ARM® Cortex®-M3 DesignStart™ Eval

Revision: r0p0

RTL and Testbench User Guide

ARM 100894_0000_00_en

ARM® Cortex®-M3 DesignStart™ Eval

RTL and Testbench User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	14 June 2017	Non-Confidential	First release for r0p0

Product Status

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

Web Address

http://www.arm.com

Contents

ARM® Cortex®-M3 DesignStart™ Eval RTL and Testbench User Guide

Preface

This preface introduces the *ARM® Cortex®-M3 DesignStart™ Eval RTL and Testbench User Guide*.

It contains the following:

About this book

This book describes the information required for system design and RTL simulation using Cortex®-M3 DesignStart™ Eval.

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

r*m* Identifies the major revision of the product, for example, r1.

p*n* Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

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

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces Cortex-M3 DesignStart Eval, its features, and its documentation structure.

Chapter 2 Design flow options
This chapter describes the design flow options for Cortex-M3 DesignStart Eval.

Chapter 3 Technical overview
This chapter gives an overview of the structure and main components of the example system in Cortex-M3 DesignStart Eval.

Chapter 4 Functional description
This chapter describes the memory maps, I/O pins, and TRNG registers in Cortex-M3 DesignStart Eval.

Chapter 5 Testbench
This chapter describes the components included with the testbench in Cortex-M3 DesignStart Eval.

Chapter 6 Simulation and integration tests
This chapter describes the integration tests and how to run the simulation.

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

Glossary

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

See the *ARM® Glossary* for more information.

Typographic conventions

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.

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

`monospace italic`
Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`
Denotes language keywords when used outside example code.

`<and>`
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 following figure 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.

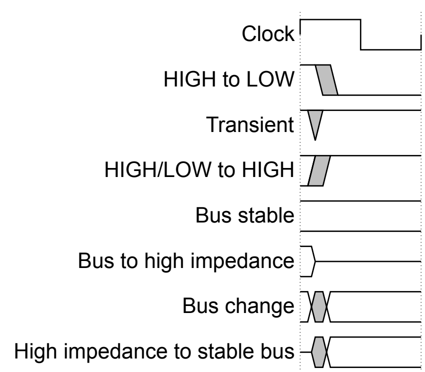

Figure 1 Key to timing diagram conventions

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

ARM publications

- Cortex®-M3 DesignStart™ Eval publications:
  - — *ARM® Cortex®-M3 DesignStart™ Eval FPGA User Guide* (ARM 100896).
  - — *ARM® Cortex®-M3 DesignStart™ Eval RTL and FPGA Quick Start Guide* (ARM 100895).
  - — *ARM® Cortex®-M3 DesignStart™ Eval Customization Guide* (ARM 100897).
- Other ARM publications:
  - — *ARM® Cortex®-M System Design Kit Technical Reference Manual* (ARM DDI0479).
  - — *ARM® TrustZone® TRNG True Random Number Generator Technical Reference Manual* (ARM 1009676).
  - — *ARM® PrimeCell™ Real Time Clock (PL031) Technical Reference Manual* (ARM DDI 0224).
  - — *ARM® PrimeCell® Synchronous Serial Port (PL022) Technical Reference Manual* (ARM DDI 0194).
  - — *ARM® Versatile™ Express Cortex®-M Prototyping System (V2M-MPS2 and V2M-MPS2+) Technical Reference Manual* (ARM 100112).
  - — *Application Note AN531 uSDCARD SPI Adapter for the Cortex-M Prototyping System (MPS2+)* (ARM DAI 0531).
  - — *Application Note AN502 Adapter for Arduino for the Cortex-M Prototyping System (MPS2 and MPS2+)* (ARM DAI 0502).
  - — *ARM® AMBA® 3 AHB-Lite Protocol Specification (v1.0)* (ARM IHI 0033).
  - — *ARM® Architecture Reference Manual ARMv7, for ARMv7-M architecture profile* (ARM DDI0403).
  - — *ARM® Cortex®-M3 Technical Reference Manual* (ARM 100165).
  - — *ARM® Cortex®-M3 Devices Generic User Guide* (ARM DUI0552).

Other publications

None.

Feedback

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 *ARM Cortex-M3 DesignStart Eval RTL and Testbench User Guide*.
- The number ARM 100894_0000_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter introduces Cortex-M3 DesignStart Eval, its features, and its documentation structure.

It contains the following sections:

1.1 About Cortex®-M3 DesignStart™ Eval

Cortex-M3 DesignStart Eval provides developers an easy way to develop and simulate SoC designs based on the ARM Cortex-M3 processor. It allows a system designer to design and test on a simulator and then proceed with hardware prototyping using an FPGA.

The Cortex-M3 DesignStart Eval package is aimed at developers who are new to ARM or have limited soft IP system design experience. The package includes the following:

- *1.1.1 RTL* on page 1-12.
- *1.1.2 Execution Testbench* on page 1-13.
- *1.1.3 FPGA Evaluation Flow* on page 1-13.

Cortex-M3 DesignStart Eval provides an easy entry into the ARM ecosystem, rather than a complete solution for all Cortex-M processor design scenarios.

The hardware ecosystem in Cortex-M3 DesignStart Eval is built around the CoreLink™ SSE-050 Subsystem and includes the use of the *Cortex-M System Design Kit* (CMSDK) standard library of *Advanced High-performance Bus* (AHB) and *Advanced Peripheral Bus* (APB) components. For more information on the CMSDK, see the *ARM® Cortex®-M System Design Kit Technical Reference Manual*.

The software ecosystem in Cortex-M3 DesignStart Eval uses the ARM *Cortex Microcontroller Software Interface Standard* (CMSIS) software standard library.

The use of CMSDK and CMSIS, coupled with a reprogrammable FPGA, allows for a fast turnaround and prototyping of Cortex-M3 processor-based hardware and software.

Cortex-M3 DesignStart Eval does not support the implementation of the Cortex-M3 processor into silicon. Any implementation of the Cortex-M3 processor into silicon requires you to obtain Cortex-M3 DesignStart Pro, or take a full Cortex-M3 processor license from ARM.

A Cortex-M3 DesignStart Pro license offers the following:

- The Cortex-M3 processor.
- The SDK-100 *System Design Kit* (SDK), which includes:
  - The CoreLink SSE-050 Subsystem.
  - The CMSDK components.
  - A *Real Time Clock* (RTC).
  - A stand-alone *True Random Number Generation* (TRNG).

An *Embedded Trace Macrocell* (ETM) is not included in Cortex-M3 DesignStart Pro, and requires a separate license.

If you are working on ASIC implementation, then ARM recommends that you license Cortex-M3 DesignStart Pro as early as possible.

1.1.1 RTL

The RTL in Cortex-M3 DesignStart Eval includes the components and peripherals that are required to implement a complete example system in an FPGA.

The example system is intended to provide a reference starting point for a typical IoT endpoint application and is a supported ARM mbed™ platform when implemented on the ARM *Versatile Express Cortex-M Prototyping System* (V2M-MPS2+) platform.

The Cortex-M3 DesignStart Eval RTL provides an example system that includes:

- A Cortex-M3 processor in a fixed configuration (obfuscated but synthesizable).
- A modified CoreLink SSE-050 subsystem supporting a single Cortex-M3 processor with support for debug and trace.
- A memory subsystem supporting *Execute In Place* (XIP). The MPS2+ platform preloads a code file at powerup.
- Two timers for Operating System use (privileged access only).

- Peripherals for:
  - — Application use, including Timers, UART, Watchdog, *Real Time Clock* (RTC), *True Random Number Generator* (TRNG).
  - — MPS2+ platform, including Color LCD, Audio, and Ethernet.
  - — Arduino Shield expansion using the adapter for the Arduino board.
- SPI interface supporting application persistent storage on microSD card.
- Reusable ARM *Advanced Microcontroller Bus Architecture* (AMBA) SoC interconnect components for system level development.

You must not modify the obfuscated Cortex-M3 processor (`cortexm3ds_logic.v`).

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

- `fpga_top.v`
- `m3ds_user_partition.v`
- `m3ds_peripherals_wrapper.v`

1.1.2 Execution Testbench

The Execution Testbench in Cortex-M3 DesignStart Eval is an RTL package that allows system design and simulation with a suitable Verilog simulator.

The Cortex-M3 DesignStart Eval Execution Testbench includes:

- A simulation model of the processor that includes register visibility and instruction execution tracing.
- Memory models that match the FPGA target.
- ARM CoreSight™ debug test engine that is preconfigured for a single fixed debug and trace implementation.
- Integration tests for memories and internal peripherals.

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

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

- `tb_fpga_shield.v`
- `cmsdk_uart_capture_ard.v`

1.1.3 FPGA Evaluation Flow

The Cortex-M3 DesignStart Eval FPGA Evaluation Flow allows developers to build an image file of the simulation system that can be used with the ARM *Versatile Express Cortex-M Prototyping System* (V2M-MPS2+). The FPGA image can be customized to the user system requirements.

The Cortex-M3 DesignStart Eval FPGA Evaluation Flow requires the purchase of the MPS2+ FPGA platform.

The MPS2+ FPGA platform includes a *Motherboard Configuration Controller* (MCC) on the baseboard, which provides the following features that are necessary to emulate an ARM mbed compliant system:

- Target application code. The target has no flash memory. The SRAM is instead initialized at powerup by the MCC using information stored on the configuration microSD card.
- DAPLink implementing CMSIS-DAP over USB for debug access.
- UART access is provided by a serial connector (and included serial to USB cable).
- *Real Time Clock* (RTC) initialization from baseboard processor on powerup.

For more information on how to use the MPS2+ FPGA platform, see the *ARM® Versatile™ Express Cortex®-M Prototyping System (V2M-MPS2 and V2M-MPS2+) Technical Reference Manual*.

You must not redistribute any FPGA bit files or other representations of the design that are produced from Cortex-M3 DesignStart Eval.

1.2 Using the documentation

There are several documents that are provided with Cortex-M3 DesignStart Eval.

Scope of this document

The *ARM® Cortex®-M3 DesignStart™ Eval RTL and Testbench User Guide* is the main document for system design and RTL simulation using Cortex-M3 DesignStart Eval.

Other documents

The following table shows the documents that relate to the design flow processes for Cortex-M3 DesignStart Eval:

Table 1-1 Other Cortex-M3 DesignStart Eval documents

Document name	Purpose
ARM® Cortex®-M3 DesignStart™ Eval FPGA User Guide	Describes how to build an FPGA image and evaluate software running on the *Versatile Express Cortex-M Prototyping System* (V2M-MPS2+).
ARM® Cortex®-M3 DesignStart™ Eval RTL and FPGA Quick Start Guide	Describes how to run basic tests using an RTL simulator and the MPS2+ FPGA platform. **Note** This is a procedural user-level document that gives a complete example of a working system. This document is highly recommended for users who do not have previous experience of ARM products.
ARM® Cortex®-M3 DesignStart™ Eval Customization Guide	Describes the high-level steps to integrate your own peripherals, and make other modifications to the Cortex-M3 DesignStart Eval system.

For more information about:

- Programming the Cortex-M3 processor, see the *ARM® Cortex®-M3 Technical Reference Manual*.
- Software development on a Cortex-M3 device, see the *ARM® Cortex®-M3 Devices Generic User Guide*. This is a generic device user-level reference document.
- The ARM architecture that the Cortex-M3 processor complies with, and the instruction set and exception model it uses, see the *ARM® Architecture Reference Manual ARMv7, for ARMv7-M architecture profile*.
- The AHB-Lite master interface that the Cortex-M3 processor implements, see the *ARM® AMBA® 3 AHB-Lite Protocol Specification (v1.0)*.
- Peripherals and interconnect components, see the *ARM® Cortex®-M System Design Kit Technical Reference Manual*.
- The *Real Time Clock* (RTC), see the *ARM® PrimeCell™ Real Time Clock (PL031) Technical Reference Manual*.
- The *Serial Peripheral Interface* (SPI), see the *ARM® PrimeCell® Synchronous Serial Port (PL022) Technical Reference Manual*.
- The MPS2+ FPGA platform, see the *ARM® Versatile™ Express Cortex®-M Prototyping System (V2M-MPS2 and V2M-MPS2+) Technical Reference Manual*.
- The *True Random Number Generator* (TRNG), see the *ARM® TrustZone® TRNG True Random Number Generator Technical Reference Manual*.

