Motorola Built-In Test (MBIT) Diagnostic Software

Test Reference Guide

MBITA/RM1

June 2002 Edition

Notice

Contents

About This Guide

This guide identifies and describes the supported devices and subtests of the Motorola Built-In Test (MBIT) 1.01 diagnostic software for the MVME51*xx* family of boards running the Wind River Systems, Inc. VxWorks® real-time operating system. MBIT also depends on the use of the Tornado® 2.1 development environment.

This guide is a companion to the *Motorola Built-In Test (MBIT) Diagnostic Software User's Manual*, listed in Appendix A, *Related Documentation*, The *User's Manual* explains how to install and use MBIT.

This *Test Reference Guide* supports both the board level version of MBIT (PN: MBIT-BRD-51XX) and the system level version of MBIT (PN: MBIT-SYS-51XX). Refer to Chapter 1, *MBIT Overview* of this manual for a description of each version.

This guide is intended for use by software programmers or individuals with experience in the C programming language.

For easy use, this guide is divided into chapters based on the supported devices of the MBIT diagnostics.

Overview of Contents

This manual is divided into the following chapters and appendices:

Chapter 1, *Introduction*, provides an overview of the two MBIT versions, a list of the supported devices and subtests, the subtest default values, along with each subtest's attributes.

Chapter 2, *CPU Tests*, provides descriptions and requirements for the CPU tests.

Chapter 3, *L2 Cache Tests*, provides descriptions and requirements for the L2 cache tests.

Chapter 4, *System Memory Controller Test*, provides a description and requirements for the system memory controller (SMC) test.

Chapter 5, *PCI Host Bridge Test*, provides a description and requirements for the PCI host bridge (PHB) test.

Chapter 6, *Multiprocessor and ISA Interrupt Controller Tests*, provides descriptions and requirements for the multiprocessor and ISA interrupt controller tests.

Chapter 7, *ECC Memory Tests*, provides descriptions and requirements for the ECC memory tests.

Chapter 8, *Serial EEPROM Tests*, provides descriptions and requirements for the serial EEPROM tests.

Chapter 9, *NVRAM Test*, provides a description and requirements for the NVRAM test.

Chapter 10, *Real Time Clock Tests*, provides descriptions and requirements for the real time clock (RTC) tests.

Chapter 11, *UART Tests*, provides descriptions and requirements for the UART tests.

Chapter 12, *VME Bridge Tests*, provides descriptions and requirements for the VME bridge tests.

Chapter 13, *Ethernet Tests*, provides descriptions and requirements for the Ethernet tests.

Chapter 14, *System I/O Controller Test*, provides a description and requirements for the system I/O controller test.

Chapter 15, *Parallel Device Tests*, provides descriptions and requirements for the parallel device tests.

Chapter 16, *SCSI Device Tests*, provides descriptions and requirements for the SCSI device tests.

Chapter 17, *Flash Memory Tests*, provides descriptions and requirements for the Flash memory tests.

Chapter 18, *DS1621 Thermometer Tests*, provides descriptions and requirements for the DS1621 thermometer tests.

Appendix A, *Related Documentation*, provides a list of related documentation for the MBIT software.

Comments and Suggestions

Motorola welcomes and appreciates your comments on its documentation. We want to know what you think about our manuals and how we can make them better. Mail comments to:

Motorola Computer Group
Reader Comments DW164
2900 S. Diablo Way
Tempe, Arizona 85282

You can also submit comments to the following e-mail address: reader-comments@mcg.mot.com

In all your correspondence, please list your name, position, and company. Be sure to include the title and part number of the manual and tell how you used it. Then tell us your feelings about its strengths and weaknesses and any recommendations for improvements.

Conventions Used in This Manual

The following typographical conventions are used in this document:

bold

is used for user input that you type just as it appears; it is also used for commands, options and arguments to commands, and names of programs, directories and files.

italic

is used for names of variables to which you assign values, for function parameters, and for structure names and fields. Italic is also used for comments in screen displays and examples, and to introduce new terms.

`courier`

is used for system output (for example, screen displays, reports), examples, and system prompts.

<Enter>, **<Return>** or **<CR>**

represents the carriage return or Enter key.

Ctrl

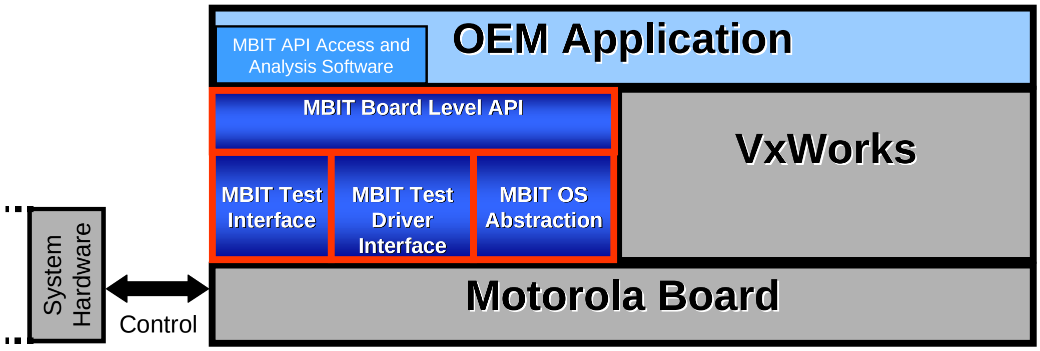
represents the Control key. Execute control characters by pressing the Ctrl key and the letter simultaneously, for example, Ctrl-d.

Introduction 1

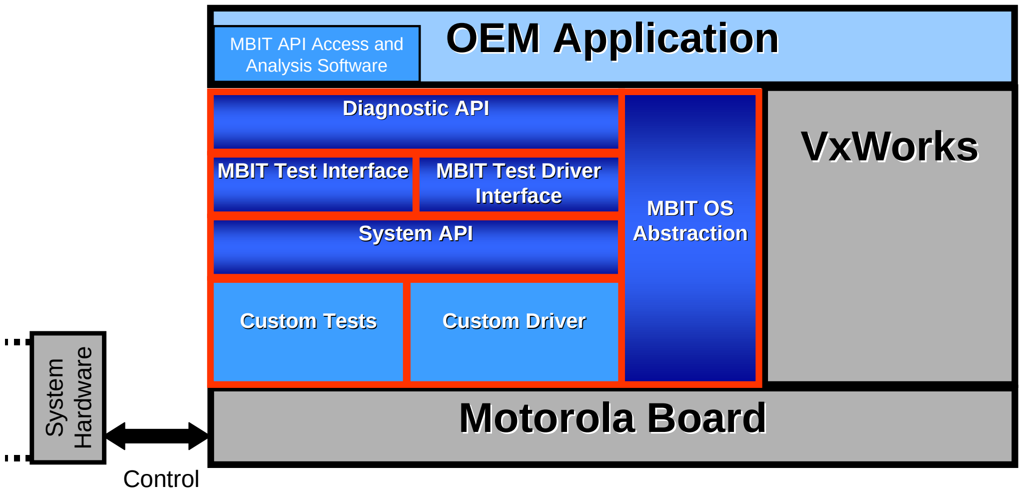
Introduction

MBIT is an off-the-shelf software infrastructure designed to verify the correct operation of Motorola Computer Group hardware. MBIT is available in two versions—board level MBIT and system level MBIT.

❑ Board level MBIT (PN:MBIT-BRD-51XX)—a comprehensive diagnostic software package designed to verify the correct operation of board mounted logical devices. All tests can execute at boot-up and selected tests can run continuously in the background of user applications. An application programming interface (API) is included to provide access to test results and to control the operation of device tests.



❑ System level MBIT (PN: MBIT-SYS-51XX)—includes all of the functionality and API function calls of the board level version and enables system-wide testing. It provides a framework and additional API function calls to support the inclusion of software designed to test custom hardware and/or system components.



This chapter contains the following sections of information:

Supported Devices and Subtests on page 1-3

Subtest Default Values on page 1-10

Subtest Attributes on page 1-14

Supported Devices and Subtests

Each supported device has a set of associated subtests. The table below lists each device with its associated subtest(s).

Table 1-1. Supported Devices and Subtests

Supported Device	Device String	Associated Subtest(s)	Subtest String
Processor	BIT_PROCESSOR	Integer arithmetic test	BIT_CPU_INT_ARITHMETIC
		Integer rotate/shift test	BIT_CPU_INT_ROTATE_SHIFT
		Integer load/store test	BIT_CPU_INT_LOAD_STORE
		Integer load/store multiple test	BIT_CPU_INT_LOAD_STORE_M
		Integer load/store string test	BIT_CPU_INT_LOAD_STORE_S
		Integer load/store byte reverse test	BIT_CPU_INT_LOAD_STORE_BR
		Integer compare and logical test	BIT_CPU_INT_LOGICAL
		Floating point arithmetic test	BIT_CPU_FLT_ARITHMETIC
		Floating point multiply-add/subtract test	BIT_CPU_FLT_MULTIPLY_ADD
		Floating point rounding/conversion test	BIT_CPU_FLT_ROUND_CONVERT
		Floating point load/store/move test	BIT_CPU_FLT_LOAD_STORE_MOVE
		Condition register logical test	BIT_CPU_CONDITION_REG

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
L2 Cache	BIT_L2_CACHE	Cache flush test	BIT_L2_CACHE_FLUSH
		Cache invalidate test	BIT_L2_CACHE_INV
		Cache lock test	BIT_L2_CACHE_LOCK
		Cache pattern test	BIT_L2_CACHE_PATTERN
		Cache size test	BIT_L2_CACHE_SIZE
		Cache write-back test	BIT_L2_CACHE_WRITEBACK
		Cache write-through test	BIT_L2_CACHE_WRITETHRU
System Memory Controller	BIT_MEMORY_ CONTROLLER	System Memory controller device visibility test	BIT_SMC_DEVICE_VISIBILITY
PCI Bus Bridge	BIT_LOCAL_BUS _TO_PCI_BRIDGE	PCI host bridge device visibility test	BIT_PHB_DEVICE_VISIBILITY
Interrupt Controller	BIT_INTERRUPT_ CONTROLLER	MPIC interrupt test	BIT_MPIC_INTERRUPTS
		ISA interrupt test	BIT_ISA_INTERRUPTS
ECC Memory	BIT_ECC_SDRAM	ECC single-bit error insertion test	BIT_ECC_SBIT_ERROR_INSERTION
		ECC multi-bit error insertion test	BIT_ECC_MBIT_ERROR_INSERTION
		RAM bit walk test	BIT_RAM_BIT_WALK
		RAM ones complement test	BIT_RAM_ONES_COMPLEMENT
		RAM patterns test	BIT_RAM_PATTERNS
		RAM address permutation test	BIT_RAM_ADDR_PERMUTATIONS
Serial EEPROM 1	BIT_SERIAL_ ROM1	VPD verify test	BIT_SROM_VPD_VERIFY
Serial EEPROM 2	BIT_SERIAL_ ROM2	VPD verify test	BIT_SROM_VPD_VERIFY
Serial EEPROM 3	BIT_SERIAL_ ROM3	SPD verify test	BIT_SROM_SPD_VERIFY

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
Serial EEPROM 4	BIT_SERIAL_ ROM4	SPD verify test	BIT_SROM_SPD_VERIFY
Serial EEPROM 5	BIT_SERIAL_ ROM5	SPD verify test	BIT_SROM_SPD_VERIFY
Serial EEPROM 6	BIT_SERIAL_ ROM6	User configuration data read test	BIT_SROM_USR_DATA_READ
Non-volatile RAM	BIT_ NONVOLATILE_ RAM1	NVRAM predefined memory test	BIT_NVRAM_PATTERNS
Real Time Clock	BIT_REAL_TIME_ CLOCK	Real time clock battery test	BIT_RTC_BATTERY
		Real time clock alarm test	BIT_RTC_ALARM
		Real time clock test	BIT_RTC_CLOCK
		Real time clock set test	BIT_RTC_SET_CLOCK
		Real time clock accuracy test	BIT_RTC_CLOCK_ACCURACY
		Watchdog timer test	BIT_RTC_WATCHDOG
Serial Port 1	BIT_ASYNC_ SERIAL_ DEVICE1	UART register test	BIT_SERIAL_REGISTER
		UART baud rate test	BIT_SERIAL_BAUD_RATE
		UART internal loopback polled mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ POLL
		UART internal loopback interrupt mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ INT
		UART external loopback with modem controls test	BIT_SERIAL_EXTERNAL_LOOPBACK

