHOLDACKn	Output	Sleep extend acknowledge		60%
Reset	**WDOGRESETREQ**	Output	Watchdog reset request		50%
	LOCKUPRESET	Output	Reset system when locked up		50%
	HRESETn	Input	AHB reset	60%	
	PORESETn	Input	Cold reset	60%	
	DBGRESETn	Input	Debug reset	60%	
	PRESETn	Input	APB reset	60%	

Table 6-4 Timing constraints for FCLK and HCLK (continued)

Interface	Signals	Type	Descriptions	Input delay	Output delay
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%
Misc	**PCLKEN**	Input	**PCLK** enable the for AHB to APB bridge, it is currently tied HIGH in the example clock controller	60%	
	RSTBYPASS	Input	Reset bypass input for Cortex-M0 only	20%	
	DFTRSTDISABLE	Input	Reset bypass input for Cortex-M0+ only	20%	
MTB RAM	**TSTART**	Input	External Trace Start pin	60%	
	TSTOP	Input	External Trace Stop pin	60%	
MTB SRAM	**SRAMBASEADDR[31:0]**	Input	MTB SRAM base address	60%	
	RAMHCLK	Output	MTB RAM clock, optionally gated		60%
	RAMRD[31:0]	Input	MTB connected RAM read-data	60%	
	RAMAD[AWIDTH-3:0]	Output	MTB connected RAM address		60%
	RAMWD[31:0]	Output	MTB connected RAM write-data		60%
	RAMCS	Output	MTB connected RAM chip select		60%
	RAMWE[3:0]	Output	MTB RAM byte lane write enables		60%

Table 6-5 shows the timing constraints for the **SWCLKTCK** domain interface signals.

Table 6-5 Timing constraints for SWCLKTCK domain interface signals

Interface	Signals	Type	Descriptions	Input delay	Output delay
Debug	**SWDITMS**	Input	Data In (SW) / TMS (JTAG)	60%	
	TDI	Input	Test data in (JTAG)	60%	
	SWDO	Output	Serial Wire Data Out (SW)		60%
	SWDOEN	Output	Data Output enable (SW)		60%
	nTDOEN	Output	Data Output Enable (JTAG)		60%
	TDO	Output	Data Output (JTAG)		60%
	nTRST	Input	Test reset (JTAG)	60%	

6.4.3 Combinatorial paths

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

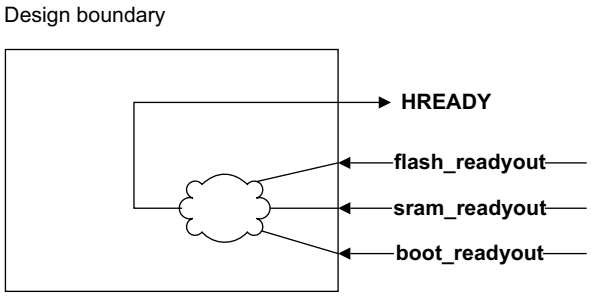


Figure 6-3 HREADY path

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` Performs *Logical Equivalence Checking* (LEC) between the Verilog RTL and the synthesized netlist.

`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
Debug tester

This appendix describes the debug components in the design kit. It contains the following sections:

A.1 About the debug tester

The debug tester is a testbench component that tests the connectivity of the debug feature. It is a Cortex-M0 or Cortex-M0+ based system that controls the Cortex-M0 or Cortex-M0+ DAP in the example microcontroller system. It uses the JTAG or Serial-Wire protocol through a simple I/O interface.

Figure A-1 shows the debug tester.