1.3 Directory structure

The following diagram shows the main directories of the Cortex-M3 DesignStart Eval:

```
<install_directory>/
├──docs/
├──cortexm3_model/
├──cmsdk/
├──m3designstart/
│   ├──fpga/
│   ├──logical/
│   │   └──testbench/
│   │       ├──execution_tb/
│   │       └──testcodes/
│   └──software/
├──m3designstart_iot/
├──rtc_pl031/
├──smm/
├──shared/
├──trng/
└──boards/
    └──Recovery/
```

Figure 1-1 Main directories for Cortex-M3 DesignStart Eval

The following table describes the contents of each top-level directory:

Table 1-2 Directory descriptions

Directory	Description
`docs/`	Contains documentation for Cortex-M3 DesignStart Eval.
`cortexm3_model/`	Contains the Cycle Model view of the Cortex-M3 processor integration level.
`cmsdk/`	Contains the RTL for: • ARM *Cortex-M System Design Kit* (CMSDK) components. Some CMSDK components are used in the example system in Cortex-M3 DesignStart Eval.
`m3designstart/`	Contains the following: • Cortex-M3 DesignStart Eval example system. • Testbench in `testbench/execution_tb/`. • Integration tests in `testbench/testcodes/`. • ARM *Cortex Microcontroller Software Interface Standard* (CMSIS) support files for the Cortex-M3 DesignStart Eval. • Scripts for building an FPGA image.
`m3designstart_iot/`	Cortex-M3 DesignStart Eval version of ARM CoreLink SSE-050 subsystem.
`rtc_pl031/`	Real Time Clock peripheral.
`smm/`	Peripherals and support code for the MPS2+ FPGA platform.
`shared/`	Contains IP-XACT bus definitions.

Table 1-2 Directory descriptions (continued)

Directory	Description
`trng/`	Stand-alone True Random Number Generator.
`boards/Recovery`	Contains the files that are required to be loaded onto the microSD card of the MPS2+ FPGA platform, in order to program and run the prebuilt FPGA image and software.

1.4 Limitations

You should not use the processor technology or the supporting deliverables as an indicator of what is received under a full license of the ARM Cortex-M3 processor.

Example system

Although the system that forms the basis for the design is built from components intended for an ASIC implementation, you are required to consider various conditions when planning to migrate from the example system here to a fully optimized ASIC implementation.

The subsystem provides some support for fine-grained power management, but does not implement any actual power control features. Typically, an ASIC would implement several different clock and power domains aimed at providing good performance without having an undue impact on standby power drain. Some critical guarantees are required for the timing of power rails and control signals, particularly in devices that implement embedded flash memory.

Obfuscated RTL

The obfuscated RTL view gives acceptable results when implemented in the FPGA, but does not provide a good reference for place-and-route prototyping.
There are no standard cell libraries included with Cortex-M3 DesignStart Eval.

Peripherals

The peripheral set that is provided for the example FPGA system is limited compared with the full set, which may be required for a small ASIC.

Integration tests

The integration tests included with Cortex-M3 DesignStart Eval can be used as a starting point for a full test suite, but they are not exhaustive and will need to be extended as part of the work to design a full ASIC.

ROM table

The CoreSight ROM table is part of the obfuscated `CORTEXM3INTEGRATIONDS` level, and Cycle Model. This has a fixed device identifier, which indicates that this is an example system supporting the Cortex-M3 processor from ARM. In a production device, the identification registers indicate the company that makes the device, and their own part number.

Chapter 2
Design flow options

This chapter describes the design flow options for Cortex-M3 DesignStart Eval.

It contains the following section:

2.1 Potential development routes

There are a range of options to use the Cortex-M3 DesignStart Eval package for your own design flows, with a combination of RTL, FPGA, or system modeling tools.

Simulation using the Cortex-M3 DesignStart Eval product is a likely first step in a wide range of possible design flows that leads you to develop your own products based on ARM technology. The various stages in the design flows require that you license other EDA tools, and have access to suitable compute resources. You may also need to license additional IPs to complete the process.

Cortex-M3 DesignStart Eval can be used with either an RTL simulator, or an FPGA platform, and a development toolchain. A limited term license for the ARM Keil® *Microcontroller Development Kit* (MDK) is included with DesignStart Eval. You can use this to evaluate the flows and perform low-level prototyping or modeling.

Cortex-M3 DesignStart Eval must not be used to manufacture devices.

You can also extend your evaluation of Cortex-M3 DesignStart Eval environment by using existing off-the-shelf standard parts and modules to extrapolate from the results you obtain in simulation, or on FPGA.

The model of the processor provided with Cortex-M3 DesignStart Eval is built using the ARM Cycle Model technology. These models can also be used for system level modeling and software evaluation. Fully featured Cycle Models can be licensed from ARM.

Cortex-M3 DesignStart Pro is a fast-track license option to access the full Cortex-M3 processor and SDK-100 deliverables to develop your own SoC design. This allows detailed SoC *Power, Performance and Area* (PPA) investigations and enables manufacture of devices. If you already have a good understanding of the design flow and the product you intend to develop, then Cortex-M3 DesignStart Pro may be a more appropriate starting point compared to the Cortex-M3 DesignStart Eval product.

The key use cases of the various development options are shown in the following table:

Table 2-1 Potential development routes

Use cases	Development options
Familiarization with ARM IP and flows	• Simulation in Cortex-M3 DesignStart Eval. • FPGA Evaluation Flow in Cortex-M3 DesignStart Eval.
Development cycle planning and familiarization	• Simulation in Cortex-M3 DesignStart Eval. • FPGA Evaluation Flow in Cortex-M3 DesignStart Eval.
Proof of concept demonstrator	• FPGA Evaluation Flow in Cortex-M3 DesignStart Eval. • Full suite of ARM Cycle Model.
Peripheral and accelerator prototyping	• Simulation in Cortex-M3 DesignStart Eval. • FPGA Evaluation Flow in Cortex-M3 DesignStart Eval.
System modeling	• Simulation in Cortex-M3 DesignStart Eval. • FPGA Evaluation Flow in Cortex-M3 DesignStart Eval. • Full suite of ARM Cycle Model.
System and software performance analysis	• Simulation in Cortex-M3 DesignStart Eval. • FPGA Evaluation Flow in Cortex-M3 DesignStart Eval. • Full suite of ARM Cycle Model.
SoC PPA analysis	• Cortex-M3 DesignStart Pro.
Power optimizations	• Extend evaluation of Cortex-M3 DesignStart Eval using existing off-the-shelf standard parts or modules. • Cortex-M3 DesignStart Pro.

Table 2-1 Potential development routes (continued)

Use cases	Development options
Software development	• FPGA Evaluation Flow in Cortex-M3 DesignStart Eval. • Extend evaluation of Cortex-M3 DesignStart Eval using existing off-the-shelf standard parts or modules. • Full suite of ARM Cycle Model.
SoC implementation and device manufacture	• Cortex-M3 DesignStart Pro.

Chapter 3
Technical overview

This chapter gives an overview of the structure and main components of the example system in Cortex-M3 DesignStart Eval.

It contains the following sections:

3.1 Example system

Cortex-M3 DesignStart Eval provides an example system, which includes all the components and peripherals that you require to implement a functioning mbed OS endpoint device.

The example system is designed to be implemented on the ARM *Versatile Express Cortex-M Prototyping System* (V2M-MPS2+), and comes with a full simulation environment.

The example system is built around the Cortex-M3 processor and the CoreLink SSE-050 Subsystem. In addition to the standard peripherals provided by the MPS2+ board, the example system in Cortex-M3 DesignStart Eval provides the following peripherals:

- Two timers that are dedicated for mbed OS usage.
- Timers, UART, Watchdog, *Real Time Clock* (RTC), and *True Random Number Generator* (TRNG) for application use.
- SPI interface for microSD card, using the microSD card SPI adapter board.

The following diagram shows an overview of the different hierarchies in the example system:

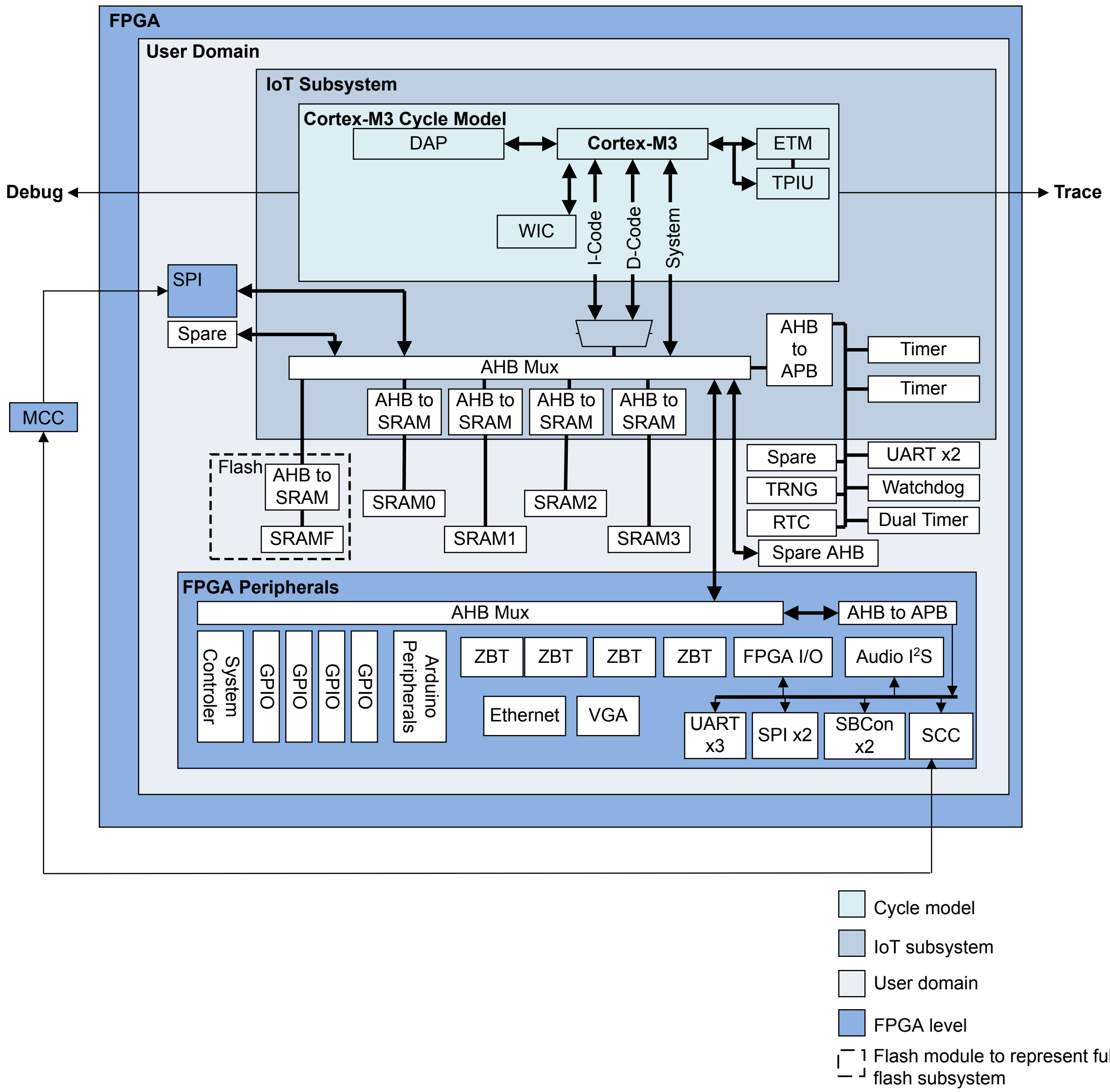

Figure 3-1 Cortex-M3 DesignStart Eval example system

Related references

3.2 Processor on page 3-24.
3.3 IoT subsystem on page 3-26.
3.4 FPGA peripherals and Arduino shield support on page 3-29.