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
Serial Port 2	BIT_ASYNC_ SERIAL_ DEVICE2	UART register test	BIT_SERIAL_REGISTER
		UART baud rate test	BIT_SERIAL_BAUD_RATE
		UART internal loopback polled mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ POLL
		UART internal loopback interrupt mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ INT
		UART external loopback with modem controls test	BIT_SERIAL_EXTERNAL_LOOPBACK
Serial Port 3	BIT_ASYNC_ SERIAL_ DEVICE3	UART register test	BIT_SERIAL_REGISTER
		UART baud rate test	BIT_SERIAL_BAUD_RATE
		UART internal loopback polled mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ POLL
		UART external loopback with modem controls test	BIT_SERIAL_EXTERNAL_LOOPBACK
Serial Port 4	BIT_ASYNC_ SERIAL_ DEVICE4	UART register test	BIT_SERIAL_REGISTER
		UART baud rate test	BIT_SERIAL_BAUD_RATE
		UART internal loopback polled mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ POLL
		UART external loopback with modem controls test	BIT_SERIAL_EXTERNAL_LOOPBACK

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
Serial Port 5	BIT_ASYNC_ SERIAL_ DEVICE5	UART register test	BIT_SERIAL_REGISTER
		UART baud rate test	BIT_SERIAL_BAUD_RATE
		UART internal loopback polled mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ POLL
		UART internal loopback interrupt mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ INT
		UART internal loopback DMA mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ DMA
		UART external loopback with modem controls test	BIT_SERIAL_EXTERNAL_LOOPBACK
Serial Port 6	BIT_ASYNC_ SERIAL_ DEVICE6	UART register test	BIT_SERIAL_REGISTER
		UART baud rate test	BIT_SERIAL_BAUD_RATE
		UART internal loopback polled mode test	BIT_SERIAL_INTERNAL_ LOOPBACK_POLL
		UART internal loopback interrupt mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ INT
		UART internal loopback DMA mode test	BIT_SERIAL_INTERNAL_LOOPBACK_ DMA
		UART external loopback with modem controls test	BIT_SERIAL_EXTERNAL_LOOPBACK

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
VME Bus Bridge	BIT_PCI_TO_ VME_BRIDGE	VME bridge register read/write test	BIT_VME_REGISTER
		VME general-purpose target I/O test	BIT_VME_RW_TARGET
		VME short target I/O test	BIT_VME_SHORT_IO
		VME standard target I/O test	BIT_VME_STANDARD_IO
		VME extended target I/O test	BIT_VME_EXTENDED_IO
		VME CR/CSR visibility test	BIT_VME_CSR
		VME DMA (extended I/O) target test	BIT_VME_DMA
		VME bridge location monitor test	BIT_VME_LOCMON
Ethernet 1	BIT_ETHERNET_ DEVICE1	Serial EEPROM device verify test	BIT_ETHERNET_ROM_VERIFY
		Register test	BIT_ETHERNET_REGISTER
		Internal loopback test	BIT_ETHERNET_INTERNAL_ LOOPBACK
		External loopback test	BIT_ETHERNET_EXTERNAL_ LOOPBACK
		Serial EEPROM device accessibility test	BIT_ETHERNET_ROM_ACCESS
		Register accessibility test	BIT_ETHERNET_REGISTER_ACCESS

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
Ethernet 2	BIT_ETHERNET_ DEVICE2	Serial EEPROM device verify test	BIT_ETHERNET_ROM_VERIFY
		Register test	BIT_ETHERNET_REGISTER
		Internal loopback test	BIT_ETHERNET_INTERNAL_ LOOPBACK
		External loopback test	BIT_ETHERNET_EXTERNAL_ LOOPBACK
		Serial EEPROM device accessibility test	BIT_ETHERNET_ROM_ACCESS
		Register accessibility test	BIT_ETHERNET_REGISTER_ACCESS
ISA Bus Bridge	BIT_PCI_TO_ISA_ BRIDGE	ISA host bridge device visibility test	BIT_ISA_DEVICE_VISIBILITY
Parallel Port	BIT_PARALLEL_ DEVICE1	Parallel port register test	BIT_PARALLEL_REGISTER
		Parallel port FIFO test	BIT_PARALLEL_FIFO
SCSI Bus Controller	BIT_SCSI_ DEVICE1	SCSI SCRIPTS RAM test	BIT_SCSI_RAM
		SCSI bridging fault test	BIT_SCSI_LOCAL_DATA
		SCSI target arbitration test	BIT_SCSI_ID
		SCSI parity detection test	BIT_SCSI_PARITY
		SCSI illegal instruction detection test	BIT_SCSI_INSTRUCTIONS
		SCSI internal loopback test	BIT_SCSI_INTERNAL_LOOPBACK
Flash Bank A	BIT_FLASH1	Flash stuck bit test	BIT_FLASH_STUCK
		Flash float bit test	BIT_FLASH_FLOAT
Flash Bank B	BIT_FLASH2	Flash stuck bit test	BIT_FLASH_STUCK
		Flash float bit test	BIT_FLASH_FLOAT

Table 1-1. Supported Devices and Subtests (continued)

Supported Device	Device String	Associated Subtest(s)	Subtest String
Digital Thermometer	BIT_DIGITAL_ THERMOMETER	Read temperature test	BIT_THERMOMETER_READ_TEMP
		Access TH command test	BIT_THERMOMETER_ACCESS_TH
		Access TL command test	BIT_THERMOMETER_ACCESS_TL
		Access configuration command test	BIT_THERMOMETER_ACCESS_ CONFIG
		Read counter slope test	BIT_THERMOMETER_READ_ COUNTER_SLOPE
		Tout test	BIT_THERMOMETER_ALARM_TEST

Subtest Default Values

The following table contains the iteration, duration, and control default subtest values for each supported subtest.

Table 1-2. Subtest Default Values

Supported Device	Associated Subtest(s)	Iteration	Duration	Control
CPU	All CPU tests	1	1000	HALT_ON_ERROR
L2 Cache	Cache flush test	1	5000	HALT_ON_ERROR
	Cache invalidate test	1	5000	HALT_ON_ERROR
	Cache lock test	1	5000	HALT_ON_ERROR
	Cache pattern test	1	15000	HALT_ON_ERROR
	Cache size test	1	5000	HALT_ON_ERROR
	Cache write-back test	1	5000	HALT_ON_ERROR
	Cache write-through test	1	5000	HALT_ON_ERROR
System Memory Controller	System memory controller device visibility test	1	1000	RUN_TILL_ COMPLETION

Table 1-2. Subtest Default Values (continued)

Supported Device	Associated Subtest(s)	Iteration	Duration	Control
PCI Bus Bridge	PCI host bridge device visibility test	1	1000	RUN_TILL_ COMPLETION
Interrupt Controller	MPIC interrupt test	1	1000	HALT_ON_ERROR
	ISA interrupt test	1	1000	HALT_ON_ERROR
ECC Memory	ECC single-bit error insertion test	1	1000	HALT_ON_ERROR
	ECC multi-bit error insertion test	1	1000	HALT_ON_ERROR
	RAM bit walk test	1	50000	HALT_ON_ERROR
	RAM ones complement test	1	100000	HALT_ON_ERROR
	RAM patterns test	1	200000	HALT_ON_ERROR
	RAM address permutation test	1	25000	HALT_ON_ERROR
Serial EEPROM	VPD verify test	1	1000	HALT_ON_ERROR
	SPD verify test	1	5000	HALT_ON_ERROR
	User configuration data read test	1	1000	HALT_ON_ERROR
Non-volatile RAM	NVRAM predefined memory test	1	1000	HALT_ON_ERROR
Real-time clock	Real time clock battery test	1	1000	HALT_ON_ERROR
	Real time clock alarm test	1	3000	HALT_ON_ERROR
	Real time clock test	1	3000	HALT_ON_ERROR
	Real time clock set test	1	3000	HALT_ON_ERROR
	Real time clock accuracy test	1	35000	HALT_ON_ERROR
	Watchdog timer test	1	3000	HALT_ON_ERROR

Table 1-2. Subtest Default Values (continued)

Supported Device	Associated Subtest(s)	Iteration	Duration	Control
Serial Ports	UART register test	1	1000	HALT_ON_ERROR
	UART baud rate test	1	1000	HALT_ON_ERROR
	UART internal loopback polled mode test	1	1000	HALT_ON_ERROR
	UART internal loopback interrupt mode test	1	1000	HALT_ON_ERROR
	UART internal loopback DMA mode test	1	1000	HALT_ON_ERROR
	UART external loopback with modem controls test	1	1000	HALT_ON_ERROR
VME Bus Bridge	VME bridge register read/write test	1	1000	HALT_ON_ERROR
	VME general-purpose target I/O test	1	6000	HALT_ON_ERROR
	VME short target I/O test	1	6000	HALT_ON_ERROR
	VME standard target I/O test	1	6000	HALT_ON_ERROR
	VME extended target I/O test	1	6000	HALT_ON_ERROR
	VME CR/CSR visibility test	1	6000	HALT_ON_ERROR
	VME DMA (extended I/O) target test	1	6000	HALT_ON_ERROR
	VME bridge location monitor test	1	1000	HALT_ON_ERROR
Ethernet	Serial EEPROM device verify test	1	1000	HALT_ON_ERROR
	Register test	1	1000	HALT_ON_ERROR
	Internal loopback test	1	5000	HALT_ON_ERROR
	External loopback test	1	5000	HALT_ON_ERROR
	Serial EEPROM device accessibility test	1	1000	HALT_ON_ERROR
	Register accessibility test	1	1000	HALT_ON_ERROR

Table 1-2. Subtest Default Values (continued)

Supported Device	Associated Subtest(s)	Iteration	Duration	Control
ISA Bus Bridge	ISA host bridge device visibility test	1	1000	HALT_ON_ERROR
Parallel Port	Parallel port register test	1	1000	HALT_ON_ERROR
	Parallel port FIFO test	1	1000	HALT_ON_ERROR
SCSI Bus Controller	SCSI SCRIPTS RAM test	1	1000	HALT_ON_ERROR
	SCSI bridging fault test	1	2000	HALT_ON_ERROR
	SCSI target arbitration test	1	3500	HALT_ON_ERROR
	SCSI parity detection test	1	1000	HALT_ON_ERROR
	SCSI illegal instruction detection test	1	1000	HALT_ON_ERROR
	SCSI internal loopback test	1	4000	HALT_ON_ERROR
Flash	Flash stuck bit test	1	5000	HALT_ON_ERROR
	Flash float bit test	1	10000	HALT_ON_ERROR
Digital Thermometer	Read temperature test	1	1000	HALT_ON_ERROR
	Access TH command test	1	1000	HALT_ON_ERROR
	Access TL command test	1	1000	HALT_ON_ERROR
	Access configuration command test	1	1000	HALT_ON_ERROR
	Read counter slope test	1	1000	HALT_ON_ERROR
	Tout test	1	2000	HALT_ON_ERROR

Subtest Attributes

During the execution of any MBIT subtest, access to the device under test is not allowed. Table 1-3 describes additional restrictions and attributes for each supported subtest. The following bullets describe the table heading of Table 1-3.

- Subtest – The name of the subtest being described.
- Latency Issues – A subtest contains protected critical regions, which may induce latency and prevent or limit outside task execution (that is, **bitIntLock()**, **intLock()**, **taskLock()**).
- Abortable – A subtest may be immediately aborted. If the subtest is indicated as *not* abortable, then the subtest contains latency issues that may prevent the subtest from aborting.
- Execution Time (ms) – The average subtest execution time in milliseconds.
- Driver Needed – A subtest requires the operating system (OS) supplied driver.
- Driver Allowed – A subtest will run successfully with the OS supplied driver started.
- External Cables – A subtest requires additional off-board setup (that is, serial loopback cables, Ethernet loopback cables, VME slave board, etc.).

Refer to the appropriate chapter in this manual for more information regarding a device and its particular subtest(s).