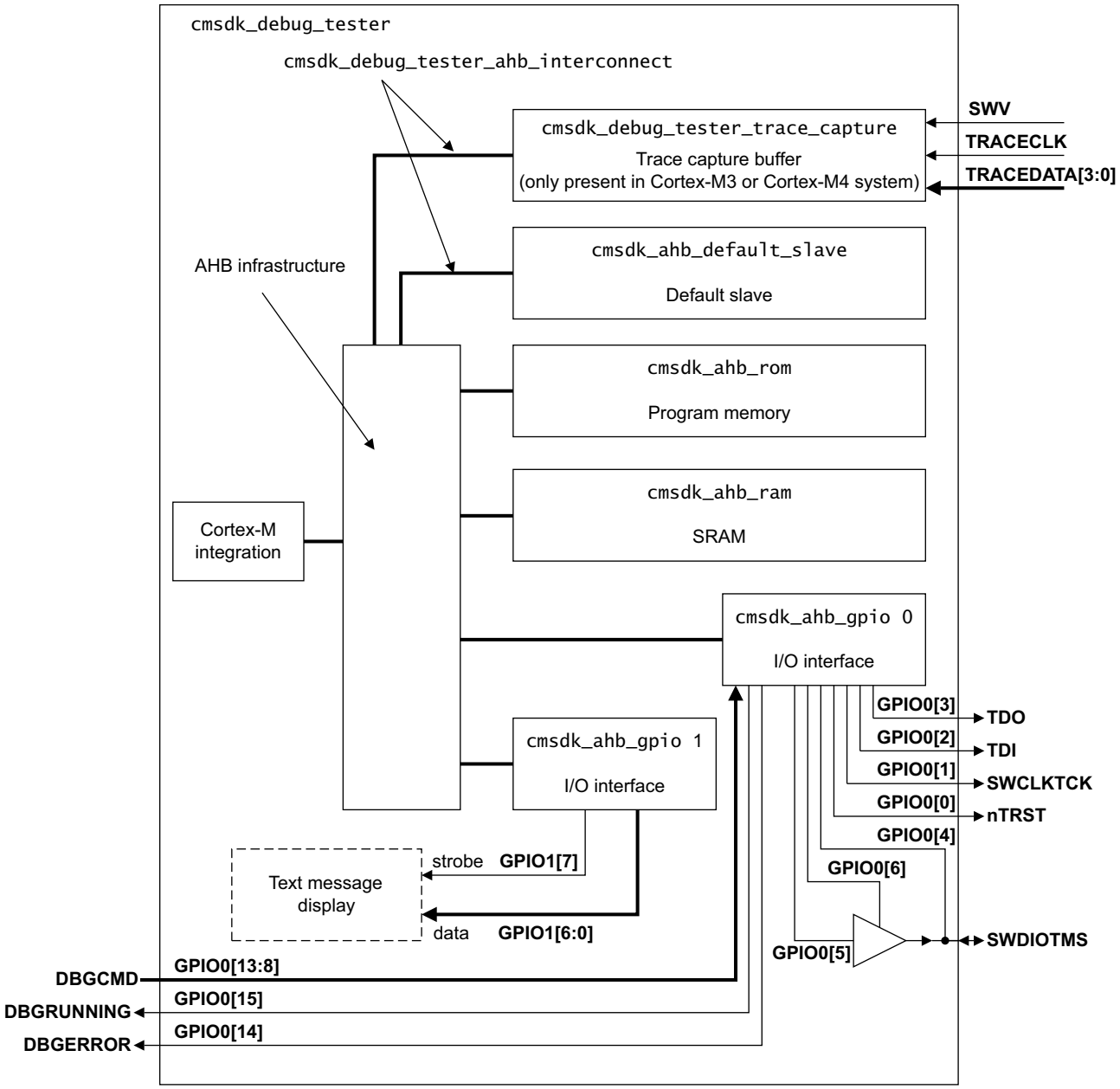


Figure A-1 Debug tester

A.2 Memory map

Table A-1 shows the memory map address space for the debug tester.

Table A-1 Debug tester memory map

Address	Descriptions
`0x00000000` to `0x0000FFFF`, size 64KB	ROM
`0x20000000` to `0x20003FFF`, size 16KB	SRAM
`0x40000000` to `0x40000FFF`, size 4KB	GPIO0
`0x40001000` to `0x40001FFF`, size 4KB	GPIO1
Other addresses except the Cortex-M0 or Cortex-M0+ internal System Control Space	Default slave

A.3 Debug tester software

The debug tester software is located in `software/debug_tester`. This directory includes pre-compiled hex files for the debug tester processor for both little-endian and big-endian configurations of the example microcontroller:

- `debugtester_le.hex`.
- `debugtester_be.hex`.