3.2 Processor

The processor in ARM Cortex-M3 DesignStart Eval is a fixed configuration of the Cortex-M3 processor. This enables easy evaluation access to the Cortex-M3 processor technology without the flexibility to configure the design, which is included in the full product.

The processor in Cortex-M3 DesignStart Eval is delivered in two alternative forms, which are the obfuscated RTL, and a Cycle Model.

Obfuscated RTL

The obfuscated RTL is used to rebuild an FPGA image of the system when a modification is done on the example system. Only a limited set of internal registers are exposed for debug purposes.

The obfuscated RTL of the processor is preconfigured, and it is a synthesizable Verilog version of the full Cortex-M3 processor. It is not intended for a production SoC, and does not show optimum performance if used for ASIC implementation evaluations.

Cycle Model

The Cycle Model of the processor includes visibility of the internal processor architectural registers, for simulation and debug purposes. The model is linked with your simulator during compilation. The model also generates a Tarmac log, which is a textual trace output file that contains all the instructions executed, and register and memory transactions.

This section contains the following subsections:
- *3.2.1 Processor configuration* on page 3-24.
- *3.2.2 Processor debug features* on page 3-25.

3.2.1 Processor configuration

The processor obfuscated RTL and Cycle Model in Cortex-M3 DesignStart Eval are configured to the following parameter values:

Table 3-1 Processor configuration parameter and values

Parameter	Value	Description
`MPU_PRESENT`	1	*Memory Protection Unit* (MPU) is present. A tie-off pin allows this parameter to be hidden from software for emulating the performance of a design with no MPU.
`NUM_IRQ`	64	64 interrupts are supported. Interrupt 45-63 are free. Other interrupts have default connections. **Note** The full Cortex-M3 processor can support up to 240 interrupts.
`LVL_WIDTH`	3	Eight levels of interrupt priority are supported (3 bits of priority).
`TRACE_LVL`	3	The following trace components are present: • *Embedded Trace Macrocell* (ETM). • *Instrumentation Trace Macrocell* (ITM). • *Data Watchpoint and Trace* (DWT) unit. • *Trace Port Interface Unit* (TPIU). • *AMBA AHB Trace Macrocell* (HTM) interface. This can be used in FPGA for visibility of data accesses. If you want to use the HTM to generate data trace, then you are required to license it from ARM.

Table 3-1 Processor configuration parameter and values (continued)

Parameter	Value	Description
DEBUG_LVL	3	Full debug functionality is supported, including data matching for watchpoint generation.
JTAG_PRESENT	1	Both JTAG and SWD are supported.
CLKGATE_PRESENT	0	Architectural clock gating is not supported, which results in better FPGA implementation.
RESET_ALL_REGS	0	Not all registers have a defined reset value. Only architecturally reset registers are explicitly reset.
OBSERVATION	0	The observation port is not present. The internal state of the processor is not observable.
WIC_PRESENT	1	The *Wake-up Interrupt Controller* (WIC) is present.
WIC_LINES	67	The WIC is sensitive to all interrupt events.
BB_PRESENT	1	The bit-banding feature is implemented. Bit-banding enables every individual bit in the bit-banding region to be directly accessible for a word-aligned address. For more information, see the *ARM® Cortex®-M3 Technical Reference Manual*.
CONST_AHB_CTRL	1	AHB-Lite compliance is ensured. This is recommended for best compatibility with peripherals. For more information, see the *ARM® AMBA® 3 AHB-Lite Protocol Specification (v1.0)*.

Note

The processor in Cortex-M3 DesignStart Pro can be configured without restriction.

3.2.2 Processor debug features

The processor obfuscated RTL and Cycle Model implement an example of the Cortex-M3 processor integration level with preintegrated debug and trace components, and is configured for single core operation only.

The debug features include:

- *Serial Wire-JTAG Debug Access Port* (SW-JTAG DAP), which is similar to standard Cortex-M3 implementation. This provides the external interface to the debug hardware on the FPGA board.
- *AHB-Debug Access Port* (AHB-DAP), which provides access to processor registers and system memory.
- *Data Watchpoint and Trace* (DWT) unit. This provides automated tracing of processor events.
- *Instrumentation Trace Macrocell* (ITM). This allows limited and low overhead debug messaging, which is initialized by software.
- Instruction-only *Embedded Trace Macrocell* (ETM). This allows for non-intrusive, full cycle, and accurate instruction trace.
- A 4-pin Cortex-M processor optimized trace port supporting Single Wire Output protocol.

To use the 4-pin ETM trace mode of the *Trace Port Interface Unit* (TPIU), dedicated trace capture is required. You must connect to the trace port to observe the trace generated by the ETM. To implement an ETM in silicon, you are required to license it separately from ARM.

Since the debug configuration is fixed, only a simple connectivity test is provided for the Serial Wire configuration.

3.3 IoT subsystem

Cortex-M3 DesignStart Eval includes an IoT compute subsystem that extends the processor with the following:

- Bus interconnect.
- SRAM controller.
- Timer peripherals that are required by the ARM mbed software.
- Expansion capability for embedded flash controller.
- Radio product expansion interface (ARM Cordio® radio IP or similar communications interface).

This section contains the following subsections:

- *3.3.1 Subsystem features* on page 3-26.
- *3.3.2 Subsystem components* on page 3-26.
- *3.3.3 ARM Cordio radio integration* on page 3-27.
- *3.3.4 Example peripheral integration* on page 3-28.

3.3.1 Subsystem features

The features of the IoT subsystem in Cortex-M3 DesignStart Eval include:

- Instantiation of the Cortex-M3 processor with debug and trace.
- Two *Advanced Peripheral Bus* (APB) timers.
- A multi-layer AMBA AHB-Lite interconnect.
  - — Two AHB-Lite slave expansion ports with access to the full peripheral and memory range.
  - — Two AHB-Lite master expansion ports (one mapped to a 64KB region).
  - — 14 APB4 master expansion ports (4KB address space each).
- A memory system consisting of:
  - — AHB-Lite master expansion and two APB4 master expansion ports dedicated for connection to a flash interface (on chip or external, and optionally with cache).
  - — Static memory arranged in four banks of 32KB.

The IoT subsystem used for the example system in Cortex-M3 DesignStart Eval is a simplified version of CoreLink SSE-050. You can replace the subsystem with the full version of CoreLink SSE-050 for production. You can also create alternative subsystem designs that suit the needs of your application using the *Cortex-M System Design Kit* (CMSDK) components, which are included in Cortex-M3 DesignStart Eval. For more information on the CMSDK, see the *ARM® Cortex®-M System Design Kit Technical Reference Manual*.

3.3.2 Subsystem components

The following diagram gives an overview of the IoT subsystem in Cortex-M3 DesignStart Eval:

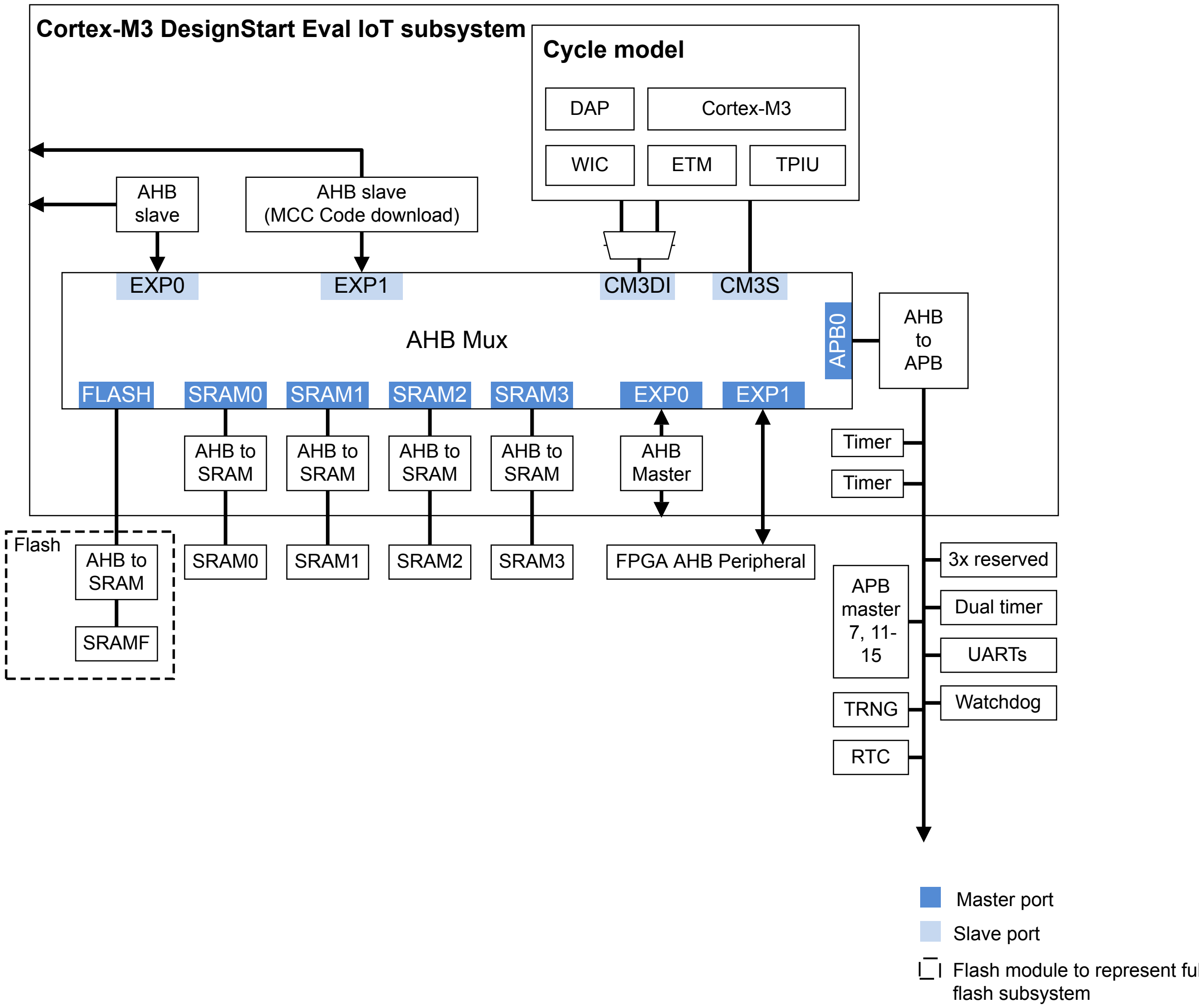

Figure 3-2 Cortex-M3 DesignStart Eval IoT subsystem

The ports of the IoT subsystem in the example system are connected as follows:

- The debug port is routed to the baseboard. It is connected to the USB port using the CMSIS-DAP protocol.
- The trace port is routed to the baseboard connector.
- One AHB master port is used for peripherals.
- One AHB master port is unused.
- One AHB slave port is used for the *Motherboard Configuration Controller* (MCC) for code download.
- One AHB slave port is unused.
- Six APB master ports that are used for closely coupled peripherals.
- Three APB master ports that are reserved for memory system and flash control.
- Five APB master ports that are unused.

3.3.3 ARM Cordio radio integration

The IoT subsystem in Cortex-M3 DesignStart Eval is designed to integrate with the ARM Cordio radio IP.

If the ARM Cordio radio IP is integrated, the following are used:

- One AHB-Lite master port for driving the Link Layer Control Channel interface.
- One AHB-Lite slave port for *Direct Memory Access* (DMA).

The radio also requires interrupt and GPIO connections for status and control.

For more information on adding the ARM radio IP to the subsystem, see the *ARM® Cortex®-M3 DesignStart™ Eval Customization Guide*.

3.3.4 Example peripheral integration

There are several APB master expansion ports from the IoT subsystem that are used for tightly coupled peripherals for application use.

These APB ports provide functions that are common to many SoC developments. These ports are shown in *Figure 3-1 Cortex-M3 DesignStart Eval example system* on page 3-23.