Table 1-3. Subtest Attributes

Subtest String	Latency Issues**	Abortable	Execution Time (ms)*	Driver Needed	Driver Allowed	External Cables
BIT_CPU_INT_ ARITHMETIC	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_INT_ ROTATE_SHIFT	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_INT_LOAD_ STORE	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_INT_LOAD_ STORE_M	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_INT_LOAD_ STORE_S	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_INT_LOAD_ STORE_BR	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_INT_ LOGICAL	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_FLT_ ARITHMETIC	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_FLT_ MULTIPLY_ADD	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_FLT_ ROUND_CONVERT	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_FLT_LOAD_ STORE_MOVE	none	yes	< 0	n/a	n/a	n/a
BIT_CPU_ CONDITION_REG	none	yes	< 0	n/a	n/a	n/a
BIT_L2_CACHE_ FLUSH	bitIntLock	no	33	n/a	n/a	n/a
BIT_L2_CACHE_INV	bitIntLock	no	33	n/a	n/a	n/a
BIT_L2_CACHE_LOCK	bitIntLock	no	33	n/a	n/a	n/a
BIT_L2_CACHE_ PATTERN	bitIntLock	no	33	n/a	n/a	n/a

Table 1-3. Subtest Attributes (continued)

Subtest String	Latency Issues**	Abortable	Execution Time (ms)*	Driver Needed	Driver Allowed	External Cables
BIT_L2_CACHE_SIZE	bitIntLock	no	33	n/a	n/a	n/a
BIT_L2_CACHE_ WRITEBACK	bitIntLock	no	33	n/a	n/a	n/a
BIT_L2_CACHE_ WRITETHRU	bitIntLock	no	33	n/a	n/a	n/a
BIT_SMC_DEVICE_ VISIBILITY	bitIntLock	yes	166	yes (serial device 1, serial device 2)	yes	no
BIT_PHB_DEVICE_ VISIBILITY	bitIntLock	yes	16	yes (serial device 3, serial device 4)	yes	no
BIT_ECC_SBIT_ ERROR_INSERTION	bitIntLock	yes	33	n/a	n/a	n/a
BIT_ECC_MBIT_ ERROR_INSERTION	bitIntLock	yes	33	n/a	n/a	n/a
BIT_RAM_BIT_WALK	bitIntLock	yes	3,367	n/a	n/a	n/a
BIT_RAM_ONES_ COMPLEMENT	bitIntLock	yes	6,734	n/a	n/a	n/a
BIT_RAM_ PATTERNS	bitIntLock	yes	15,133	n/a	n/a	n/a
BIT_RAM_ADDR_ PERMUTATIONS	bitIntLock	yes	2,684	n/a	n/a	n/a
BIT_SROM_VPD_ VERIFY	none	yes	266	n/a	n/a	n/a
BIT_SROM_USR_ DATA_READ	none	yes	266	n/a	n/a	n/a
BIT_SROM_SPD_ VERIFY	none	yes	266	n/a	n/a	n/a
BIT_NVRAM_ PATTERNS	bitIntLock	yes	183	no	yes	n/a

Table 1-3. Subtest Attributes (continued)

Subtest String	Latency Issues**	Abortable	Execution Time (ms)*	Driver Needed	Driver Allowed	External Cables
BIT_RTC_BATTERY	bitIntLock	yes	< 0	no	no	n/a
BIT_RTC_ALARM	bitIntLock	yes	33,334	no	no	n/a
BIT_RTC_CLOCK	bitIntLock	yes	33,334	no	no	n/a
BIT_RTC_SET_CLOCK	bitIntLock	yes	< 0	no	no	n/a
BIT_RTC_CLOCK_ ACCURACY	bitIntLock	yes	33,333	no	no	n/a
BIT_RTC_ WATCHDOG	bitIntLock	yes	3,301	no	no	n/a
BIT_SERIAL_ REGISTER	intLock (z85230), bitIntLock (z85230, tl16c550)	yes	251	no	yes, no (serial device 5, serial device 6)	no
BIT_SERIAL_BAUD_ RATE	intLock (z85230), bitIntLock (z85230, tl16c550)	yes	749	no	yes, no (serial device 5, serial device 6)	no
BIT_SERIAL_ INTERNAL_ LOOPBACK_POLL	intLock (z85230), bitIntLock (z85230, tl16c550)	yes	466	no	yes, no (serial device 5, serial device 6)	no
BIT_SERIAL_ INTERNAL_ LOOPBACK_INT	intLock (z85230), bitIntLock (z85230, tl16c550)	yes	133	no	yes, no (serial device 5, serial device 6)	no
BIT_SERIAL_ INTERNAL_ LOOPBACK_DMA	intLock (z85230), bitIntLock (z85230, tl16c550)	yes	651	no	yes, no (serial device 5, serial device 6)	no

Table 1-3. Subtest Attributes (continued)

Subtest String	Latency Issues**	Abortable	Execution Time (ms)*	Driver Needed	Driver Allowed	External Cables
BIT_SERIAL_ EXTERNAL_ LOOPBACK	intLock (z85230), bitIntLock (z85230, tl16c550)	yes	551	no	yes, no (serial device 5, serial device 6)	yes
BIT_VME_REGISTER	bitIntLock	yes	2,000	no	yes	no
BIT_VME_RW_ TARGET	bitIntLock	yes	2,000	no	yes	slave board
BIT_VME_SHORT_IO	bitIntLock	yes	2,000	no	yes	slave board
BIT_VME_ STANDARD_IO	bitIntLock	yes	2,000	no	yes	slave board
BIT_VME_ EXTENDED_IO	bitIntLock	yes	2,000	no	yes	slave board
BIT_VME_CSR	bitIntLock	yes	2,000	no	yes	slave board
BIT_VME_DMA	bitIntLock	yes	2,484	no	yes	slave board
BIT_VME_LOCMON	bitIntLock	yes	2,000	no	yes	no
BIT_ETHERNET_ ROM_VERIFY	intLock (i82559)	yes	116	no	no	no
BIT_ETHERNET_ REGISTER	intLock (i82559)	yes	83	no	no	no
BIT_ETHERNET_ INTERNAL_ LOOPBACK	intLock (i82559)	yes	199	no	no	no
BIT_ETHERNET_ EXTERNAL_ LOOPBACK	intLock (i82559)	yes	201	no	no	yes
BIT_ETHERNET_ ROM_ACCESS	intLock (i82559)	yes	33	no	no	no
BIT_ETHERNET_ REGISTER_ ACCESS	intLock (i82559)	yes	33	no	no	no

Table 1-3. Subtest Attributes (continued)

Subtest String	Latency Issues**	Abortable	Execution Time (ms)*	Driver Needed	Driver Allowed	External Cables
BIT_ISA_DEVICE_ VISIBILITY	bitIntLock	yes	149	yes (serial device 3, serial device 4)	yes	no
BIT_PARALLEL_ REGISTER	bitIntLock	yes	< 0	no	no	no
BIT_PARALLEL_ FIFO	bitIntLock	yes	< 0	no	no	no
BIT_SCSI_RAM	bitIntLock	yes	16	no	no	no
BIT_SCSI_LOCAL_ DATA	bitIntLock	yes	33	no	no	no
BIT_SCSI_ID	bitIntLock	yes	16	no	no	no
BIT_SCSI_PARITY	bitIntLock	yes	66	no	no	no
BIT_SCSI_ INSTRUCTIONS	bitIntLock	yes	66	no	no	no
BIT_SCSI_INTERNAL_ LOOPBACK	bitIntLock	yes	99	no	no	no
BIT_MPIC_ INTERRUPTS	bitIntLock	yes	133	yes (serial device 1, serial device 2)	yes	no
BIT_ISA_ INTERRUPTS	bitIntLock	yes	< 0	yes (serial device 3, serial device 4)	yes	no
BIT_FLASH_STUCK	none	yes	< 0	n/a	n/a	n/a
BIT_FLASH_FLOAT	none	yes	1,416	n/a	n/a	n/a
BIT_THERMOMETER_ READ_TEMP	bitIntLock	yes	751	no	no	n/a
BIT_THERMOMETER_ ACCESS_TH	bitIntLock	yes	583	no	no	n/a

Table 1-3. Subtest Attributes (continued)

Subtest String	Latency Issues**	Abortable	Execution Time (ms)*	Driver Needed	Driver Allowed	External Cables
BIT_THERMOMETER_ ACCESS_TL	bitIntLock	yes	583	no	no	n/a
BIT_THERMOMETER_ ACCESS_CONFIG	bitIntLock	yes	533	no	no	n/a
BIT_THERMOMETER_ READ_COUNTER_ SLOPE	bitIntLock	yes	751	no	no	n/a
BIT_THERMOMETER_ ALARM_TEST	bitIntLock	yes	1,567	no	no	n/a

Notes

1. *Average execution time is based on tests run using default parameters. The tests were executed on a MVME5100 with 64MB RAM. The memory test execution times will vary depending on the amount of memory available for testing.
2. **These are routines used to protect critical sections and may induce some latency in responding to interrupts.

CPU Tests 2

This chapter provides descriptions and requirements for the following CPU tests:

CPU Tests

All of the CPU tests in this chapter have the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Integer Arithmetic Test

[**BIT_CPU_INT_ARITHMETIC**]

This test verifies correct operation of the integer arithmetic instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test tests the 24 **add**, six **sub**, four **mul**, two **div**, and one **neg** instructions.

Each **add** instruction performs an addition and the result is compared to the true value. A test fails if the result and true values are not equal. Each **add** instruction is also tested for varying overflow and carry conditions. A test fails if the condition register (CR)/integer and exception register (XER) are set incorrectly.

Each of the **sub**, **mul**, **div**, and **neg** instructions are tested for a correct result.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_ALU_FAULT—CPU arithmetic logic unit fault has occurred

BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Integer Rotate/Shift Test

[**BIT_CPU_INT_ROTATE_SHIFT**]

This test verifies correct operation of the integer rotate/shift instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test saves the link register and all registers used within the routine. It then performs seven rotate and shift instruction tests and several variants of each of the instructions as described below:

Rotate Left Word Immediate then Mask Insert (rlwimi)

This test loads register **r**14 with a binary test pattern and register **r**15 with a binary pattern representing the expected result of this instruction. This instruction is executed and the result in **r**16 is compared with the expected result in **r**15. If the results are not equal, the routine returns to the caller with a failed status. However, if the result is as expected, the next test is performed.

This test is in three parts, test A through test A2.

test A	This test rotates five bits left and uses all ones for a mask in bits 0...31. The result is a simple rotate left five bits.
test A1	This test inserts the five most significant bits from the source register into the least significant five bits of the destination register, using the simplified mnemonic **inslwi** **r**A,**r**S,*n*,*b*. *n* is the number of bits to insert and *b* is the starting bit position of the destination register where the bits are inserted.
test A2	This test inserts the five least significant bits of the source register into the five most significant bits of the destination register, using the simplified mnemonic **insrwi** **r**A,**r**S,*n*,*b*.

Note Both test A1 and test A2 insert either the most significant *n* bits of the source register or the least significant *n* bits of the source register into an *n* bit field of the destination register, starting at the bit position defined by the instructions *b* operand. The remaining bits of the destination register are unchanged.

Rotate Left Word Immediate then AND with Mask (rlwinm)

This test is in ten parts, test B through test B9. Test B uses the normal syntax for the instruction, while tests B1 through B9 use the & (ampersand) simplified mnemonics for the variants of the instruction.

test B	This test is a simple rotate left *n* bits using the instruction **rlwinm r**A,**r**S,SH,MB,ME with the mask set to all ones in bits 0...31.
test B1	This test extracts *n* bits from the source register and places them left justified into the destination register, clearing the remaining bits of the destination register.
test B2	This test extracts *n* bits from the source register and places them right justified into the destination register, clearing the remaining bits of the destination register.
test B3	This test is a simple rotate left *n* bits, which is a duplicate of test B, but uses the simplified mnemonic for the instruction.
test B4	This test is a simple rotate right *n* bits using the simplified mnemonic for the instruction.
test B5	This test shifts the source register left *n* bits, then places the result in the destination register, clearing the least significant five bits.
test B6	This test shifts the source register right *n* bits, then places the result in the destination register, clearing the *n* most significant bits.
test B7	This test extracts *n* bits from bit position *b* through 31 of the source register and places them into the same bit positions of the destination register, while clearing all high order bits that are the more significant bits above the indicated start bit position *b*.

test B8	This test extracts *n* bits from bit position 0 through *n* of the source register and places them into the same bit positions of the destination register, while clearing all low order bits that are the least significant bits below the & number of bits extracted.
test B9	This test clears *b* high order bits of the source register and places the remaining bits into the destination register shifted left by *n* bits, where *n* <= *b* < 32. The least significant bits of the destination register, corresponding to the *n* bits shifted, are filled with zeros.