If you configure your example microcontroller to support big-endian, you must also ensure that the debug tester loads the `debugtester_be.hex` image.

These precompiled images are built for the Cortex-M0 processor, and so will execute correctly on whichever Cortex-M processor you use in the debug tester.

The design kit includes the source code of the debug tester software, a makefile configured for Arm DS-5 and Arm GCC, and a project file for Keil MDK. Normally, you do not have to recompile the software for the debug tester. However if you want to recompile the software image, follow these steps:

1. Edit the makefile to modify the compile configuration if required, for example, to target a particular Cortex-M processor, or to switch to a different toolchain.
2. Do one of the following:
   - Run `make` if you are using Arm DS-5 or Arm GCC.
   - Open the project file and compile the project if you are using Keil MDK-ARM.

     Note

     You must open the correct project file for the processor and endian configuration chosen. For example, open `debugtester_cm0_be.uvproj` if you are using a Cortex-M0 processor configured for big-endian.

Table A-2 shows the debug tester source files.

Table A-2 Debug tester source files

File	Description
`cmsdk_debugtester.h`	CMSIS header file which describes the debug tester.
`debugtester.c`	Main program of the debug tester.
`debugtester.h`	Main program header file. Edit this file if you need to enable message printing from the debug tester.
`debugtester_functions.h`	This file defines the functions that the debug tester supports. It is not used by the debug tester code itself, but is included by tests running on the example microcontroller that use the debug tester functions.
`retarget_cmsdk_debugtester.c`	`Printf` retargeting function. Enables C `printf()` calls to use the debug tester character output device.
`system_cmsdk_debugtester.c`	Empty `SystemInit()` function. Included for CMSIS compatibility.
`system_cmsdk_debugtester.h`	Header file for `system_cmsdk_debugtester.c`.

The compilation process also requires the processor support header files to be in the directory `software/cmsis/CMSIS/Include/`.

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location	Affects
First release	-	-

Table B-2 Differences between Issue A and Issue B

Change	Location	Affects
After an engineering review, confidentiality of document changed from Non-Confidential to Confidential	Whole document	All revisions
References changed from Cortex-M0 to reflect *Arm® Cortex®-M System Design Kit Technical Reference Manual*	*About the Cortex-M0 and Cortex-M0+ System Design Kit* on page 1-2 Table 2-7 on page 2-11 *APB subsystem memory map* on page 2-17	All revisions
Description for `cmsdk_ahb_gpio` item updated	Table 2-1 on page 2-3	All revisions
Note added to Preprocessing macro `ARM_CMSDK_SLOWSPEED_PCLK` in table	Table 2-7 on page 2-11	All revisions
Address `0x40020000-0xFFFFFFFF` Without remap wording updated	Table 2-10 on page 2-16	All revisions
File location amended in last paragraph of AHB memory map section to: `software/common/bootloader/bootloader.c`	*AHB memory map* on page 2-16	All revisions

Table B-2 Differences between Issue A and Issue B (continued)

Change	Location	Affects
Signal amended to **WDOGRESETREQ** in Example system controller figure	Figure 2-6 on page 2-19	All revisions
Example microcontroller clock and reset operation figure updated	Figure 2-9 on page 2-27	All revisions
Separate hierarchy for tie-off figure updated	Figure 2-10 on page 2-29	All revisions
Testbench block diagram figure updated	Figure 3-1 on page 3-2	All revisions
Paragraph updated to: Before the simulation stops, 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.	*UART text output capturing and escape code* on page 3-3	All revisions
Function description updated to: Read the value from an address in the memory	Table 3-20 on page 3-12	All revisions
Return data updated to reflect *None*	Table 3-22 on page 3-13	All revisions
Input data description second line updated to: Memory[StackTop + `0xC`] = Fault generating HardFault Handler (`0xf123`, `BX LR`)	Table 3-23 on page 3-13	All revisions
Directory name amended to: `software/cmsis/CM0/`	Table 4-1 on page 4-4	All revisions
Demonstration software and functional tests directories added to Installation directory table: • `software/common/demos` • `software/common/validation` • `software/common/bootloader` • `software/common/dhry` • `software/common/retarget` • `software/common/scripts`	Table 4-1 on page 4-4	All revisions
Sections removed on makefile in `rtl_sim`, combined with simulation stage because function no longer available	*Compile the test code* on page 4-8	All revisions
Additional example software test `bitband_tests` added	Table 5-1 on page 5-2	All revisions
`cd mytest` command added to second item in itemized list	*Creating a new test* on page 5-5	All revisions
Note updated	*Implementation flow* on page 6-4	All revisions
Description for `ortex_m0_mcu_system_tech.tcl` script name updated	Table 6-3 on page 6-4	All revisions
Paragraph added: In the case where the design configuration is changed, for example, by adding a DMA controller, you also might have to modify the timing constraints to meet the timing budget.	*Maximum frequency* on page 6-5	All revisions