ARM recommends that you use three of the APB expansion ports with a cache (port 3), and embedded flash memory controller (ports 9 and 10). The design hierarchy instantiates `m3ds_simple_flash` as a wrapper for the AHB to SRAM instance. This wrapper provides access to the **FLASH0** AHB port, and the APB ports. Therefore, a full embedded flash subsystem can be used as a drop-in replacement.

The other tightly coupled peripherals are:

- *Cortex-M System Design Kit* (CMSDK) Dual Timer.
- Two CMSDK UARTs, where one peripheral is connected to a serial port on the MPS2+ baseboard.
- PL031 *Real Time Clock* (RTC), which relies on being initialized by the *Motherboard Configuration Controller* (MCC) when the board is powered on.
- CMSDK Watchdog.
- Stand-alone *True Random Number Generator* (TRNG).

3.4 FPGA peripherals and Arduino shield support

Cortex-M3 DesignStart Eval uses an example system integration to show how a fully functioning system can be built up. This system is built around the ARM *Versatile Express Cortex-M Prototyping System* (V2M-MPS2+), which provides several peripherals and connectors.

All the FPGA peripherals are connected to one of the AHB master expansion ports.

In addition to the memories integrated in the IoT subsystem in Cortex-M3 DesignStart Eval (and implemented using FPGA block RAM), the on-board ZBT SSRAM, and PSRAM devices are connected to the processor.

The following table lists the included peripherals:

Table 3-2 Included peripherals

Location	Supported peripherals
Integrated in subsystem	The supported peripherals internal to the example system include: • Dual APB timer.
Example system	The closely coupled peripherals in the example system include: • Dual timer. • UART. • *Real Time Clock* (RTC). • Watchdog. • *True Random Number Generator* (TRNG).
MPS2+ base board	The supported peripherals include: • UART. • PL022 (synchronous serial port) for LCD module. • I^2C audio output. • MicroSD card SPI adapter. • Ethernet using a memory-mapped interface. • VGA using a memory-mapped interface.
Arduino shield expansion for the MPS2+ platform	The supported peripherals include: • GPIO. • Two SPI ports. • SPI for ADC. • Two I^2C ports. • Three UART ports.

Note

Cortex-M3 DesignStart Eval supports the V2M-MPS2+ FPGA platform, and not the V2M-MPS2 FPGA platform.

The uSDCard SPI adapter is included with newly purchased MPS2+ FPGA platforms and is also available for purchase from ARM to use with MPS2+ FPGA platforms that you already own.

The Arduino shield adapter board is not included with the MPS2+ FPGA platform, and can be purchased from ARM.

The design hierarchy places the peripherals according to the following groups:

- Closely coupled peripherals, and the four primary GPIO peripherals.
- MPS2+ board APB peripherals.
- MPS2+ board external RAM components, and memory mapped devices that include two unused GPIO peripherals.

The following diagram shows the different peripheral groups:

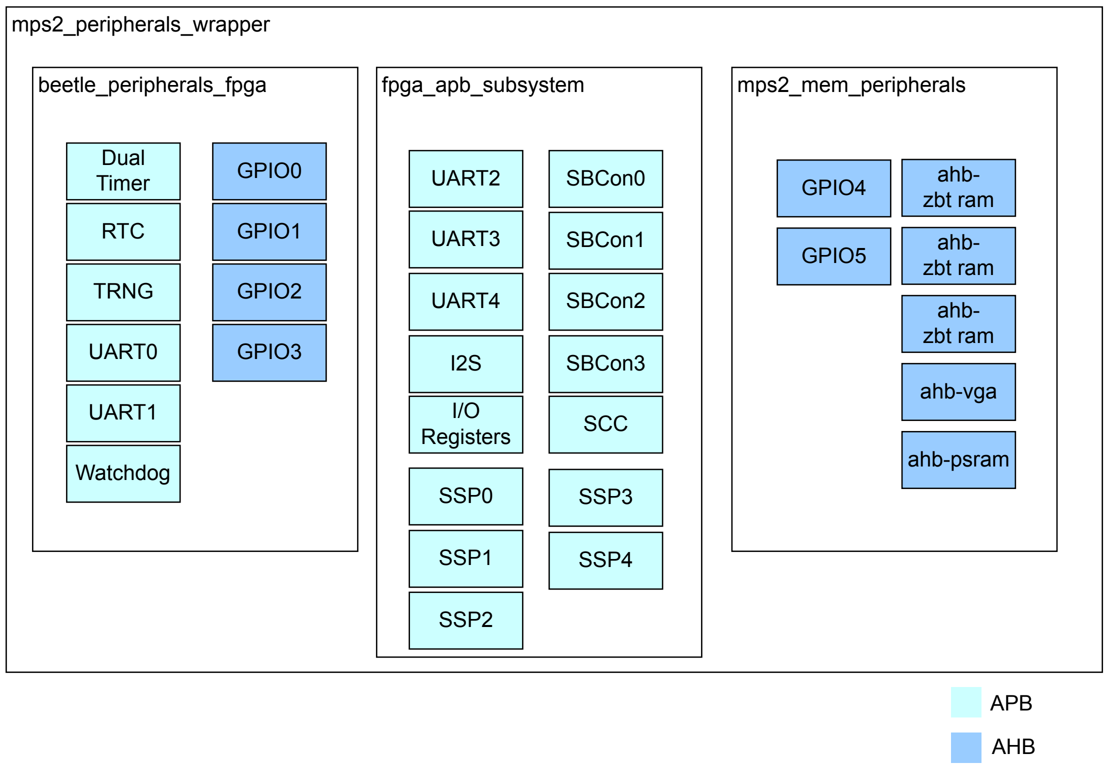

Figure 3-3 Peripheral groups

3.5 mbed OS support

The MPS2+ FPGA platform, with the Cortex-M3 DesignStart Eval image, is a supported mbed target. You can use the mbed online toolchain to develop applications using the mbed OS.

To compile and run a basic LED blinking program on the MPS2+ platform using the online mbed compiler, follow these steps:

1. Go to the mbed compiler site at *http://developer.mbed.org/compiler/*.
2. Import the program and libraries into your workspace by selecting **New program**. This gives a dialog for you to select the following:

Platform	ARM Cortex-M3 DesignStart
Template	Blinky LED Hello World
Program name	`mbed_blinky`

3. Select **Compile** to compile the program and produce a `.bin` file, which will be downloaded automatically.
4. Rename the `.bin` file to follow an 8:3 character format (for example, `mbed_bl.bin`).
5. Ensure that your MPS2+ platform is powered up and connected to the computer. Your computer should recognize the MPS2+ platform as an external USB drive, named `V2M_MPS2`.
6. Copy the `.bin` file to the following MPS2+ platform drive:

   ```
   V2M_MPS2:\SOFTWARE
   ```

7. Edit the following file to reference the `.bin` file.

   ```
   V2M_MPS2:\MB\HBI0263C\AN511\images.txt
   ```

   For example:

   ```
   TITLE: Versatile Express Images Configuration File

   [IMAGES]
   TOTALIMAGES: 1                     ;Number of Images (Max: 32)

   IMAGE0ADDRESS: 0x00000000          ;Please select the required executable program
   ;IMAGE0FILE: \SOFTWARE\st_m3ds.axf ; - M3 DesignStart selftest
   IMAGE0FILE: \SOFTWARE\mbed_bl.bin  ; Compiled on mbed.org
   ```

Note

If you are using the mbed command-line interface, use **cm3ds_mps2** as the platform name.

The MPS2+ FPGA platform, as delivered, does not support using the mbed bootloader to update the firmware, since the *Motherboard Configuration Controller* (MCC) loads the firmware at powerup.

You must also ensure that any software image uses the 8:3 filename format. To emulate the normal mbed drag and drop programming behavior:

1. Copy the .bin file (matching the `images.txt` file).
2. Copy the `reboot.txt` file to the `V2M_MPS2` drive. This causes the new image to be loaded.

The *http://developer.mbed.org* website has up-to-date information about the support for this platform, with links to the example software.

Chapter 4
Functional description

This chapter describes the memory maps, I/O pins, and TRNG registers in Cortex-M3 DesignStart Eval.

It contains the following sections:

4.1 Memory map overview

The following figure shows the memory map for Cortex-M3 DesignStart Eval:

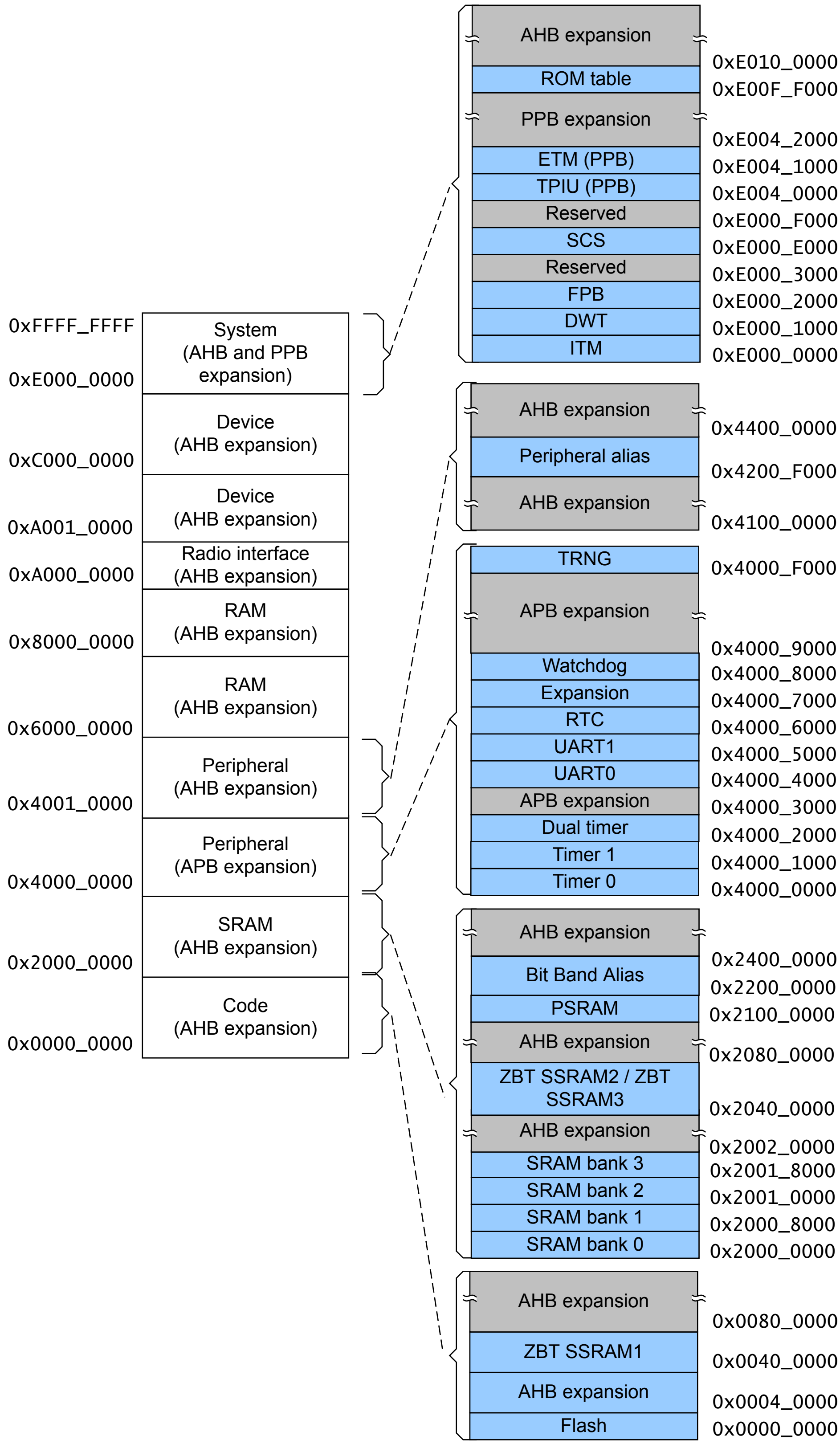

Figure 4-1 Top-level memory map

This section contains the following subsections:

4.1.1 Code and RAM memory map

The following table shows the memory map for Code and RAM in Cortex-M3 DesignStart Eval:

Table 4-1 Code and RAM memory map

Type	Start	End	Peripheral	Size	Subsystem connection	Comment	Bit band region
Code	`0x00000000`	`0x0003FFFF`	Flash	256KB	TARGFLASH0	FPGA Block RAM	-
	`0x00040000`	`0x003FFFFF`	AHB expansion	-	TARGEXP1	-	-
	`0x00400000`	`0x007FFFFF`	ZBT SSRAM1	4MB	TARGEXP1	User memory	-
	`0x00800000`	`0x1FFFFFFF`	AHB expansion	-	TARGEXP1	-	-
RAM	`0x20000000`	`0x20007FFF`	SRAM0	32KB	TARGSRAM0	FPGA Block RAM	Yes
	`0x20008000`	`0x2000FFFF`	SRAM1	32KB	TARGSRAM1	FPGA Block RAM	Yes
	`0x20010000`	`0x20017FFF`	SRAM2	32KB	TARGSRAM2	FPGA Block RAM	Yes
	`0x20018000`	`0x2001FFFF`	SRAM3	32KB	TARGSRAM3	FPGA Block RAM	Yes
	`0x20020000`	`0x203FFFFF`	AHB expansion	2MB	TARGEXP1	-	Partial
	`0x20400000`	`0x207FFFFF`	ZBT SSRAM2 and ZBT SSRAM3	4MB	TARGEXP1	User memory	-
	`0x20800000`	`0x20FFFFFF`	AHB expansion	8MB	AHB	-	-
	`0x21000000`	`0x21FFFFFF`	PSRAM	16MB	TARGEXP1	User memory	-
	`0x22000000`	`0x23FFFFFF`	SRAM or Expansion alias	32MB	-	Bit band alias	-
	`0x24000000`	`0x3FFFFFFF`	AHB expansion	-	TARGEXP1	-	-

Each bit in the 1MB bit band region can be accessed individually by making an access to the corresponding word in the 32MB bit band alias region. Only bit [0] is used when you make an access to the bit band alias region.

The base bit band region can be accessed directly in the same way as normal memory, using word, halfword, or byte accesses.

4.1.2 Peripheral memory map

The following table shows the memory map for peripherals in Cortex-M3 DesignStart Eval:

Note

Each bit in the 1MB bit band region can be accessed individually by making an access to the corresponding word in the 32MB bit band alias region. Only bit [0] is used when you make an access to the bit band alias region. The base bit band region can be accessed directly in the same way as normal memory, using word, halfword, or byte accesses.

Table 4-2 Peripheral memory map

Start	End	Peripheral	Size	Subsystem connection	Comment	Bit band region
0x40000000	0x40000FFF	Timer0	4KB	APB[0]	Internal to subsystem	Yes
0x40001000	0x40001FFF	Timer1	4KB	APB[1]	Internal to subsystem	Yes
0x40002000	0x40002FFF	Dual Timer	4KB	APB[2]	Dual timer	Yes
0x40003000	0x40003FFF	Reserved	4KB	APB[3]	Use for flash cache	Yes
0x40004000	0x40004FFF	UART0	4KB	APB[4]	MCC UART	Yes
0x40005000	0x40005FFF	UART1	4KB	APB[5]	MPS2+ UART	Yes
0x40006000	0x40006FFF	RTC	4KB	APB[6]	Real Time Clock	Yes
0x40007000	0x40007FFF	APB expansion	4KB	APB[7]	-	Yes
0x40008000	0x40008FFF	Watchdog	4KB	APB[8]	Watchdog	Yes
0x40009000	0x40009FFF	Reserved	4KB	APB[9]	Use for flash memory	Yes
0x4000A000	0x4000AFFF	Reserved	4KB	APB[10]	Use for flash memory	Yes
0x4000B000	0x4000BFFF	APB expansion	4KB	APB[11]	-	Yes
0x4000C000	0x4000CFFF	APB expansion	4KB	APB[12]	-	Yes
0x4000D000	0x4000EFFF	APB expansion	8KB	APB[13]	-	Yes
0x4000E000	0x4000EFFF	APB expansion	4KB	APB[14]	-	Yes
0x4000F000	0x4000FFFF	TRNG	4KB	APB[15]	True Random Number Generator	Yes
0x40010000	0x40010FFF	GPIO0	4KB	AHB	EXP[15:0]	Yes
0x40011000	0x40011FFF	GPIO1	4KB	AHB	EXP[31:16]	Yes
0x40012000	0x40012FFF	GPIO2	4KB	AHB	EXP[47:32]	Yes
0x40013000	0x40013FFF	GPIO3	4KB	AHB	EXP[51:48]	Yes
0x4001F000	0x4001FFFF	SysCtrl	4KB	AHB	CMSDK system controller	Yes
0x40020000	0x40020FFF	PL022	4KB	APB	Connector J21	Yes

Table 4-2 Peripheral memory map (continued)

Start	End	Peripheral	Size	Subsystem connection	Comment	Bit band region
0x40021000	0x40021FFF	PL022	4KB	APB	LCD touch screen	Yes
0x40022000	0x40022FFF	Serial control	4KB	APB	LCD module	Yes
0x40023000	0x40023FFF	Serial control	4KB	APB	Audio configuration	Yes
0x40024000	0x40024FFF	I^2C	4KB	APB	Audio	Yes
0x40025000	0x40025FFF	SPI ADC	4KB	APB	EXP[19:16]	Yes
0x40026000	0x40026FFF	SPI Shield0	4KB	APB	EXP[14:11]	Yes
0x40027000	0x40027FFF	SPI Shield1	4KB	APB	EXP[40:38], EXP[44]	Yes
0x40028000	0x40028FFF	FPGA control	4KB	APB	-	Yes
0x40029000	0x40029FFF	I^2C Shield0	4KB	APB	EXP[15], EXP[5]	Yes
0x4002A000	0x4002AFFF	I^2C Shield1	4KB	APB	EXP[41], EXP[31]	Yes
0x4002C000	0x4002CFFF	UART2	4KB	APB	Shield0 - EXP[4], EXP[0]	Yes
0x4002D000	0x4002DFFF	UART3	4KB	APB	Shield1 - EXP[30], EXP[26]	Yes
0x4002E000	0x4002EFFF	UART4	4KB	APB	Shield BT - EXP[24], EXP[23]	Yes
0x4002F000	0x4002FFFF	SCC Registers	4KB	APB	Serial configuration	Yes
0x40030000	0x40030FFF	GPIO4	4KB	AHB	No external connection	Yes
0x40031000	0x40031FFF	GPIO5	4KB	AHB	No external connection	Yes
0x40200000	0x40FFFFFF	Ethernet	-	APB	16-bit memory interface	-
0x41000000	0x4100FFFF	VGA console	-	AHB	Memory interface	-
0x41100000	0x4113FFFF	VGA image	256KB	AHB	Memory interface	-
0x41140000	0x5FFFFFFF	AHB expansion	-	TARGEXP1	-	-
0x42000000	0x43FFFFFF	Peripherals	32MB	-	Bit band alias	-

4.1.3 External RAM, device, and system memory map

The following table shows the memory map for external RAM, devices, and system in Cortex-M3 DesignStart Eval.

Table 4-3 External RAM, device, and system memory map

Type	Start	End	Peripheral	Size	Subsystem connection	Comment
External RAM	`0x60000000`	`0x7FFFFFFF`	AHB expansion	512MB	TARGEXP1	-
	`0x80000000`	`0x9FFFFFFF`	AHB expansion	512MB	TARGEXP1	-
Device	`0xA0000000`	`0xA000FFFF`	AHB expansion	64KB	TARGEXP0	-
	`0xA0010000`	`0xBFFFFFFF`	AHB expansion	-	TARGEXP1	-
	`0xC0000000`	`0xDFFFFFFF`	AHB expansion	512MB	TARGEXP1	-
System	`0xE0000000`	`0xE0000FFF`	*Instrumentation Trace Macrocell* (ITM)	4KB	Internal *Private Peripheral Bus* (PPB)	-
System	`0xE0001000`	`0xE0001FFF`	*Data Watchpoint and Trace* (DWT)	4KB	Internal PPB	-
System	`0xE0002000`	`0xE0002FFF`	*Flash Patch and Breakpoint* (FPB)	4KB	Internal PPB	-
System	`0xE0003000`	`0xE000DFFF`	Reserved	-	Internal PPB	-
System	`0xE000E000`	`0xE000EFFF`	*System Control Space* (SCS)	4KB	Internal PPB	-
System	`0xE000F000`	`0xE003FFFF`	Reserved	-	Internal PPB	-
System	`0xE0040000`	`0xE0040FFF`	*Trace Port Interface Unit* (TPIU)	4KB	Internal PPB	Trace
System	`0xE0041000`	`0xE0041FFF`	*Embedded Trace Macrocell* (ETM)	4KB	Internal PPB	Trace
System	`0xE0042000`	`0xE00FEFFF`	Reserved	-	-	-
System	`0xE00FF000`	`0xE00FFFFF`	ROM table	4KB	AHB	Debug and trace
System	`0xE0100000`	`0xFFFFFFFF`	AHB expansion	-	-	-

4.2 Interrupt mapping

The following table describes the interrupt assignments in Cortex-M3 DesignStart Eval:

Table 4-4 Interrupt assignments

INTISR bit	Source	Description
NMI	Watchdog	Watchdog
0	UART0	UART0 Tx and Rx combined
1	Reserved	Unused
2	UART1	UART1 Tx and Rx combined
3	Reserved	Reserved for APB slaves
4	Reserved	Reserved for APB slaves
5	RTC	Real Time Clock
6	GPIO0	Combined
7	GPIO1	Combined
8	Timer0	Timer0
9	Timer1	Timer1
10	Dual Timer	Dual Timer
11	Reserved	Reserved for APB slaves
12	UART 0/1/2/3/4	UART 0/1/2/3/4 overflow
13	Reserved	Reserved for APB slaves
14	Reserved	Unused
15	MPS2+ board	Touch screen
16	GPIO0-0	GPIO0 pins
17	GPIO0-1	GPIO0 pins
18	GPIO0-2	GPIO0 pins
19	GPIO0-3	GPIO0 pins
20	GPIO0-4	GPIO0 pins
21	GPIO0-5	GPIO0 pins
22	GPIO0-6	GPIO0 pins
23	GPIO0-7	GPIO0 pins
24	GPIO0-8	GPIO0 pins
25	GPIO0-9	GPIO0 pins
26	GPIO0-10	GPIO0 pins
27	GPIO0-11	GPIO0 pins
28	GPIO0-12	GPIO0 pins
29	GPIO0-13	GPIO0 pins
30	GPIO0-14	GPIO0 pins
31	GPIO0-15	GPIO0 pins
32	Reserved	Reserved for flash

Table 4-4 Interrupt assignments (continued)

INTISR bit	Source	Description
33	Reserved	Reserved for flash
34	Reserved	Reserved for Cordio BT4
35	Reserved	Reserved for Cordio BT4
36	Reserved	Reserved for Cordio BT4
37	Reserved	Reserved for Cordio BT4
38	Reserved	Reserved for Cordio BT4
39	Reserved	Reserved for Cordio BT4
40	Reserved	Reserved for Cordio BT4
41	Reserved	Reserved for Cordio BT4
42	GPIO2	Combined
43	GPIO3	Combined
44	TRNG	True Random Number Generator
45	UART2	Combined Tx and Rx
46	UART3	Combined Tx and Rx
47	Ethernet	Ethernet interrupt
48	I^2S	I^2S interrupt
49	MPS2 SPI0	SPI header interrupt
50	MPS2 SPI1	CLCD SPI interrupt
51	MPS2 SPI2	ADC for Shield
52	MPS2 SPI3	Shield0 SPI
53	MPS2 SPI4	Shield1 SPI
54	GPIO4	Combined
55	GPIO5	Combined
56	UART4	Tx and Rx combined

4.3 I/O signals

This section describes the I/O signals at the top level of the example system inCortex-M3 DesignStart Eval.