Rotate Left Word then AND with Mask (rlwnm)

This test is in five parts, test C through test C4.

test C	This test is a simple rotate left and mask. The pattern in the source register is rotated left *n* bits as determined by the value in the least significant five bits of the shift count register. The result is placed into the destination register and masked with all ones beginning at bit MB and ending at bit ME. The instruction is coded using the simplified mnemonic **rotlw**, equivalent to **rlwnmr r**A,**r**S,**r**B,0,31.
test C1	This test extracts *n* bits from the source register beginning at bit MB through bit ME, places these bits left justified into the destination register, and clears the remaining bits in the destination register. The number of bits to extract is contained in the five least significant bits of the **r**B register.
test C2	This test extracts *n* bits from the source register and places them right justified into the destination register, clearing the remaining bits of the destination register.
test C3	This test is a simple rotate left *n* bits mask with all ones, where the rotate count *n* is contained in the least significant five bits of the **r**B register and the mask is defined by the MB through ME bits.
test C4	This test is a simple rotate right *n* bits mask with all ones, where the rotate count *n* is contained in the least significant five bits of the **r**B register and the mask is defined by the MB through ME bits.

Shift Left Word (slw)

This test is in two parts, test D and test D1.

test D	This test shifts a bit pattern, contained in the source register, left *n* bits and places the result in the destination register with zeros filling the least significant *n* bits of the destination register.
test D1	This test clears the register and the *n* least significant bits are lost. This test is performed with a negative value in the source register and compared to a negative result.

Shift Right Algebraic Word (sraw)

This test is in three parts, test E through E2.

test E	This test is similar to test D1, except that it begins with a positive value in the source register and compares the result to a positive result.
test E1	This test is performed with the value of register **r**B set to **32**, which sets the content of the destination register to a value of **–1** when the source register contains a negative value.
test E2	This test is performed with the value of register **r**B set to **32**, which sets the content of the destination register to **0** when the source register contains a positive value.

Shift Right Algebraic Word Immediate (srawi)

This test is in two parts, test F and test F1.

test F	This test performs the same test as test E1, except the shift count is an immediate value rather than a register value.
test F1	This test performs the same test as test E2, except the shift count is an immediate value rather than a register value.

Shift Right Word (srw)

This test is in two parts, test G and test G1.

test G	This test shifts a bit pattern, contained in the source register, right *n* bits and places the result in the destination register with zeros, filling the most significant *n* bits of the destination register.
test G1	This test clears the destination register to all zeros using the **srw** instruction with a value of **32** in register **r**B. This test uses the **srw** instruction to record the condition of the result, which if correct, sets the EQ bit in the condition register CR0 field.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_MCIU_FAULT—CPU multi-cycle integer unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Integer Load/Store Test

[**BIT_CPU_INT_LOAD_STORE**]

This test verifies correct operation of the integer load/store instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test contains routines that test the 16 integer load and 12 integer store instructions. For each instruction, the appropriate byte, half word, or word is stored to memory. The memory location is then loaded into a general-purpose register (GPR) and compared to the value that was stored. Instructions that include an update are also tested for correctness. This test also tests the two synchronization instructions, **Load Word and Reserve Indexed (lwarx)** and **Store Word Conditional Indexed (stwcx)**.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_LSU_FAULT—CPU load/store unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Integer Load/Store Multiple Test

[**BIT_CPU_INT_LOAD_STORE_M**]

This test verifies correct operation of the integer load/store multiple instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test contains routines designed to test the **Load Multiple Word (lmw)** and **Store Multiple Word (stmw)** instructions.

Three registers, **r**29, **r**30, and **r**31, are initialized with three test words and stored in memory using the **stmw** instruction. The same registers are then cleared and the **lmw** instruction is executed to load the test words from memory. Each of the multiple registers are then individually compared to the initial test words that were stored.

A failure to compare correctly in any register results in an immediate return to the calling routine. Successful execution verifies that data can both be stored to and loaded from memory using these instructions.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported

BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_LSU_FAULT—CPU load/store unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Integer Load/Store String Test

[**BIT_CPU_INT_LOAD_STORE_S**]

This test verifies correct operation of the integer load/store string instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test tests the four load string and store string instructions: **Load String Word Immediate (lswi)**, **Store String Word Immediate (stswi)**, **Load String Word Indexed (lswx)**, and **Store String Word Indexed (stswx)**.

A test string is loaded from memory into the GPR. The loaded string is then written back to a different memory area. The written string is then compared to the test string, one word at a time, until the string length is reached. If any comparison fails, the test fails.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_LSU_FAULT—CPU load/store unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Integer Load/Store Byte-Reverse Test

[**BIT_CPU_INT_LOAD_STORE_BR**]

This test verifies correct operation of the integer load/store byte-reverse instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test tests the four byte-reverse indexed instructions: **Load Half Word Byte-Reverse Indexed (lhbrx)**, **Store Half Word Byte-Reverse Indexed (sthbrx)**, **Load Word Byte-Reverse Indexed (lwbrx)**, and **Store Word Byte-Reverse Indexed (stwbrx)**.

For each instruction, the appropriate word/half word is loaded into a GPR and compared to its byte-reversed value. The GPR value is then stored in byte-reversed order into data storage. The data storage value is then loaded into a GPR and compared to the non-reversed value.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_LSU_FAULT—CPU load/store unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Integer Compare and Logical Test

[**BIT_CPU_INT_LOGICAL**]

This test verifies correct operation of the integer compare and logical instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test tests the four integer compare and 17 integer logical instructions. Using known values for the operands, each compare/logical instruction is executed and the result is compared to the correct value.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_SCIU_FAULT—CPU single-cycle integer unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Floating-Point Arithmetic Test

[**BIT_CPU_FLT_ARITHMETIC**]

This test verifies correct operation of the floating-point arithmetic instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

The following 11 routines are designed to test the floating-point arithmetic instructions.

Floating-Point Add Double (fadd)

This instruction is tested by adding two floating-point values and is checked according to the following equations: **A** + **B** = **C** and **A** = **C** – **B**, where **A** and **B** are constants. The value of **A**, after the subtraction, is compared to the initial value of **A**, and if equal, the addition is considered successful. A failure to compare correctly results in an immediate return to the calling method. Successful execution allows the routine to drop through to the next test until all instructions have been executed.

Floating-Point Add Single (fadds)

This instruction is tested as described in *Floating-Point Add Double (fadd)*, except for the single precision mode.

Floating-Point Double Precision Divide (fdiv)

This instruction is tested according to the following equation: **C** = **A**/**B** and checked by **A** = **B*****C**. The initial value of **A** and the resultant value are compared, and if equal, the test is considered successful.

Floating-Point Single Precision Divide (fdivs)

This instruction is tested as described in *Floating-Point Double Precision Divide (fdiv)*, except for the single precision mode.

Floating-Point Double Precision Multiply (fmul)

This instruction is tested according to the following equation: **C** = **A*****B** and checked by **A** = **C**/**B**. The initial value of **A** and the resultant are compared, and if equal, the test is considered successful.

Floating-Point Single Precision Multiply (fmuls)

This instruction is tested as described in *Floating-Point Double Precision Multiply (fmul)*, except for the single precision mode.

Floating-Point Reciprocal Estimate (fres)

This instruction is tested by executing the instruction to obtain the reciprocal estimate of a constant **A**. The actual value of 1/**A** (calculated in the conventional manner) is then determined and the result of the reciprocal estimate is checked to determine if it falls within the stated precision of the instruction of one part in 256. This check is performed using the following equation: precision = ((estimate of **A**/1/**A**) – 1.0). See page 8-88 of the *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors* manual, listed in Appendix A, *Related Documentation*, for this equation. If the precision is less than or equal to one part in 256, the result is correct.

Floating-Point Reciprocal Square Root Estimate (frsqrte)

This instruction is tested as described in *Floating-Point Reciprocal Estimate (fres)*. However, for this instruction the test for success uses the following equation: precision = ((estimate of sqrt **A**/1/sqrt **A**) – 1.0). See page 8-91 of the *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors* manual, listed in Appendix A, *Related Documentation*, for this equation. The results of the instruction are correct if the precision is less than or equal to one part in 32.

Floating-Point Double Subtract (fsub)

This instruction is tested by executing the instruction to obtain the difference between two values and then checked by addition. The execution of the **fsub** instruction is identical to that of **fadd**, except the contents of the operand that is being subtracted participates with its sign bit (bit 0) inverted.

Floating-Point Single Subtract (fsubs)

This instruction is tested as described in *Floating-Point Double Subtract (fsub)*, except for the single precision mode.

Floating-Point Select (fsel)

This instruction, which uses the syntax of **f**D,**f**A,**f**B,**f**C, selects and places either **f**B or **f**C in destination register **f**D, according to the signed value in **f**A. If **f**A is positive, **f**B is selected. Otherwise, if **f**A is negative, **f**C is selected.

To test the instruction, a positive value is assigned **f**A and the result of executing the instruction is checked to confirm that the value of **f**B was placed in **f**D. The value in **f**A is then negated and the instruction is executed again. Success is indicated if the value of **f**C is placed in **f**D.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_FPU_FAULT—CPU floating-point unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Floating-Point Multiply-Add/Subtract Test

[**BIT_CPU_FLT_MULTIPLY_ADD**]

This test verifies correct operation of the floating-point multiply-add/subtract instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

The following nine routines are designed to test the floating-point multiply-add/subtract instructions.

Floating Multiply-Add Double (fmadd)

This instruction is tested using constant test values to calculate **X** = (**A***B**) + **C**. The result, **X**, is then compared to a value calculated via the conventional multiply and add instructions. When both calculations result in the same value, the test is considered successful.

Floating Multiply-Add Single (fmadds)

This instruction is tested as described in *Floating Multiply-Add Double (fmadd)*, except the floating multiply-add single instruction is tested.

Floating Multiply-Subtract Double (fmsub)

This instruction is tested using constant test values to calculate **X** = (**A***B**) – **C**. The result, **X**, is then compared to a value calculated via the conventional multiply and then compared to a value calculated via the conventional multiply and subtract instructions. When both calculations result in the same value, the test is considered successful.

Floating Multiply-Subtract Single (fmsubs)

This instruction is tested as described in *Floating Multiply-Subtract Double (fmsub)*, except the floating multiply-subtract single instruction is tested.

Floating Negative Multiply-Add Double (fnmadd)

This instruction is tested using constant test values to calculate –**X** = (**A***B**) + **C**. The result, –**X**, is then compared to a value calculated via the conventional multiply and add instructions, followed by a floating-point negate instruction. When both calculations result in the same value, the test is considered successful.

Floating Negative Multiply-Add Single (fnmadds)

This instruction is tested as described in *Floating Negative Multiply-Add Double (fnmadd)*, except the floating negative multiply-add single instruction is tested.

Floating Negative Multiply-Subtract Double (fnsub)

This instruction is tested using constant test values to calculate –**X** = (**A***B**) + –**C**. The result, –**X**, is then compared to a value calculated via the conventional multiply and add instructions, followed by a floating-point negate instruction. When both calculations result in the same value, the test is considered successful.

Negative Floating Negative Multiply-Add Single (fnsubs)

This instruction is tested as described in *Floating Negative Multiply-Subtract Double (fnsub)*, except the negative floating negative multiply-add single instruction is tested.

Floating Multiply-Add Single with Rc = 1 (fmadds.)

This instruction is tested as described in *Floating Multiply-Add Double (fmadd)* to allow the condition code of the result to be recorded. However, in this case, the condition code is tested after executing the instruction. Test constants are used that result in an overflow condition and if the condition code reflects this condition, the test is considered successful.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_FPU_FAULT—CPU floating-point unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Floating-Point Rounding/Conversion Test

[**BIT_CPU_FLT_ROUND_CONVERT**]

This test verifies correct operation of the floating-point rounding/conversion instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

The following four routines are designed to test the floating-point rounding/conversion instructions.

Floating Convert to Integer Word (fctiw)

This instruction is tested by converting a constant floating-point test value to an integer and storing the result in memory. The rounding mode is set to round toward positive infinity before the instruction is executed. After executing the instruction, the stored result is loaded from memory and compared to an integer value that represents the expected result of the rounding and conversion. Success is indicated if the actual and expected results are the same.

Floating Convert to Integer Word with Round Toward Zero (fctiwz)

This instruction is tested as described in *Floating Convert to Integer Word (fctiw)*, except that the rounding mode is set by the instruction itself.

Floating Convert to Integer Word with RC = 1 (fctiw.)