Table B-3 Differences between issue B and issue C

Change	Location	Affects
Updated references to Cortex-M0 to include Cortex-M0+	Throughout document	r1p0
All module names prefixed with `cmsdk_`	Throughout document	r1p0
Changed *RealView Development Suite* (RVDS) to *Development Suite* (DS-5) and CodeSourcery g++ to Arm GCC	Throughout document	r1p0

Table B-3 Differences between issue B and issue C (continued)

Change	Location	Affects
Updated directory structure to show details of `software/` directory	Figure 1-1 on page 1-4, Figure 4-1 on page 4-3, Table 4-1 on page 4-4, and Figure 4-2 on page 4-11	r1p0
Added information on big-endian support with Arm GCC	*Big-endian support with Arm GCC* on page 1-6	r1p0
Added Cortex-M0+ components	Figure 2-1 on page 2-2 and Table 2-1 on page 2-3	r1p0
Updated interface diagrams for Cortex-M0	Figure 2-2 on page 2-4 and Figure 2-4 on page 2-6	r1p0
Added interface diagrams for Cortex-M0+	Figure 2-3 on page 2-5 and Figure 2-5 on page 2-7	r1p0
Updated Verilog file names, parameters, and definitions	Table 2-4 on page 2-8 to Table 2-7 on page 2-11	r1p0
Added file locations for Cortex-M0+	*Processor file locations* on page 2-10	r1p0
Added Tarmac description	*Tarmac* on page 2-10	r1p0
Added Verilog preprocessing definitions for Cortex-M0+	Table 2-7 on page 2-11	r1p0
Added Verilog parameters for Cortex-M0+	*Verilog parameters for Cortex-M0+* on page 2-14	r1p0
Added instructions on changing CPU type to Cortex-M0+	*Changing the processor type* on page 2-15	r1p0
Added Cortex-M0+ components to AHB memory map	Table 2-10 on page 2-16	r1p0
Added MTB functions to GPIO alternate function table	Table 2-15 on page 2-23	r1p0
Updated interrupt assignments to include I/O port GPIO	Table 2-16 on page 2-24	r1p0
Added AHB system ROM table	*AHB system ROM table* on page 2-26	r1p0
Consolidated debug tester test functions into a single table	Table 3-2 on page 3-8	r1p0
Updated installation directory	Table 4-1 on page 4-4	r1p0
Added installation instructions for Cortex-M0+	*Installing the processor core* on page 4-5	r1p0
Updated makefile descriptions	*Compile the test code* on page 4-8	r1p0
Added tests for Cortex-M0+ components	Table 5-1 on page 5-2	r1p0
Added defines for tests	*Generic header file for tests* on page 5-3	r1p0
Added MTB RAM and MTB SRAM signal timing constraints	Table 6-4 on page 6-6	r1p0
Replaced running synthesis script description with new makefile description	*Running the makefile* on page 6-8	r1p0
Updated debug tester block diagram	Figure A-1 on page A-2	r1p0
Updated debug tester memory map	*Debug tester memory map* on page A-3	r1p0
Updated debug tester software description	*Debug tester software* on page A-4	r1p0
Deleted GPIO peripheral description	Appendix A *Debug tester*	r1p0

Table B-4 Differences between issue C and issue D

Change	Location	Affects
Addresses with remap `0xF0220000-0xFFFFFFFF` and `0x40020000-0xEFFFFFFF` changed to unused.	*AHB memory map* on page 2-16	All revisions