For more details about the interfaces, see the *ARM® Cortex®-M3 DesignStart™ Eval FPGA User Guide*.

This section contains the following subsections:

4.3.1 Clock, reset, and user I/O signals

The following table displays the clock, reset, and user I/O signals:

Table 4-5 Clock, reset, and user I/O signals

Signal	Direction	Description
OSCCLK[0]	Input	Main clock at 50MHz.
OSCCLK[1]	Input	Main clock at 24.576MHz.
OSCCLK[2]	Input	Main clock at 25MHz.
CB_nPOR	Input	Powerup reset (active-LOW).
CB_nRST	Input	Reset (released after code download is done, active-LOW).
CLKOUT[1:0]	Output	PLL generated clock
CLKIN[1:0]	Input	Loop back clock from **CLKOUT**.
USER_LED[1:0]	Output	LEDs
USER_PB[1:0]	Input	Push buttons
EXP[51:0]	Input and output	I/O expansion port

4.3.2 Debug and trace signals

The following table displays the debug and trace signals:

Table 4-6 Debug and trace signals

Signal	Direction	Description
CS_nTRST	Input	JTAG nTRST
CS_TDI	Input	JTAG Test Data Input
CS_TCK	Input	Serial Wire Debug clock or JTAG clock.
CS_TMS	Input and output	Serial Wire Debug I/O or JTAG Test Mode Select.
CS_TDO	Output	SWV or JTAG Test Data Output.
CS_nSRST	Input	Not required for Cortex-M.

Table 4-6 Debug and trace signals (continued)

Signal	Direction	Description
CS_TRACECLK	Output	Trace clock
CS_TRACECTL	Output	Trace control
CS_TRACEDATA[15:0]	Output	Trace data. Only pins [3:0] are used by Cortex-M3 processor.

4.3.3 ZBT SSRAM signals

The following tables describe the signals for the ZBT SSRAMs:

Table 4-7 64-bit ZBT SSRAM1 connections

Signal	Direction	Description
SSRAM1_DQ[63:0]	Input and output	Data
SSRAM1_DQP[7:0]	Input and output	Parity data (not used)
SSRAM1_CLK[1:0]	Output	Clock
SSRAM1_A[20:0]	Output	Address
SSRAM1_nBW[7:0]	Output	Byte lane writes (active-LOW)
SSRAM1_nCE1	Output	Chip select
SSRAM1_nWE	Output	Write enable (lower 32-bit, active-LOW)
SSRAM1_nCEN	Output	Write clock enable (active-LOW, tied to 0)
SSRAM1_nOE	Output	Output enable (active-LOW)
SSRAM1_MODE	Output	Not used (tied to 0)
SSRAM1_ADVnLD	Output	Not used (tied to 0)
SSRAM1_ZZ	Output	Not used (tied to 0)

Table 4-8 32-bit ZBT SSRAM2 connections

Signal	Direction	Description
SSRAM2_DQ[31:0]	Input and output	Data (byte lane #A)
SSRAM2_DQP[3:0]	Input and output	Parity data (not used)
SSRAM2_CLK	Output	Clock
SSRAM2_A[20:0]	Output	Address
SSRAM2_nBW[3:0]	Output	Byte lane writes (active-LOW)
SSRAM2_nCE1	Output	Chip select
SSRAM2_nWE	Output	Write enable (lower 32-bit, active-LOW)
SSRAM2_nCEN	Output	Write clock enable (active-LOW, tied to 0)
SSRAM2_nOE	Output	Output enable (active-LOW)
SSRAM2_MODE	Output	Not used (tied to 0)
SSRAM2_ADVnLD	Output	Not used (tied to 0)
SSRAM2_ZZ	Output	Not used (tied to 0)

Table 4-9 32-bit ZBT SSRAM3 connections

Signal	Direction	Description
SSRAM3_DQ[31:0]	Input and output	Data (byte lane #A)
SSRAM3_DQP[3:0]	Input and output	Parity data (not used)
SSRAM3_CLK	Output	Clock (SSRAM1_CLK)
SSRAM3_A[20:0]	Output	Address
SSRAM3_nBW[3:0]	Output	Byte lane writes (active-LOW)
SSRAM3_nCE1	Output	Chip select
SSRAM3_nWE	Output	Write enable (lower 32-bit, active-LOW)
SSRAM3_nCEN	Output	Write clock enable (active-LOW, tied to 0)
SSRAM3_nOE	Output	Output enable (active-LOW)
SSRAM3_MODE	Output	Not used (tied to 0)
SSRAM3_ADVnLD	Output	Not used (tied to 0)
SSRAM3_ZZ	Output	Not used (tied to 0)

4.3.4 Static memory interface signals for PSRAM and Ethernet peripherals

The following table describes the I/O signals for the static memory interface:

Note

All enable, select, reset, read, and sleep signals are active-LOW.

Table 4-10 Static memory interface signals

Signal	Direction	Description
SMB_ETH_IRQ	Input	Ethernet IRQ interrupt
SMB_DQ[15:0]	Input and output	Data
SMB_A	Output	Address
SMB_ETH_nCS	Output	Ethernet chip select
SMB_nLB	Output	Lower byte select
SMB_nOE	Output	Output enable
SMB_nRD	Output	Read
SMB_nRESET	Output	Reset
SMB_nUB	Output	Upper byte select
SMB_nWE	Output	Write enable
SMB_PSRAM_nCE[1:0]	Output	PSRAM chip select
SMB_nZZ	Output	Sleep

4.3.5 UART and SPI signals

The following tables describe the UART and SPI signals:

Table 4-11 UART signals

Signal	Direction	Description
UART_RXD	Input	UART received data
UART_TXD	Output	UART transmit data

Table 4-12 SPI signals

Signal	Direction	Description
SPI_MISO	Input	SPI data in
SPI_SCK	Output	SPI clock
SPI_MOSI	Output	SPI data out
SPI_nSS	Output	SPI device select

4.3.6 VGA and Audio signals

The following tables describe the VGA and Audio signals:

Table 4-13 VGA signals

Register	Direction	Description
VGA_HSYNC	Output	VGA H-Sync
VGA_VSYNC	Output	VGA V-Sync
VGA_R[3:0]	Output	VGA red data
VGA_G[3:0]	Output	VGA green data
VGA_B[3:0]	Output	VGA blue data

Table 4-14 Audio signals

Signal	Direction	Description
AUD_SDOUT	Input	Audio DAC data
AUD_SDA	Input and output	Serial Control data
AUD_MCLK	Output	Audio codec master clock (12MHz)
AUD_SCLK	Output	Audio interface bit clock
AUD_LRCK	Output	Left and right Audio left clock
AUD_SDIN	Output	Audio ADC data
AUD_nRESET	Output	Audio reset (active-LOW)
AUD_SCL	Output	Serial Control port clock

4.3.7 Color LCD touch screen signals

The following tables describe the color LCD and touch screen signals:

Table 4-15 Color LCD signals

Signal	Direction	Description
CLCD_SDO	Input	Data out of color LCD into FPGA.
CLCD_PDH[17:10]	Input and output	Configuration signals
CLCD_PDL[8:1]	Input and output	**CLCD_PD[5:1]** is connected to color LCD header. **CLCD_PD[8:6]** are configuration signals.
CLCD_CS	Output	Chip select (active-LOW).
CLCD_RS	Output	Command or display data selection.
CLCD_WR_SCL	Output	Serial clock out
CLCD_RD	Output	Read data
CLCD_RESET	Output	Reset (active-LOW).
CLCD_BL_CTRL	Output	Back-light enable
CLCD_SDO	Output	Data out of FPGA into color LCD.

Table 4-16 Touch screen signals

Signal	Direction	Description
CLCD_T_CS	Input	Interrupt
CLCD_T_SDA	Input and output	Serial data
CLCD_T_SCL	Output	Serial clock

4.3.8 FPGA configuration signal

The following table describes the FPGA configuration signal:

Table 4-17 FPGA configuration signal

Signal	Direction	Description
FPGA_CONFIG_nLRST	Output	Config interrupt from serial configuration module.

4.3.9 GPIO alternate functions

GPIO alternate functions are used to support two sets of Arduino shield expansion peripherals.

The 52-pin expansion port (EXP port) is driven from GPIO0-3. For each pin, the relevant bit of ALTFUNC must be set to use the alternate function. If this bit is not set to use the alternate function, the pin is used for GPIO. You can control the ALTFUNC bits using the ALTFUNCSET and ALTFUNCCLR registers of each GPIO.

The GPIO alternate functions use the following pins:

Table 4-18 GPIO alternate functions

EXP pin number	GPIO pin	Description
0	**GPIO0[0]**	UART2 (Shield0) Rx data
4	**GPIO0[4]**	UART2 (Shield0) Tx Data

Table 4-18 GPIO alternate functions (continued)

EXP pin number	GPIO pin	Description
26	**GPIO1[10]**	UART3 (Shield1) Rx Data
30	**GPIO1[14]**	UART3 (Shield1) Tx Data
23	**GPIO1[7]**	UART4 Tx data
24	**GPIO1[8]**	UART4 Rx Data
5	**GPIO0[5]**	Shield0 SCL (I^2S)
15	**GPIO0[15]**	Shield0 SDA (I^2S)
31	**GPIO1[15]**	Shield1 SCL (I^2S)
41	**GPIO2[9]**	Shield1 SDA (I^2S)
11	**GPIO0[11]**	Shield0 SCK (SPI)
12	**GPIO0[12]**	Shield0 CS_n (SPI)
13	**GPIO0[13]**	Shield0 MOSI (SPI)
14	**GPIO0[14]**	Shield0 MISO (SPI)
44	**GPIO2[12]**	Shield1 SCK (SPI)
38	**GPIO2[6]**	Shield1 CS_n (SPI)
39	**GPIO2[7]**	Shield1 MOSI (SPI)
40	**GPIO2[8]**	Shield1 MISO (SPI)
19	**GPIO1[3]**	Shield ADC SCK (SPI)
16	**GPIO1[0]**	Shield ADC CS_n (SPI)
18	**GPIO1[2]**	Shield ADC MOSI (SPI)
17	**GPIO1[1]**	Shield ADC MISO (SPI)

4.4 True Random Number Generator (TRNG)

The TRNG is used as a source of entropy for secure internet communications. The *Transport Layer Security* (TLS) in mbed supports TRNG, and drivers are included in the mbed OS. The register descriptions are provided here to help in understanding the driver code.

For ASIC implementation, the TRNG uses a combination of ring oscillators built using digital inverter cells. The TRNG also requires post-production characterization (per implementation), to achieve optimum performance.

For FPGA implementation, the TRNG should be configured to use a *Pseudo Random Bit Sequence* (PRBS). Although this results in a usable entropy source for development, it is not truly random and must not be used in production.

The selection of ring oscillator (`dx_inv_chain`) or PRBS is determined by the `DX_FPGA` define. This define is in the file `logical/fpga_top/verilog/fpga_options_defs.v`.

For more details on the TRNG register attributes and address space, see the *ARM® TrustZone® TRNG True Random Number Generator Technical Reference Manual*.

4.5 System controller peripheral

The system controller provides the following:

- Control outputs for memory remap and a *Performance Monitor Unit* (PMU).

  Note

  Memory remap and PMU are not used in the Cortex-M3 DesignStart Eval example system.

- A mechanism to monitor the source of system reset requests.
- A control to enable system reset when the processor enters a lockup state. When the processor is in the lockup state, it does not execute any instructions.

The following table describes the system controller programmers model:

Table 4-19 System controller programmers model

Address	Name	Type	Reset	Descriptions
`0x4001F000`	REMAP	RW	`0b1`	Bit 0: **1** Enable remap output. **0** Disable remap output. Software symbol: `CMSDK_SYSCON->REMAP`. Not used in Cortex-M3 DesignStart Eval.
`0x4001F004`	PMUCTRL	RW	`0b0`	Bit 0: **1** Enable PMU output. **0** Disable PMU. Software symbol: `CMSDK_SYSCON->PMUCTRL`. Not used in Cortex-M3 DesignStart Eval.
`0x4001F008`	RESETOP	RW	`0b0`	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	`0b0`	Bit 2: If 1, processor lockup state 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 4-19 System controller programmers model (continued)

Address	Name	Type	Reset	Descriptions
`0x4001FFE0`	PID0	RO	`0x27`	Peripheral ID 0. **[7:0]** Part number[7:0]
`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

Chapter 5
Testbench

This chapter describes the components included with the testbench in Cortex-M3 DesignStart Eval.