This instruction is also tested as described in *Floating Convert to Integer Word (fctiw)*. However, the test value to be converted is made larger than the range of values that may be represented as a single precision value. In this instance, the converted value is declared invalid and bits 32–63 of the destination floating-point register (FPR) are set to 0x7fffffff. After executing the instruction, both of these conditions, invalid bit set and bits 32–63 = 0x7fffffff, are tested. Success is indicated by a valid compare and the invalid VX bit is set.

Floating-Point Round to Single (fprs)

This instruction is tested by first setting the rounding mode to round toward minus infinity. A test value is then loaded as a double precision value into an FPR and the value stored in memory as a single precision value. The double precision value is then converted to single precision value, using the **fprs** instruction. The result of the conversion is stored in memory as a single precision value. The value stored before executing the instruction and the value stored as a result of the instruction are then read back from memory and compared for equality. Success is indicated if both are the same. This procedure is necessary to test this instruction due to the manner in which single precision and double precision values are represented in memory and floating-point registers.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_FPU_FAULT—CPU floating-point unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Floating-Point Load/Store/Move Test

[**BIT_CPU_FLT_LOAD_STORE_MOVE**]

This test verifies correct operation of the floating-point load/store/move instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

The following seven routines are designed to test the floating-point load/store/move instructions.

The load and store instructions are complementary in that each load instruction has a similar store instruction. Therefore, the load and store instructions are tested as a pair where possible.

Load Floating-Point Double (lfd) and Store Floating-Point Double (stfd)

These instructions load a double precision constant value and store that value in a memory location. The stored value is then retrieved and compared to the initial value stored. If the loaded and stored values are equal, the tests performed correctly.

Load Floating-Point Double with Update (lfdu) and Store Floating-Point Double with Update (stfdu)

These instructions load the address of a memory location into a GPR and save that address in a second GPR. A data constant is then stored at the effective address (EA) indicated by one of the GPRs using the **stfdu** instruction. The EA in that register is updated by the store instruction and a **lfdu** instruction is then executed with a negative displacement. The operation of the **lfdu** instruction updates the EA again, except with a negative displacement, such that it should now be the same as the stored address. The addresses of this GPR and the GPR that holds the saved memory address are then compared. If both registers contain the same value, both the **stfdu** and **lfdu** instructions updated the address correctly.

Load Floating-Point Double with Update Indexed (lfdux) and Store Floating-Point Double with Update Indexed (stfdux)

These instructions are similar to *Load Floating-Point Double with Update (lfdu) and Store Floating-Point Double with Update (stfdu)*, except that the address indexed value is loaded into a GPR and the **stfdux** instruction is executed to store a value in memory and update and index the EA. The index is then made negative and the **lfdux** instruction is executed to retrieve the saved value and update and index the EA. Then, the EA is compared with the initial EA and when equal, both instructions have performed correctly.

Load Floating-Point Double with Index (lfdx) and Store Floating-Point Double with Index (stfdx)

These instructions test the **stfdx** and **lfdx** instructions. A constant value is stored in memory using the **stfdx** instruction and then, using the same index value, the stored double word is retrieved from memory using the **lfdx** instruction. The retrieved value is then compared with the value stored, and if equal, both instructions operated correctly.

Store Floating-Point as Integer Word Indexed (stfiwx)

This instruction loads a floating-point constant that is comprised of an integer part and a fractional part. The floating-point constant is then stored using the **stfiwx** instruction, which stores the value as an integer. The stored value is then retrieved from memory using the **Load Word and Zero (lwz)** instruction and compared with an integer word that represents the expected result. If the actual and expected results are the same, the instruction has operated correctly.

Load Floating-Point Single Indexed (lfsx), Store Floating-Point Single Indexed (stfsx), Load Floating-Point Single with Update Indexed (lfsux), Store Floating-Point Single with Update Indexed (stfsux), Load Floating-Point Single with Update (lfsu), Store Floating-Point Single with Update (stfsu), Load Floating-Point Single (lfs), and Store Floating-Point Single (stfs)

These instructions perform the same operations on single precision values as the corresponding double precision instructions described above. The only difference is the double versus single precision. These instructions are therefore, tested as described above for the double precision tests using single precision test operands.

Floating Negate (fneg), Floating Negative Absolute Value (fnabs), Floating Absolute Value (fabs), and Floating Move Register (fmr)

These instructions test the floating-point move instructions.

The **fneg** instruction is tested by loading a positive floating-point constant and then negating that value using the **fneg** instruction. The result of the negation is then algebraically added to the initial value and that result is compared to **0**. If the actual result is **0**, the instruction performed correctly.

The **fnabs** instruction is tested by performing the **fnabs** operation on the previously loaded positive test value and then comparing the result to the previously negated value obtained in the **fneg** instruction. If the two values are equal, the instruction operated correctly. This test is performed using an initial positive value because the instruction always sets the sign bit of the absolute value of the operand, regardless of the sign of the initial operand.

The **fabs** instruction is tested by using the instruction to obtain the absolute value of the previously negated value obtained in the **fneg** instruction. The result is compared to the initial positive operand and if equal, the instruction operated correctly.

The **fmr** instruction is tested by moving the content of one floating-point register to another and then comparing the content of the two. If both are the same, the instruction operated correctly. This instruction is also tested using the dot notation to record the resultant condition code. This second way of testing is executed similar to the first way and the condition code bits in the CR1 field are tested for any over or underflow exceptions. If there were none, the instruction operated correctly.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_FPU_FAULT—CPU floating-point unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

Condition Register Logical Test

[**BIT_CPU_CONDITION_REG**]

This test verifies correct operation of the condition register logical instructions for the Motorola PowerPC architecture-compatible processors.

Test Description

This test tests the nine condition register logical instructions. The condition register is loaded with a known value of 0xFFFFFFFF. Each instruction is executed using various bits/fields of the CR for operands. The specified destination CR bits are tested for the correct result. The nine instructions to be tested are described below.

Condition Register AND (crand)

The bit in the condition register specified by **A** is ANDed with the bit in the condition register specified by **B**. The result is placed into the condition register bit specified by **C**. If **C** = **1**, the test passes, otherwise the test fails.

Condition Register OR (cror)

The bit in the condition register specified by **A** is ORed with the bit in the condition register specified by **B**. The result is placed into the condition register bit specified by **C**. If **C** = **1**, the test passes, otherwise the test fails.

Condition Register XOR (crxor)

The bit in the condition register specified by **A** is XORed with the bit in the condition register specified by **B**. The result is placed into the condition register bit specified by **C**. If **C** = **0**, the test passes, otherwise the test fails.

Condition Register NAND (crnand)

The bit in the condition register specified by **A** is ANDed with the bit in the condition register specified by **B**. The complemented result is placed into the condition register bit specified by **C**. If **C** = **0**, the test passes, otherwise the test fails.

Condition Register NOR (crnor)

The bit in the condition register specified by **A** is ORed with the bit in the condition register specified by **B**. The complemented result is placed into the condition register bit specified by **C**. If **C** = **0**, the test passes, otherwise the test fails.

Condition Register Equivalent (creqv)

The bit in the condition register specified by **A** is XORed with the bit in the condition register specified by **B** and the complemented result is placed into the condition register bit specified by **C**. If **C** = **1**, the test passes, otherwise the test fails.

Condition Register AND with Complement (crandc)

The bit in the condition register specified by **A** is ANDed with the complement of the bit in the condition register specified by **B**. The complemented result is placed into the condition register bit specified by **C**. If **C** = **0**, the test passes, otherwise the test fails.

Condition Register OR with Complement (crorc)

The bit in the condition register specified by **A** is ORed with the complement of the condition register bit specified by **B**. The result is placed into the condition register bit specified by **C**. If **C** = **1**, the test passes, otherwise the test fails.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_CPU_BPU_FAULT—CPU branch processing unit fault has occurred
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

L2 Cache Tests 3

L2 Cache Tests

This chapter provides descriptions and requirements for the L2 cache tests. The references to the L2 cache utility methods are only available to the diagnostic developer and not the user. These methods are listed below.

bitL2CacheSizeGet()

bitL2CacheIsWritebackCapable()

bitL2CacheIsLockable()

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_L2_CACHE_FLUSH] [BIT_L2_CACHE_INV] [BIT_L2_CACHE_LOCK] [BIT_L2_CACHE_PATTERN] [BIT_L2_CACHE_SIZE] [BIT_L2_CACHE_WRITEBACK] [BIT_L2_CACHE_WRITETHRU]	tests/l2cache/ l2CacheTests.h	L2_CACHE_PARAMS

L2 Cache Flush, Invalidate, Lock, Pattern, Size, Write-Back, Write-Through Tests

[**BIT_L2_CACHE_FLUSH**], **BIT_**[**L2_CACHE_INV**], [**BIT_L2_CACHE_LOCK**], [**BIT_L2_CACHE_PATTERN**], [**BIT_L2_CACHE_SIZE**], [**BIT_L2_CACHE_WRITEBACK**], [**BIT_L2_CACHE_WRITETHRU**]
tests/l2CacheTests.h

The **tests/l2CacheTests.h** file defines the parameter structures and fields mentioned in all of the L2 cache tests.

Each of the L2 cache tests has the same general description and requirements. The individual tests only vary in the combination and order of various cache enables, disables, reads, writes, invalidates, etc.

❑ The L2 Cache Flush test [**BIT_L2_CACHE_FLUSH**] verifies correct operation of the L2 cache flush function. This test is only executed if the L2 cache supports write-back. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

❑ The L2 Cache Invalidate test [**BIT_L2_CACHE_INV**] verifies correct operation of the L2 cache invalidate function. This test is only executed if the L2 cache supports write-back. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

❑ The L2 Cache Lock test [**BIT_L2_CACHE_LOCK**] verifies correct operation of the L2 cache lock feature. This test is only executed if the L2 cache is lockable. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

- ❑ The L2 Cache Pattern test [**BIT_L2_CACHE_PATTERN**] writes a variety of data patterns to the L2 cache and verifies that the data exists in both the L2 cache and system memory. This test has the following default test values:

Iteration:	1
Duration:	15000
Control:	HALT_ON_ERROR

- ❑ The L2 Cache Size test [**BIT_L2_CACHE_SIZE**] verifies that the real size of the L2 cache is the same as the size indicated by hardware. The size indicated by hardware is retrieved using the MBIT utility function, **bitL2CacheSizeGet()**. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

- ❑ The L2 Cache Write-Back test [**BIT_L2_CACHE_WRITEBACK**] verifies the L2 cache operates properly in writeback mode. This test is only executed if the L2 cache supports writeback. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

❏ The L2 Cache Write-Through test [**BIT_L2_CACHE_WRITETHRU**] verifies the L2 cache operates properly in write-through mode. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

Test Description

Before invoking any L2 cache test, the MBIT cache attributes must be initialized by calling **bitSetCacheAttributes()**. MBIT sets cache attributes and capabilities based upon the processor type. MBIT performs this step during its startup initialization. If attribute initialization fails, the cache tests will not run.

Each test replaces the processor’s page table and BAT registers, as required to run the test. Replacing the page table involves allocating memory the size of the existing page table and aligning it to an address of that size. If this memory allocation fails, the test does not run.

Configuring the IBAT and DBAT registers involves setting each BAT to inhibit caching and then using an available DBAT to map the test buffer memory region. The single DBAT used to map the test buffer memory region is configured according to the test requirements as write-back or write-through with caching enabled. If there is no available DBAT for the test buffer memory region, the last DBAT is used. As a result, the memory region mapped by the last DBAT is inaccessible during the L2 cache test. When an unused DBAT is available, memory regions mapped by the other DBAT registers, the IBAT registers, and the page table remain accessible except that caching is inhibited for those regions.

Cache write-back support is ascertained using the MBIT utility function, **bitL2CacheIsWritebackCapable()**. Cache locking support is ascertained using the MBIT utility function, **bitL2CacheIsLockable()**.

Note: These tests (except **BIT_L2_CACHE_LOCK**) are supported on the Motorola MPC750 class, MPC755 class, MPC7400 and MPC7410 processors. The **BIT_L2_CACHE_LOCK** test is only supported on the MPC7400 and MPC7410 processors.

Affected Peripheral Devices

This test does not affect any peripheral devices, however, it does replace the page table, DBAT, and IBAT values during the test.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes the test-specific parameters by reference. The test parameter structure is defined in **tests/l2cache/l2CacheTests.h** (**L2_CACHE_PARAMS**).

All of the L2 cache tests have the same default parameter, *l2CacheSize* = **0**

The contents of the structure and its effects on the test is discussed below:

l2CacheSize

is the size in bytes of the L2 cache as indicated by hardware. The install routine sets this value using the MBIT utility function, **bitL2CacheSizeGet()**. The *l2CacheSize* value passed to the install routine is not authenticated and is overwritten by the install routine. If the value returned by **bitL2CacheSizeGet()** is **0**, the test does not run.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_RESOURCE_MGMT_FAULT—a memory management error occurred
BIT_INSTALL_TEST_FAILED—test installation failed

BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_OPERATION_IN_PROGRESS—a previous operation (test or abort) is in progress
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

System Memory Controller Test 4

System Memory Controller Test

This chapter provides a description and requirements for the system memory controller (SMC) test.

The table below highlights the test's string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_SMC_DEVICE_VISIBILITY]	tests/visibilityTests.h	VISIBILITY_PARAMS

System Memory Controller Device Visibility Test

[**BIT_SMC_DEVICE_VISIBILITY**]
tests/visibilityTests.h

The **tests/visibilityTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SMC_DEVICE_VISIBILITY**] test.

This test probes devices in a list to determine if the device is visible. The SMC and PCI host bridge (PHB) visibility tests use a common test routine and test mechanism.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	RUN_TILL_COMPLETION

Test Description

This test will use its default parameters by passing a **NULL** test parameter structure to the test routine. This test will ensure the visibility of devices on both sides of the **BIT_MEMORY_CONTROLLER**. A list of devices that will be tested are as follows:

BIT_MEMORY_CONTROLLER

BIT_ECC_SDRAM

BIT_ASYNC_SERIAL_DEVICE1

BIT_ASYNC_SERIAL_DEVICE2

BIT_FLASH1

BIT_FLASH2

BIT_SERIAL_ROM1

BIT_SERIAL_ROM2

BIT_SERIAL_ROM3

BIT_SERIAL_ROM6

A location is designated for visibility testing on each device. For locations designated as read-only or write-only, the routine does not verify that the read or write succeeded. The only situations that cause a read-only or write-only visibility test to fail is when an exception occurs due to the read or write. For locations designated as read and write, the routine first saves the contents of the location, then writes a test pattern that differs from the location's original contents, reads the test pattern back, compares the patterns, and finally restores the location's original contents. If the pattern comparison fails, a failure status is returned.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/visibilityTests.h** (**VISIBILITY_PARAMS**).

This test's default parameters are as follows:
deviceList = **NULL**
numDevices = **0**

The contents of the structure and their effects on the test are discussed below:

deviceList

is set to **NULL** by default and is ignored otherwise.

numDevices

is set to **0** by default and is ignored otherwise.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INSTALL_DEV_FAILED—device installation failed
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_VISIBILITY_FAULT—device failed visibility test
BIT_NO_VISIBILITY_LOCATION—no location on device designated for visibility testing

PCI Host Bridge Test 5

PCI Host Bridge Test

This chapter provides a description and requirements for the PCI host bridge (PHB) test.

The table below highlights the test’s string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_PHB_DEVICE_VISIBILITY]	tests/visibilityTests.h	VISIBILITY_PARAMS

PCI Host Bridge Device Visibility Test

[**BIT_PHB_DEVICE_VISIBILITY**]
tests/visibilityTests.h

The **tests/visibilityTests.h** file defines the parameter structures and fields mentioned in the [**BIT_PHB_DEVICE_VISIBILITY]** test.

This test probes devices in a list to determine if the device is visible. The PHB and system memory controller (SMC) visibility tests use a common test routine and test mechanism.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	RUN_TILL_COMPLETION

Test Description

This test will use its default parameters by passing a **NULL** test parameter structure to the test routine. This test will ensure the visibility of devices on both sides of the **BIT_LOCAL_BUS_TO_PCI_BRIDGE**. A list of devices that will be tested are as follows:

BIT_LOCAL_BUS_TO_PCI_BRIDGE

BIT_INTERRUPT_CONTROLLER

BIT_PCI_TO_VME_BRIDGE

BIT_ETHERNET_DEVICE1

BIT_ETHERNET_DEVICE2

BIT_ASYNC_SERIAL_DEVICE3

BIT_ASYNC_SERIAL_DEVICE4

BIT_SCSI_DEVICE1

A location is designated for visibility testing on each device. For locations designated as read-only or write-only, the routine does not verify that the read or write succeeded. The only situations that cause a read-only or write-only visibility test to fail is when an exception occurs due to the read or write. For locations designated as read and write, the routine first saves the contents of the location, then writes a test pattern that differs from the location’s original contents, reads the test pattern back, compares the patterns, and finally restores the location’s original contents. If the pattern comparison fails, a failure status is returned.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/visibilityTests.h** (**VISIBILITY_PARAMS**).

This test's default parameters are as follows:
deviceList = **NULL**
numDevices = **0**

The contents of the structure and its effects on the test are discussed below:

deviceList

is set to **NULL** by default and is ignored otherwise.

numDevices

is set to **0** by default and is ignored otherwise.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INSTALL_DEV_FAILED—device installation failed
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_VISIBILITY_FAULT—device failed visibility test
BIT_NO_VISIBILITY_LOCATION—no location on device designated for visibility testing

Multiprocessor and ISA Interrupt Controller Tests

6

This chapter provides descriptions and requirements for the following multiprocessor interrupt controller (MPIC) and the ISA interrupt controller tests:

MPIC Interrupt Test on page 6-1

ISA Interrupt Test on page 6-3

Multiprocessor and ISA Interrupt Controller Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_MPIC_INTERRUPTS]	tests/interrupts/interruptsTests.h	INTERRUPT_TEST_ PARAMS
[BIT_ISA_INTERRUPTS]	tests/interrupts/interruptsTests.h	INTERRUPT_TEST_ PARAMS

MPIC Interrupt Test

[**MPIC_INTERRUPTS**]
tests/interrupts/interruptsTests.h

The **tests/interrupts/interruptsTests.h** file defines the parameter structures and fields mentioned in the [**BIT_MPIC_INTERRUPTS**] test.

This test verifies that the MPIC is able to accept and route device interrupts to the controlling processor.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

This test causes interrupts to be generated on select interrupt-capable devices. The test monitors the processor to ensure that the correct interrupt is received. Multiple interrupt sources are used in an attempt to determine if the interrupt controller is fully operational, marginally operational, or nonfunctional. If interrupts are received from all of the devices, the controller is considered functional. If interrupts are received from a majority of the devices, but not all, the controller is considered marginal. All other cases are considered a failure.

Note If the controller is considered marginal, it does not necessarily mean that the interrupt controller is at fault, but that the device(s) providing the interrupt may be at fault.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/visibilityTests.h** (**INTERRUPT_TEST_PARAMS**).

This test’s default parameter is as follows:
validParamsFlag = **INTERRUPT_TESTS_VALID_FLAG**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_MPIC_INTERRUPTS** test. The user must use the definition of **INTERRUPT_TESTS_VALID_FLAG**.

Return Values

BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_MPIC_INTERRUPT_FAULT—MPIC interrupt controller fault
BIT_MPIC_INTERRUPT_MARGINAL—MPIC interrupt controller marginal

ISA Interrupt Test

[**BIT_ISA_INTERRUPTS**]
tests/interrupts/interruptsTests.h

The **tests/interrupts/interruptsTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ISA_INTERRUPTS**] test.

This test verifies that the ISA interrupt controller is able to accept and route device interrupts to the controlling processor.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The interrupt controller test causes interrupts to be generated on select interrupt capable devices. The test monitors the processor to ensure that the correct interrupt is received. Multiple interrupt sources are used in an attempt to determine if the interrupt controller is fully operational, marginally operational, or nonfunctional. If interrupts are received from all of the devices, the controller is considered functional. If interrupts are received from a majority of the devices, but not all, the controller is considered marginal. All other cases are considered a failure.

Note If the controller is considered marginal, it does not necessarily mean that the interrupt controller is at fault, but that the device(s) providing the interrupt may be at fault.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/visibilityTests.h** (**INTERRUPT_TEST_PARAMS**).

This test’s default parameter is as follows:
validParamsFlag = **INTERRUPT_TESTS_VALID_FLAG**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_ISA_INTERRUPTS** test. The user must use the definition of **INTERRUPT_TESTS_VALID_FLAG**.

Return Values

BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_ISA_INTERRUPT_FAULT—ISA interrupt controller fault
BIT_ISA_INTERRUPT_MARGINAL—ISA interrupt controller marginal

ECC Memory Tests 7

This chapter provides descriptions and requirements for the following ECC memory tests:

ECC Memory Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_RAM_BIT_WALK] [BIT_RAM_ONES_COMPLEMENT] [BIT_RAM_PATTERNS]	tests/memory/ ramPatternTests.h	RAM_PATTERN_ PARAMS
[BIT_RAM_ADDR_PERMUTATIONS]	tests/memory/ ramPermutationTests.h	RAM_PERMUTATION_ PARAMS
[BIT_ECC_SBIT_ERROR_INSERTION] [BIT_ECC_MBIT_ERROR_INSERTION]	tests/memory/ eccRamTests.h	ECC_BIT_ERROR_ PARAMS

RAM Bit Walk, Ones Complement, and Patterns Tests

[**BIT_RAM_BIT_WALK**], [**BIT_RAM_ONES_COMPLEMENT**], [**BIT_RAM_PATTERNS**]
tests/memory/ramPatternTests.h

The **tests/memory/ramPatternTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RAM_BIT_WALK**], [**BIT_RAM_ONES_COMPLEMENT**], and [**BIT_RAM_PATTERNS**] tests.

Each of these RAM tests has the same general description and requirements, and all three are described under the reference of ’RAM patterns test.’

- ❏ The RAM Bit Walk [**BIT_RAM_BIT_WALK**] test writes and then reads 0x55AA55AA and 0xAA55AA55 over the memory test region. This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

- ❏ The RAM Ones Complement [**BIT_RAM_ONES_COMPLEMENT**] test writes and then reads 0x87654321 and its complements, then 0x12345678 and its complement over the memory test region. This test has the following default test values:

Iteration:	1
Duration:	100000
Control:	HALT_ON_ERROR

- ❏ The RAM Patterns [**BIT_RAM_PATTERNS**] test writes and then reads 0xFFFFFFFF, 0x00000000, 0x01010101, 0x03030303, 0x07070707, 0x0F0F0F0F, 0x1F1F1F1F, 0x3F3F3F3F, and 0x7F7F7F7F over the memory test region. This test has the following default test values:

Iteration:	1
Duration:	200000
Control:	HALT_ON_ERROR

Test Description

The RAM patterns test uses a common routine that takes a different list of patterns for each of the different tests. The protected install routine disables the L2 cache while leaving the L1 cache enabled to allow tests over large memory regions to complete in reasonable amounts of time. The protected uninstall routine re-enables the L2 cache.

The RAM test starts by writing the first pattern of the pattern list, one word at a time, across the entire memory region under test. Following pattern writes to the memory region under test, the L1 data cache is flushed and invalidated to ensure the pattern is actually written to (and then read from) RAM. The test then compares the data in the memory region to the current pattern one word at a time. If data at any memory location differs from the current pattern, an error is flagged. If the test control is set to halt on the first error, the test returns the failed status immediately. Otherwise, the test runs to completion and returns the failed status after all patterns have been tested. After testing the memory region with the first pattern, the same loop is repeated for all remaining patterns in the pattern list.

Affected Peripheral Devices

This test does not affect any peripheral devices, however, it does disable the L2 cache during the test.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/memory/ramPatternTests.h** (**RAM_PATTERN_PARAMS)**.

The parameters that most affect test performance are the *bufferPtr*/*numBytes* combination and the *patternPtr*/*numPatterns* combination. Also note how the interactions differ between *bufferPtr*/*numBytes* and *patternPtr*/*numPatterns*. In general, using the defaults result in the most exhaustive test.

These three tests' default parameters are as follows:
validParamsFlag = **RAM_PAT_TESTS_VALID_FLAG**
bufferPtr = **NULL**
numBytes = **0**
patternPtr = **NULL**
numPatterns = **0**
deinstallFreesBuffer = **TRUE**

The contents of the structure and their effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure the user is providing the right data structure to the RAM pattern tests. The user must use the definition of **RAM_PAT_TESTS_VALID_FLAG**. This value must be set regardless of whether the user intends to let the test use its defaults. Neither the install routine nor the test routine sets this flag; they only verify its correctness.

bufferPtr

provides a pointer to the RAM buffer to test. This pointer can take any address within the RAM address space, including 0x0000_0000 (**NULL**). As a result, this value is not used to determine if a valid memory buffer has been provided. If *bufferPtr* is outside the valid RAM address space (determined using the device descriptor for RAM), the test is not executed.