It contains the following sections:

5.1 Testbench overview

The testbench in Cortex-M3 DesignStart Eval includes the following:

Table 5-1 Testbench contents

Contents	Description
DUT of `tb_fpga_shield`	FPGA top-level module
Reset generator	This includes: • Powerup reset. • Debug reset. • FPGA SCC master interface reset. • FPGA configuration SPI master interface reset.
Clock generator	This includes: • System clock at 25MHz. • Audio master clock at 12MHz. • Audio interface bit clock at 12MHz.
Loop back connection	For UARTs and audio I^2S testing. This includes: • MCU UART (UART0) self-loops back. • Console UART (UART1) self-loops back. • UART2 connects to UART3 in a crossover arrangement. • UART4 self-loops back. • Audio I^2S self-loops back.
Serial debug driver	Drives the Serial Wire Debug pins based on the content on `CXDT.bin` in the `execution_tb` directory generated by the test compilation process. It is only active when running the `cxdt` test to demonstrate Serial Wire Debug functionality.
UART text message capture module	See *5.2 UART text message capture module* on page 5-53.
FPGA configuration SPI master interface driver	Writes a set of test vectors after reset to SRAM1 to demonstrate the connectivity from the baseboard *Motherboard Configuration Controller* (MCC).
Behavioral models	See *5.3 Behavioral models* on page 5-54.
Arduino shield testbench component	This consists of a behavioral model of: • The ARM Arduino shield adapter board. • Two Arduino shields connected to the board. The loop back connection for UART2, 3, and 4 is done in this testbench component.
Pullup and pulldown of various pins	The tie-offs are appropriate for normal FPGA operation.
Test code memory initialization	See *5.4 Test code memory initialization* on page 5-55.

5.2 UART text message capture module

The testbench in Cortex-M3 DesignStart Eval includes a UART text message capture module. The function of the UART capture module is to capture the input data, and output the received characters when it receives the *Carriage Return* (CR) character.

When a program wants to display a message in the simulation environment, it can execute the printf or puts functions. It can also directly call the UART routines to output the message to UART0.

When the program executes the printf or puts functions, the UART output routine executes through retargeting code and outputs the characters to the serial output of UART0. The UART capture module then captures the input data and outputs the characters when it receives the CR character.

The UART capture module captures the input data at 1 bit per cycle to reduce simulation time. This is because the high-speed test mode of the example system UART outputs each bit in one clock cycle to reduce simulation time.

If the UART outputs serial data at a different speed, then you must change the clock that connects to the UART capture module.

You can 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.

You can also use the UART capture module to turn on the SPIs, I^2Cs, and UARTs on both Arduino shields by sending a character value of 0xF.

In order for printf and puts to use the UART, stdout has been retargeted by using the following files:

- m3designstart/software/common/retarget/retarget.c
- m3designstart/software/common/retarget/uart_stdout.c
- m3designstart/software/common/retarget/uart_stdout.h

If you are using the UART, it is also important to call UartStdOutInit() before making any printf calls.

The Keil project files that are included with each of the integration tests all define an 'mps2' build target. If you use this target, the code is compiled with the FPGA_IMAGE define enabled, and the executable files are copied to a suitable 8:3 format file name. The retarget.c file uses the FPGA_IMAGE define to configure the UART for 38 400 Baud, so the tests can be run on the FPGA.

If you build targeting the FPGA, you must either copy the image from the /testcodes/<test>/mps2/ directory into the FPGA board local storage (and configure the correct image to load), or load the image using the debugger (with the mps2 target selected).

5.3 Behavioral models

The following lists the behavioral models included in the testbench of Cortex-M3 DesignStart Eval:

- A simple CMSDK 16-bit SRAM model for Ethernet access on the SMB bus.
- Third party (ISSI) Asynchronous or Page PSRAM model for PSRAM0 and PSRAM1 access on the SMB bus.
- Two 32-bit third party (GSI) Burst SRAM model for the 64-bit ZBT SSRAM1.
- One 32-bit third party (GSI) Burst SRAM model each for the 32-bit ZBT SSRAM2 and SSRAM3.

5.4 Test code memory initialization

At simulation time 0, the flash RAM is initialized in `tb_fpga_shield.u_fpga_top.u_fpga_system.u_user_partition.u_ahb_blockram_128`, based on the contents of `flash_main.ini`.

When a simulation is invoked using `make run`, the `flash_main.ini` file is converted from a test code binary file. The test code binary file was previously generated in the `test/` directory when the test was compiled using `make`.

Related references

6.2 Integration test list on page 6-58.
6.5 Compiling the integration tests on page 6-66.

Chapter 6
Simulation and integration tests

This chapter describes the integration tests and how to run the simulation.

It contains the following sections:

6.1 Scope of integration tests

The tests that are described in this chapter are integration tests.

Before you run the integration tests, you must ensure that each individual IP element in your design has been thoroughly validated as a stand-alone component.

The main purpose of the integration tests is to demonstrate that the interfaces are connected to permit each function to operate. The integration tests do not validate that there is no scenario causing an incorrect operation.

The integration tests typically:

- Check the ranges of operation. For example, accessing the top and bottom of a memory region.
- Cover the range of operation types. This includes the read and write operations, and at least one response that should return an error or fault.

If there are several interrupts for a peripheral, then each interrupt should be checked in isolation.

6.2 Integration test list

The Cortex-M3 DesignStart Eval Execution Testbench provides a series of integration tests that you can run during simulation. These tests show that the deliverables have been set up correctly and demonstrate certain behaviors of the processor.

If a test does not pass, perform the following:

- Run the `hello` test and make sure it passes. Debug this test first, if you have problems.
- Rerun the failing test with the Tarmac trace turned on in the testbench configuration options, as described in *6.4.1 Testbench configuration options* on page 6-63.

The tests provided are written in C or ARM assembly code using the CMSIS framework.

The following are the tests and their functions:

Table 6-1 Integration test list

Test name	Description
`hello`	Simple test to verify that the testbench is running correctly. Always check that this test passes when investigating any problem. The `printf` output is redirected to the UART output using UART0.
`dhry`	Simple benchmark example. You can manually measure the length of the test loop by monitoring the `HADDRI` bus at the `CORTEXM3INTEGRATIONDS` level for a repeating point in the loop.
`apb_mux_tests`	Detects the presence of various APB devices that are connected to the APB multiplexer.
`cxdt`	Demonstrate Serial Wire Debug functionality by reading the CoreSight ROM table to determine that components are present in the system.
`default_slaves_tests`	Tests the default slave activation. `default_slaves_tests` accesses invalid memory locations.
`designtest_m3`	Tests the following peripherals that are specific to the MPS2+ FPGA platform example system: • Register read and write to SCC and FPGA control registers. • Accesses to Ethernet, PSRAMs, and ZBT SSRAMs. • Data transmission to LCD I^2C.
`dualtimer_demo`	Demonstrates the features of APB dual timer.
`gpio_driver_tests`	Tests the GPIO device driver functions on GPIO0.
`gpio_tests`	Tests registers read and write, interrupt, masked access, and reserved register addresses for AHB GPIO0 to GPIO5.
`interrupt_demo`	Demonstrates the TIMER0 interrupt, GPIO0 interrupt, UART2 and UART3 interrupt.
`memory_tests`	Tests the system memory map.
`rtx_demo`	Demonstrates the Keil RTX Real-Time Operating System.
`self_reset_demo`	Demonstrates the **SYSRESETREQ** reset and lockup reset.

Table 6-1 Integration test list (continued)

Test name	Description
`sleep_demo`	Demonstrates the sleep features of: • `WFI SLEEP`. • `WFE SLEEP`. • `SLEEP-ON-EXIT`. • `WFI DEEP SLEEP`. • `WFE DEEP SLEEP`. • `SLEEP-ON-EXIT` deep sleep. • `WFI DEEP SLEEP` with `WIC`. • `WFE DEEP SLEEP` with `WIC`. • `SLEEP-ON-EXIT` with `WIC`. • `WFI DEEP SLEEP` with `WIC` switched off.
`timer_driver_tests`	Tests the timer device driver functions on TIMER0.
`timer_tests`	Tests the functionality of APB TIMER0 and TIMER1.
`uart_driver_tests`	Tests the UART device driver functions on UART2 and UART3.
`uart_tests`	Tests the functionality of UART0 to UART4. UART2 and 3 are connected in a cross arrangement in the testbench and the other UARTs are self-looped back.
`watchdog_demo`	Demonstrates the APB watchdog.

6.3 Simulation environment

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

To run simulations, you require some software applications and an environment to run them. These are not included with the Cortex-M3 DesignStart Eval. You can configure the makefile to use your preferred tools by default.

This section contains the following subsections:

6.3.1 Supported Verilog simulators

The simulation environment supports the following Verilog simulators:

- Mentor QuestaSim.
- Cadence IUS.
- Synopsys VCS.

For more information on recommended versions of the Verilog simulators, see *6.3.4 Setting up the simulation tools* on page 6-61.

6.3.2 Supported compiler toolchains

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

You can recompile the precompiled example tests or your own tests using any of the following:

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

Keil MDK is available only for the Windows platform. Therefore, to use Keil MDK, you must carry out the software compilation and the simulation in two separate stages. A limited term license of Keil MDK is included with the Cortex-M3 DesignStart Eval product. You will need to install this license to compile some of the tests that are provided.

For more information on recommended versions of the compiler toolchain, see *6.3.4 Setting up the simulation tools* on page 6-61.

6.3.3 Simulation model configuration

Cortex-M3 DesignStart Eval includes an ARM Cycle Model of the processor integration level. This model is also referred to as a DSM in this document.

You can run simulations without using the Cycle Model, but you will have less visibility when you need to debug how the code is executing, or has executed.

License installation

To use the ARM Cycle Model with your simulator, you are required to install a license, which you can obtain from the ARM website using the information provided when you registered for access to Cortex-M3 DesignStart Eval. You must have a software license server available on your network.

ARM Cycle Model products use the FLEXNet License Manager and you can either install a *node locked* license or a *floating* license. For more information, see the following FAQ topics in the *ARM Technical Support Knowledge Articles*:

- *How do I install my node locked license?*
- *How do I install my floating license?*

Environment

The Cycle Model only supports the 64-bit simulations, and must be run on the Linux platform.

When you use the model, the makefile sets the following environment variables:

`TARMAC_ENABLE` If this variable is set, the processor instruction trace log is generated in `cm_tarmac.log`.

`LD_LIBRARY_PATH` This variable ensures that the system shared library search path includes the Cycle Model directory.

6.3.4 Setting up the simulation tools

To set up the simulation tools, follow these steps:

1. Install a minimum version of the following:
   - GNU Make version 3.80.
   - GNU GCC version 4.7.2 or 4.8.3.
   - GNU Binutils 2.22.
2. Install at least one of the supported Verilog simulators. The recommended minimum versions are:
   - Mentor QuestaSim version 10.4e_1. You will need to set the environment variable `QUESTASIM_HOME` to reference your QuestaSim installation directory.
   - Cadence IUS version 15.20.008.
   - Synopsys VCS version 2016.06-SP2.

If you need to compile your own test or recompile an example test, install at least one of the supported C compilers. The recommended minimum versions are:

- ARM *Development Studio 5* (DS-5) version 5.06.409.
- ARM Keil *Microcontroller Development Kit* (MDK) version 5.22.
- *GNU Tools for* ARM *Embedded Processors* (ARM GCC) version 5-2016q2.

6.3.5 Optional FSDB waveform

If you want to do an FSDB waveform dump, install the recommended Verdi version 2016.06-SP2 and set the environment variable `VERDI_HOME` to your Verdi installation directory.

6.3.6 Tarmac trace

If you run a simulation using the processor Cycle Model, you can configure the testbench to generate a Tarmac log file in the directory where the test runs.