The *bufferPtr*/*numBytes* relationship is based on the fact that *bufferPtr* can be **NULL** (0x0000_0000 is a valid RAM address). Therefore, a non-zero *numBytes* value is used to indicate *bufferPtr* is valid and a *numBytes* value of **0** indicates the install routine should allocate the memory region to test. If the install routine allocates the memory, it allocates the largest available buffer and sets *bufferPtr* and *numBytes* accordingly. It also sets *deinstallFreesBuffer* to **TRUE** to indicate the deinstall routine frees the memory.

numBytes

provides the size of the memory test region in bytes. This value should be a multiple of the system word size. If it is not, the trailing bytes are not tested (the number of bytes not tested is less than the word size). This value, along with *validParamsFlag*, is used to determine if a valid *bufferPtr* has been provided. If no buffer is provided to the test, *numBytes* should be **0**. This allows the install routine to allocate the memory region and set *bufferPtr* and *numBytes* accordingly.

patternPtr

is an array of word-sized patterns used to test the memory region. If this value is **NULL** or *numPatterns* is **0**, the test uses the appropriate default pattern list for the designated test.

The *patternPtr*/*numPatterns* relationship is such that if *patternPtr* is **NULL**, *numPatterns* is ignored and the default values are used for *patternPtr* and *numPatterns*. On the other hand, if *numPatterns* is **0**, *patternPtr* is ignored and the default values are used to set *patternPtr* and *numPatterns*. In each case, the default patterns used depends on the RAM test being executed.

numPatterns

provides the number of patterns in the *patternPtr* list. If this value is **0** or *patternPtr* is **NULL**, the install routine uses the appropriate default pattern list for the designated test. See the *patternPtr* description.

deinstallFreesBuffer

is a Boolean flag that indicates if the deinstall routine is permitted to free the memory buffer pointed to *bufferPtr*. If the flag is **TRUE**, the test routine's deinstall routine frees *bufferPtr* regardless of where it was allocated. If it is **FALSE**, the buffer is not freed and the user becomes responsible for freeing the test buffer. See the *bufferPtr* description.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence

RAM Address Permutation Test

[**BIT_RAM_ADDR_PERMUTATIONS**]
tests/memory/ramPermutationTests.h

The **tests/memory/ramPermutationTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RAM_ADDR_PERMUTATIONS**] test.

This test performs word, half word, and byte address permutations over the memory test region.

This test has the following default test values:

Iteration:	1
Duration:	25000
Control:	HALT_ON_ERROR

Test Description

The RAM address permutation test runs in three phases.

Phase I Patterns

word:	0x10111213, 0x14151617, 0x18191A1B, 0x1C1D1E1F
half word:	0x1011, 0x1213, 0x1415, 0x1617, 0x1011, 0x1A1B, 0x1C1D, 0x1E1F
byte:	0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F

Phase II Patterns

word:	0x20212223, 0x24252627, 0x28292A2B, 0x2C2D2E2F
half word:	0x2021, 0x2223, 0x2425, 0x2627, 0x2829, 0x2A2B, 0x2C2D, 0x2E2F
byte:	0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F

Phase III Patterns

word:	0x30313233, 0x34353637, 0x38393A3B, 0x3C3D3E3F
half word:	0x3031, 0x3233, 0x3435, 0x3637, 0x3839, 0x3A3B, 0x3C3D, 0x3E3F
byte:	0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F

The test patterns used are each 16 bytes in length. The pattern values are not as important as the requirement that each of the 16 bytes has unique values and that each of the phases has unique values. The permutation test is then performed 16 bytes at a time.

If a data comparison fails after any of the reads, the status is set to a failure status. If the test control is set to **HALT_ON_ERROR**, the test returns the failure status immediately. Otherwise, the test continues over the entire memory region under test.

The protected install routine disables the L2 cache and the L1 data cache. This forces the processor to perform single beat accesses. The protected uninstall routine re-enables the L1 data cache and the L2 cache.

Affected Peripheral Devices

This test does not affect any peripheral devices, however, it does disable the L2 cache and the L1 data cache during the test.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/memory/ramPermutationTests.h** (**RAM_PERMUTATION_PARAMS**).

The parameters that most affect test performance are the *bufferPtr*/*numBytes* combination and *numWordsToSkip*. In general, using defaults results in the most efficient test. An exhaustive test over the designated memory region would require *numWordsToSkip* to be **0**. Remember that if the memory test region is large, setting *numWordsToSkip* can take a very long time to complete.

This test's default parameters are as follows:
validParamsFlag = **RAM_PERM_TESTS_VALID_FLAG**
bufferPtr = **NULL**
numBytes = **0**
numWordsToSkip = **–1**
deinstallFreesBuffer = **TRUE**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the RAM pattern tests. The user must use the definition of **RAM_PERM_TESTS_VALID_FLAG**. This value must be set regardless of whether the user intends to let the test use its defaults. Neither the install routine nor the test routine sets this flag, they only verify its correctness.

bufferPtr

provides a pointer to the RAM buffer to test. This pointer can take any address within the RAM address space, including 0x0000_0000 (**NULL**). As a result, this value is not used to determine if a valid memory buffer has been provided. See the description of *numBytes* for this functionality. If *bufferPtr* is outside the valid RAM address space (determined using the device descriptor for RAM), the test is not executed.

The *bufferPtr*/*numBytes* relationship is based on the fact that *bufferPtr* can be **NULL** (0x0000_0000 is a valid RAM address). Therefore, a non-zero *numBytes* value is used to indicate *bufferPtr* is valid and a *numBytes* value of **0** indicates the install routine should allocate the memory region to test. If the install routine allocates the memory, it allocates the largest available buffer and sets *bufferPtr* and *numBytes* accordingly. It also sets *deinstallFreesBuffer* to **TRUE** to indicate the deinstall routine frees the memory.

numBytes

provides the size of the memory test region in bytes. This value should be a multiple of the system word size. If it is not, the trailing bytes are not tested (the number of bytes not tested is less than the word size). This value along with *validParamsFlag* is used to determine if a valid *bufferPtr* has been provided. If no buffer is provided to the test, *numBytes* should be **0**, which allows the install routine to allocate the memory region and set *bufferPtr* and *numBytes* accordingly. See the *bufferPtr* description.

numWordsToSkip

causes the test to skip over the specified number of words. This permits tests over large memory regions to complete in reasonable amounts of time. If *numWordsToSkip* is negative, the default **PERMUTATION_WORDS_TO_SKIP** is used. If *numWordsToSkip* is **0**, no words are skipped and every location in the memory region is tested. Any other value causes the specified number of words to be skipped.

deinstallFreesBuffer

is a Boolean flag that indicates if the deinstall routine is permitted to free the memory buffer pointed to *bufferPtr*. If the flag is **TRUE**, the test routine’s deinstall routine frees *bufferPtr* regardless of where it was allocated. If it is **FALSE**, the buffer is not freed and the user becomes responsible for freeing the test buffer.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INSTALL_TEST_FAILED—test installation failed
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence

ECC Single- and Multi-Bit Error Insertion Tests

[**BIT_ECC_SBIT_ERROR_INSERTION**],
[**BIT_ECC_MBIT_ERROR_INSERTION**]
tests/memory/eccRamTests.h

The **tests/memory/eccRamTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ECC_SBIT_ERROR_INSERTION**] and [**BIT_ECC_MBIT_ERROR_INSERTION**] tests.

Each of the ECC bit error tests has the same general description and requirements. The individual tests only vary in whether they test single- or multi-bit errors.

❏ The ECC Single-Bit Error test [**BIT_ECC_SBIT_ERROR_INSERTION**] inserts a single erroneous bit into the memory test region to cause a single-bit error. This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

❏ The ECC Multi-Bit Error test [**BIT_ECC_MBIT_ERROR_INSERTION**] inserts two (or more) erroneous bits into the memory test region to cause a multi-bit error. This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The single- and multi-bit error tests use a common routine that inserts either one or more erroneous bits depending on the test being run. After causing a single-bit error, the test expects the single-bit error flag to be set and validated, the data to be corrected, the correction not to be the result of scrubbing, the single-bit error count to be one, the error syndrome to be correct, and that no exception is generated.

Testing multi-bit errors expects that the multi-bit error flag be set and valid and that the exception be generated by the test routine, not by scrubbing. For multi-bit errors, the data is not checked for correctness and therefore, the error syndrome is also not checked for correctness.

Memory region size and alignment is memory controller specific. For the Hawk SMC, the tests require a list of memory regions that are at least eight bytes long and have 8-byte alignment in memory. The 8-byte size requirement is the result of a design that uses one byte of check bits per eight bytes of data. The 8-byte alignment is required to properly match the data bytes with their corresponding check bits. This alignment and size value is defined as **NUM_ECC_DATA_BYTES**.

The test routines accept a list of memory regions to be tested. The list can contain up to **MAX_MEM_BANKS** memory regions but only the first properly aligned eight bytes of the memory region are actually tested. If no memory regions are provided, the test routine allocates and aligns a single memory region for the test. If a user wishes to test all memory banks (there are up to eight on the Hawk), then the user is responsible for properly handling and protecting any memory regions that may be in use by other applications. That is, no mutual exclusion is performed while the memory regions are being modified during the tests, and data at the test locations is not saved and restored. Keep in mind that if the test routine defaults are used, only one memory region is allocated resulting in a test of only one memory bank.

If the given memory regions are not properly aligned but are larger than the size requirement, the routine attempts to align the memory regions. If a region cannot be aligned or is smaller than the size requirement, an error status is returned before that region is tested. If all memory regions are (or can be) properly aligned and the regions meet the size requirements, the test runs through the list until completion or halt on error, depending on the test control parameter.

The protected install routine disables the L2 cache, disables both L1 caches, and disables RAM scrubbing. Disabling all caches provides more control over accesses that are meant to cause the ECC errors. Disabling RAM scrubbing prevents the inserted ECC errors from being corrected or exceptions from being generated by RAM refresh cycles. The protected uninstall routine re-enables both L1 caches, re-enables the L2 cache, and restores scrubbing to its original state (which may have been disabled).

Affected Peripheral Devices

This test does not affect any peripheral devices, however, it does disable the L2 cache and the L1 data cache during the test. It requires the MBIT exception handler to be installed.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/memory/eccRamTests.h** (**BIT_ECC_BIT_ERROR_PARAMS**) parameter structure.

The parameters that most affect test performance are the *bufDesc[]*/*numBuffers* combination. In general, using the defaults do not result in the most exhaustive test, however, it should still be sufficient for most applications. Setting *numBuffers* to the maximum number of memory banks present (which is less than or equal to **MAX_MEM_BANKS**, depending on a board's memory configuration) and allocating memory from each of these banks' results in the most exhaustive test possible.

These two tests' default parameters are as follows:
validParamsFlag = **RAM_ECC_TESTS_VALID_FLAG**
bufDesc[] = **NULL**
numBuffers = **0**
deinstallFreesBuffer = **TRUE**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the RAM ECC tests. The user must use the definition of **RAM_ECC_TESTS_VALID_FLAG**. This value must be set regardless of whether the user intends to let the test use its defaults. Neither the install routine nor the test routine sets this flag, they only verify its correctness.

bufDesc[]

is an array of *BUF_DESC* descriptors that describe the memory regions to be tested. The *BUF_DESC* structure has only two fields, *numBytes* and *bufferPtr*. The *bufDesc* pointer (array) itself cannot be **NULL** because the install routine needs the *bufDesc[]* storage to pass the *bufferPtr* and *numBytes* values into the test routine. The *numBytes*/*bufferPtr* combination has the same relationship as described in the other RAM tests and described below.