The log file uses cycle counts as a timestamp, and prints the following information:

IT Instruction taken (condition code passed) with disassembly.

IS Instruction skipped (condition code failed) with disassembly.

R Register update.

B Bus access (instruction, data, or system bus).

M Memory access (duplicating bus accesses, all bus accesses are memory accesses).

E Exception information and events.

For more detailed explanation of the Tarmac trace format, see:

```
m3designstart/logical/testbench/execution_tb/tarmac.txt
```

Related references

6.4.1 Testbench configuration options on page 6-63.
6.6 Running the simulation on page 6-69.

6.4 Compiling the RTL

After the simulation environment is configured, with the appropriate licenses available, you can compile the Verilog RTL in the `execution_tb` directory.

To clean the previous RTL compile, execute the following:

```
make clean
```

To compile the RTL, execute the following:

```
make compile
```

Only the following configuration options are supported for compiling the RTL:

- `SIMULATOR`.
- `DSM`.
- `SIM_64BIT`.
- `SIM_VCD`.
- `FSDB`.

For example, to compile the RTL with the processor Cycle Model using the 64-bit Synopsys VCS, execute the following:

```
make compile SIMULATOR=vcs SIM_64BIT=yes DSM=yes
```

If the compilation is successful, then the following message is displayed:

```
>> Testbench compile with vcs and DSM=yes  completed successfully, log in vcs_compile.log
```

If the compilation fails, then the following is displayed:

```
 >> ERROR: Testbench compile failed, check vcs_compile.log
```

The compile log is written to `<sim>_compile.log`.

Note

When you compile with `DSM=yes`, you might observe that the `CORTEXM3INTEGRATIONDS_dsm` module is reported as uncompiled towards the end of the process. If so, it is compiled at the end (all within the same `make compile` sequence).

This section contains the following subsection:

- *6.4.1 Testbench configuration options* on page 6-63.

6.4.1 Testbench configuration options

You can edit the `make.cfg` file in the `execution_tb` directory to configure the execution testbench. If you change any options in `make.cfg`, then you must run `make clean` and then recompile the testbench.

The following variables control the configuration options:

Table 6-2 Configuration variables

Variables	Description
`DSM`	Determines whether to compile the obfuscated RTL or Cycle Model for the Cortex-M3 processor. The options are: `yes` Compile the Cycle Model. `no` Compile the obfuscated RTL. This is the default value.
`TARMAC`	Enables or disables Tarmac trace. The options are: `yes` Generate Tarmac trace. This is the default value. `no` Disable Tarmac trace generation. Tarmac trace is only generated if the Cycle Model is used.
`SIM_64BIT`	Use 32-bit or 64-bit simulation. The options are: `yes` Use 64-bit simulation. This is the default value. `no` Use 32-bit simulation. 32-bit simulation with the Cycle Model is not supported.
`SIM_VCD`	Enables or disables VCD waveform output. The options are: `yes` Enable. The waveform output is in `dump.vcd`. `no` Disable waveform output. This is the default value.
`GUI`	Runs the simulation in GUI or batch mode. The options are: `yes` Use GUI. `no` Use batch mode. This is the default value.
`FSDB`	Enables or disables FSDB waveform output. The options are: `yes` Enable waveform output. The waveform output is in `dump.fsdb`. `no` Disable waveform output. This is the default value.
`MAX_SIMULATION_TIME`	The maximum integration test time before the test timeouts and exits. The default time is 40 000 microseconds, which is slightly longer than the longest integration test (`gpio_tests`). You can increase this default time, if necessary.
`PLUSARGS`	Allows customers to define arguments during simulation time. There are no arguments by default.
`BUILDOPTS`	Allows customers to define arguments during build time. There are no arguments by default.

Table 6-2 Configuration variables (continued)

Variables	Description
`SIMULATOR`	Defines the Verilog simulator used. The options are any of the following: `mti` Mentor QuestaSim. This is the default value. `vcs` Synopsys VCS. `ius` Cadence IUS.
`TOOL_CHAIN`	Defines the C compiler used. The options are any of the following: `gcc` ARM GCC. This is the default value. `ds5` ARM DS-5. `keil` ARM Keil MDK.

You can override the default values by doing any of the following:

- Edit the default value in `make.cfg`.
- Use the `make` command with an argument in the form of `VARIABLE_NAME=new_value`.

Related references

6.4 Compiling the RTL on page 6-63.
6.5.1 Software compilation makefile settings on page 6-66.
6.5.2 Running make in a test code directory on page 6-67.
6.5.3 Running make in the execution_tb directory on page 6-68.

6.5 Compiling the integration tests

To simplify the evaluation of Cortex-M3 DesignStart Eval, a precompiled binary is supplied for each example integration test.

These integration tests were compiled with DS-5 compiler version 5.06.409 and are present in the `m3designstart/logical/testbench/testcodes/<testname>` directory.

If the binary is accidentally removed, you can manually restore it by copying from `m3designstart/logical/testbench/testcodes/<testname>/precompiled`. There is a makefile that you can use for software compilation inside each test directory. For more information on makefile settings, see *6.5.1 Software compilation makefile settings* on page 6-66.

To compile your software, use the `make` command in one of the following locations:
- `testcodes/<testname>` directory.
- `execution_tb` directory.

Note

Ensure that your compiler tools have been installed properly before compiling the software with the makefile.

This section contains the following subsections:

6.5.1 Software compilation makefile settings

The following table shows the settings for software compilation that the makefiles support:

Table 6-3 Makefile settings

Variable	Descriptions
`TOOL_CHAIN`	Determines the compiler toolchain used. The options are: • `gcc` for ARM GCC. This is the default value. • `ds5` for ARM *Development Studio 5* (DS-5). • `keil` for ARM Keil MDK. For more information, see *Building a test in ARM Keil MDK* on page 6-67.
`TESTNAME`	Name of the software test. This must match the directory name.
`COMPILE_MICROLIB`	Use only for the `TOOLCHAIN=ds5` option. The options are: • **0** for normal C runtime library. This is the default value. • **1** for MicroLIB, which is a C runtime library that is optimized for microcontroller applications.
`USER_DEFINE`	A user-defined C preprocessing macro. Set to `-DCORTEX_M3` for most test codes. This enables a piece of test code to include the correct header for the processor when multiple example systems share the test code. You can add more preprocessing macros for your applications.
`SOFTWARE_DIR`	Shared software directory.
`CMSIS_DIR`	Base location of all ARM *Cortex Microcontroller Software Interface Standard* (CMSIS) source code.
`DEVICE_DIR`	Device-specific support files, for example, header files, and device driver files.
`STARTUP_DIR`	Startup code location.

Table 6-3 Makefile settings (continued)

Variable	Descriptions
ARM_CC_OPTIONS	ARM C Compiler options. Use only for the TOOLCHAIN=ds5 option.
ARM_ASM_OPTIONS	ARM Assembler options. Use only for the TOOLCHAIN=ds5 option.
ARM_LINK_OPTIONS	ARM Linker options. Use only for the TOOLCHAIN=ds5 option.
GNU_CC_FLAGS	GCC compile option. Use only for the TOOLCHAIN=gcc option. By default it uses the Newlib NANO C library.
LINKER_SCRIPT	ARM C Compiler options. Use only for the TOOLCHAIN=gcc option.

Note

A project file that is specific to Keil specifies the options for Keil MDK.

Building a test in ARM Keil MDK

If you use Keil MDK to compile the integration test (`TOOL_CHAIN=keil` during compilation), the `make` process pauses and displays the following:

```
Please ensure all files from Keil MDK test compilation are in the <test> directory before
pressing ENTER
```

If the Windows environment cannot access the Linux environment, you might need to first copy the whole package into the Windows environment.

To compile a test, invoke Keil MDK and open the Keil project file in `<install_directory>/m3designstart/logical/testbench/testcodes/<test>/<test>.uvprojx`, and choose compile. If necessary, copy all generated files back into the Linux environment.

When you are compiling for the simulation environment, use the 'Debug' build target. There is also an 'mps2' target, which will configure the UART to 38 400 Baud, so tests can be run using FPGA hardware.

You can open the Multi Project Workspace provided in `<install_directory>/ m3designstart/logical/testbench/testcodes/keil_multiple/cm3ds_all.uvmpw`. This allows you to batch build all of the targets at once.

6.5.2 Running make in a test code directory

There is a makefile in each test directory in `m3designstart/logical/testbench/testcodes/<testname>` for software compilation.

Settings

The makefile in `testcodes/<testname>` supports several settings during compilation. For more information, see *6.5.1 Software compilation makefile settings* on page 6-66.

Commands

The makefile in `testcodes/<testname>` supports the following commands:

`make all`
This starts the software compilation process for DS-5 or ARM GCC. You can override the variable in the makefile. For example, to compile using DS-5 with the MicroLIB option enabled, execute the following:

```
make all TOOL_CHAIN=ds5 COMPILE_MICROLIB=1
```

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

6.5.3 Running make in the execution_tb directory

You can also use the makefile in `execution_tb` to compile the simulation tests.

Settings

The makefile in `execution_tb` supports several settings during compilation. For more information, see *6.5.1 Software compilation makefile settings* on page 6-66.

Commands

The makefile in `execution_tb` supports the following commands:

`make clean_tests`
This cleans compilation of all tests specified by the `TEST_LIST` variable in the makefile.

You can override the default test list using an argument in the form of:

```
TEST_LIST='hello dhry …'
```

`make clean_all`
This cleans compilation of all tests specified by the `TEST_LIST` variable in the makefile, and the RTL compilation.

`make tests`
This compiles all tests specified by the `TEST_LIST` variable in the makefile using the default GCC compiler. You can override the default compiler using an argument in the form of:

```
TOOL_CHAIN=ds5/gcc/keil
```

You can override the default test list using an argument in the form of a space separated list:

```
TEST_LIST='hello dhry'
```

`make testcode TESTNAME=testname`
This compiles the specified test using the default GCC compiler. You can override the default compiler using an argument in the form of:

```
TOOL_CHAIN=ds5/gcc/keil
```

6.6 Running the simulation

After the RTL and test compilation, you can start the simulation in `execution_tb` directory using the following command:

```
make run TESTNAME=hello
```

Only the following configuration options are supported when running the simulation:

- `SIMULATOR`.
- `DSM`.
- `SIM_64BIT`.
- `FSDB`.
- `GUI`.
- `MAX_SIMULATION_TIME`.
- `TARMAC`.

Note

For more information on the supported configuration options, see *6.4.1 Testbench configuration options* on page 6-63.

If values for any of the configuration options `SIMULATOR`, `DSM`, or `SIM_64BIT` were specified in the previous `make compile` command, then the same set of values must be used for the `make run` command. For example, to run `hello test` on a 64-bit Synopsys VCS simulator with a simulation timeout of 100 000 microseconds using the Cycle Model, execute the following:

```
make run TESTNAME=hello SIMULATOR=vcs SIM_64BIT=no MAX_SIMULATION_TIME=100000us DSM=yes
```

When you run the simulation, a log file is generated and copied to `<sim>_<testname>_run.log` in the `execution_tb` directory. If the test passes, the message `** TEST PASSED **` is displayed in the log file.

You can run all the tests specified in the `TEST_LIST` variable by running the simulation using the `make runall` command. This command supports the same set of configuration options as `make run`. After the simulation has completed, a summary report shows the pass or fail status of each test.

This section contains the following subsection:

- *6.6.1 Investigating test failures* on page 6-69.

6.6.1 Investigating test failures

If a test fails, execute the following steps:

- Run the `hello` test and debug it if it fails.
- Rerun the failing test with the Tarmac trace turned on using the testbench configuration options, as described in *6.4.1 Testbench configuration options* on page 6-63.

Appendix A
Revisions

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

It contains the following section:

- *A.1 Revisions - Cortex®-M3 DesignStart™ Eval* on page Appx-A-71.

A.1 Revisions - Cortex®-M3 DesignStart™ Eval

This section describes the technical changes between released issues of this document.

Table A-1 Issue 00

Change	Location	Affects
First release	-	-