Because *bufferPtr* can be **NULL** (0x0000_0000), *numBytes* must be used to determine whether *bufferPtr* is actually valid. That is, if *numBytes* is **0** (or less than **NUM_ECC_DATA_BYTES**), the memory region is not tested regardless of the value of *bufferPtr*. On the other hand, if *numBytes* is at least **NUM_ECC_DATA_BYTES** long and *bufferPtr* is aligned by **NUM_ECC_DATA_BYTES**, then *bufferPtr* is used regardless of its value. If any *bufDesc[]* has a value of *numBytes* that is less than **NUM_ECC_DATA_BYTES** or a *bufferPtr* that is not properly aligned and cannot be properly aligned, the test exits when it reaches that *bufDesc[]*. It tests any valid regions that exist earlier in the *bufDesc[]* array.

numBuffers

is the number of buffers provided in the *bufDesc[]* array. This value must be greater than **0** but less than or equal to **MAX_MEM_BANKS**. Only this number of buffers is tested and any other value of *numBuffers* prevents the test from running. The test uses its default *bufDesc[]*/*numBuffers* values when *numBuffers* is **0**.

deinstallFreesBuffer

is a Boolean flag that indicates if the deinstall routine is permitted to free the memory buffer pointed to by each *bufDesc[]/bufferPtr*. If the flag is **TRUE**, the test routine's deinstall routine frees each *bufferPtr* regardless of where it was allocated. If it is **FALSE**, the buffers are not freed and the user becomes responsible for freeing the test buffers.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_ECC_DETECT_ERROR—Memory error correction failed
BIT_INIT_NOT_PERFORMED—MBIT initialization not performed

Serial EEPROM Tests 8

This chapter provides descriptions and requirements for the following serial EEPROM tests:

Serial EEPROM Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_SROM_VPD_VERIFY]	None	None
[BIT_SROM_USR_DATA_READ]	None	None
[BIT_SROM_SPD_VERIFY]	None	None

Vital Product Data Verify Test

[**BIT_SROM_VPD_VERIFY**]

This test verifies the vital product data (VPD) checksum.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The VPD verification test uses VPD utility routines to both read the VPD from the I^{2}C device and then to calculate the new cyclic redundancy check (CRC). However, before calculating the new CRC the existing one in the VPD memory buffer must be zeroed. After clearing the existing CRC, the new one is calculated using another utility routine. Finally, the newly calculated CRC is compared with the original and if the comparison fails, a failure status is returned.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_VPD_SROM_FAULT—VPD error accessing the VPD SROM
BIT_VPD_CONTAINS_NO_CRC—VPD contains no valid CRC packet
BIT_VPD-CRC_FAULT—VPD CRC did not equal the calculated CRC

Serial Presence Detect Verify Test

[**BIT_SROM_SPD_VERIFY**]

This test verifies the serial presence detect (SPD) checksum.

This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

Test Description

The SPD checksum should match the calculated checksum. If the checksums do not match, the test will return an error.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SPD_SROM_FAULT—serial presence detect access fault
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied

BIT_SPD_CHECKSUM_FAULT—serial presence detect checksum fault

User Configuration Data Read Test

[**BIT_SROM_USR_DATA_READ**]

This test verifies the ability to read from the user configuration data serial EEPROM.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test uses utility routines for the I^2C device to verify the ability to read the range of memory in which this device is mapped. The test checks for errors encountered while trying to read from the serial EEPROM. If the read fails, a failure status is returned.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—subtest is not supported
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_USER_DATA_SROM_FAULT—SROM user configuration data access error

NVRAM Test 9

This chapter provides a description and requirements for the NVRAM test.

NVRAM Test

The table below highlights the test’s string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_NVRAM_PATTERNS]	tests/m48t37y_rtc/nvRamTests.h	NVRAM_TEST_PARAMS

NVRAM Predefined Memory Test

[**BIT_NVRAM_PATTERNS**]
tests/m48t37y_rtc/nvRamTests.h

The **tests/m48t37y_rtc/nvRamTests.h** file defines the parameter structures and fields mentioned in the [**BIT_NVRAM_PATTERNS**] test.

This test verifies the correct operation of the system's NVRAM.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

This test performs a bit test of all bits and all memory locations within the NVRAM memory space. It provides two methods of testing the NVRAM. The default method uses a predefined set of patterns that is written to the memory and then read back to verify that the memory can be written and read. The second method allows the user to provide his or her own test data to exercise the NVRAM space, allowing complete control of how the memory space is tested.

The user may also select and test a subset of the NVRAM space by modifying the test parameters that are provided at the time the test is run.

Affected Peripheral Devices

This test does not affect any peripheral devices. To ensure that the NVRAM test is nondestructive, all of the NVRAM is saved before the test is run and restored after the test is complete.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/nvRamTests.h** (**NVRAM_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **NVRAM_TESTS_VALID_FLAG**
numPatterns = **NVRAM_PREDEF_TESTS**
bufferOffset = **0**
bufferLength = **0**
dataBufferPtr = **NULL**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the NVRAM test. The user must use the definition of **NVRAM_TESTS_VALID_FLAG**.

numPatterns

allows the user to determine which, if any, of the predefined tests are to be used. The default value is **NVRAM_PREDEF_TESTS** for using the predefined data patterns. The user may use any value less than this value, including **0**. If the user elects to not use the predefined tests, a value of **0** should be provided and the user must provide a non-**NULL** *dataBufferPtr* that is pointing to the user's test data.

bufferOffset

is provided to allow the user to test a subset of the NVRAM starting at an address greater than the start of memory. The default value is **0**, which ensures all the memory is tested. If it is not **0**, then all the memory before the offset is not tested.

bufferLength

allows the user to determine the amount of memory to be tested from the offset provided in the previous parameter. The default value is **0**, indicating all of the NVRAM is tested. The user may provide his or her buffer length to limit the amount of memory that is tested from the starting address. The user must ensure that the sum of the *bufferOffset* and *bufferLength* does not exceed the size of the NVRAM.

dataBufferPtr

allows the user to provide unique data patterns by setting the parameter to the address of the test data buffer. The default value is **0**, indicating that the user does not want to provide his or her own data buffer.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable

Real Time Clock Tests 10

This chapter provides descriptions and requirements for the following real time clock (RTC) tests:

Real Time Clock Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_RTC_BATTERY]	tests/m48t37y_rtc/rtcTests.h	RTC_TIME_PARAMS
[BIT_RTC_ALARM]	tests/m48t37y_rtc/rtcTests.h	RTC_TIME_PARAMS
[BIT_RTC_CLOCK]	tests/m48t37y_rtc/rtcTests.h	RTC_TIME_PARAMS
[BIT_RTC_SET_CLOCK]	tests/m48t37y_rtc/rtcTests.h	RTC_TIME_PARAMS
[BIT_RTC_CLOCK_ACCURACY]	tests/m48t37y_rtc/rtcTests.h	RTC_TIME_PARAMS
[BIT_RTC_WATCHDOG]	tests/m48t37y_rtc/rtcTests.h	RTC_WATCHDOG_ PARAMS

Real Time Clock Battery Test

[**BIT_RTC_BATTERY**]
tests/m48t37y_rtc/rtcTests.h

The **tests/m48t37y_rtc/rtcTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RTC_BATTERY**] test.

This test verifies that the battery used to maintain the clock is functional.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The battery test reads the RTC chip and reports back the status.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/rtcTests.h** (**RTC_TIME_PARAMS**).

This test’s default parameters are as follows:
validParamsFlag = **RTC_TESTS_VALID_FLAG**
year = **RTC_NO_CHANGE**
month = **RTC_NO_CHANGE**
date = **RTC_NO_CHANGE**
day = **RTC_NO_CHANGE**

hour = **RTC_NO_CHANGE**
minute = **RTC_NO_CHANGE**
second = **RTC_NO_CHANGE**
restoreClock = **TRUE**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_RTC_BATTERY** test. The user must use the definition of **RTC_TESTS_VALID_FLAG**.

year

determines the year value of the clock. The value of this parameter should be between **0** and **9999**. If the user provides **RTC_NO_CHANGE**, the current year setting is used.

month

determines the month value of the clock. This parameter must be within the range of **1** to **12**. If the user provides **RTC_NO_CHANGE**, the current month setting is used.

date

determines the date value of the clock. This parameter must be in the range of **1** to **31**. If the user provides **RTC_NO_CHANGE**, the current date setting is used.

day

determines the day of the week value of the clock. This parameter must be in the range of **1** to **7**. If the user provides **RTC_NO_CHANGE**, the current day setting is used.

hour

determines the hour value of the clock. This parameter must be in the range of **0** to **23**. If the user provides **RTC_NO_CHANGE**, the current hour setting is used.

minute

determines the minute value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current minute setting is used.

second

determines the second value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current second setting is used.

restoreClock

determines if the clock is returned to its original time upon completion of the test. If the value is **FALSE** (**0**), the clock continues to maintain its new time setting. If the value is **TRUE** (non-zero), the clock is restored to its original time before the test.

Note The user should provide a number other than **RTC_NO_CHANGE** to at least one of the parameters to achieve a reasonable time in the future that the alarm expires. If this is not the case, the test may take an excessive amount of time to run. Also, if a calculated value causes a rollover of that parameter to the next higher time unit, that higher time unit is also updated to provide the intended amount of time to be applied to the alarm calculation.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)

BIT_BATTERY_LOW_POWER—NVRAM battery power is low. Replace battery.

Real Time Clock Alarm Test

[BIT_RTC_ALARM]
tests/m48t37y_rtc/rtcTests.h

The **tests/m48t37y_rtc/rtcTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RTC_ALARM**] test.

This test verifies that the clock can generate an alarm when properly set.

This test has the following default test values:

Iteration:	1
Duration:	3000
Control:	HALT_ON_ERROR

Test Description

The alarm test configures the chip to wake the test program when a user-defined time event is reached. The user provides date, month, hour, minute, and second values that are added to the current time parameters to calculate the alarm time. When the alarm time is reached, the test is allowed to proceed, determining if the alarm expired at the correct time. It is possible for the user to set the alarm time sufficiently large as to cause the alarm time delay to be extremely large. Therefore, the user should use prudence in providing parameters to this test.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/rtcTests.h** (**RTC_TIME_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **RTC_TESTS_VALID_FLAG**
year = **RTC_NO_CHANGE**
month = **RTC_NO_CHANGE**
date = **RTC_NO_CHANGE**
day = **RTC_NO_CHANGE**
hour = **RTC_NO_CHANGE**
minute = **RTC_NO_CHANGE**
second = **2**
restoreClock = **TRUE**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_RTC_ALARM** test. The user must use the definition of **RTC_TESTS_VALID_FLAG**.

year

determines the year value of the clock. The value of this parameter should be between **0** and **9999**. If the user provides **RTC_NO_CHANGE**, the current year setting is used.

month

determines the month value of the clock. This parameter must be within the range of **1** to **12**. If the user provides **RTC_NO_CHANGE**, the current month setting is used.

date

determines the date value of the clock. This parameter must be in the range of **1** to **31**. If the user provides **RTC_NO_CHANGE**, the current date setting is used.

day

determines the day of the week value of the clock. This parameter must be in the range of **1** to **7**. If the user provides **RTC_NO_CHANGE**, the current day setting is used.

hour

determines the hour value of the clock. This parameter must be in the range of **0** to **23**. If the user provides **RTC_NO_CHANGE**, the current hour setting is used.

minute

determines the minute value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current minute setting is used.

second

determines the second value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current second setting is used.

restoreClock

determines if the clock is returned to its original time upon completion of the test. If the value is **FALSE** (**0**), the clock continues to maintain its new time setting. If the value is **TRUE** (non-zero), the clock is restored to its original time before the test.

Note The user should provide a number other than **RTC_NO_CHANGE** to at least one of the parameters to achieve a reasonable time in the future that the alarm expires. If this is not the case, the test may take an excessive amount of time to run. Also, if a calculated value causes a rollover of that parameter to the next higher time unit, that higher time unit is also updated to provide the intended amount of time to be applied to the alarm calculation.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_RTC_ALARM_FAULT—RTC alarm timer fault
BIT_RTC_ALARM_ACCURACY_FAULT—RTC alarm time accuracy fault

Real Time Clock Test

[**BIT_RTC_CLOCK**]
tests/m48t37y_rtc/rtcTests.h

The **tests/m48t37y_rtc/rtcTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RTC_CLOCK**] test.

This test verifies that the clock can increment properly when enabled.

This test has the following default test values:

Iteration:	1
Duration:	3000
Control:	HALT_ON_ERROR

Test Description

The clock test ensures that the clock is enabled and then waits a predefined amount of time to ensure that the clock has increments that amount of time.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/rtcTests.h** (**RTC_TIME_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **RTC_TESTS_VALID_FLAG**
year = **RTC_NO_CHANGE**
month = **RTC_NO_CHANGE**
date = **RTC_NO_CHANGE**
day = **RTC_NO_CHANGE**
hour = **RTC_NO_CHANGE**
minute = **RTC_NO_CHANGE**
second = **RTC_NO_CHANGE**
restoreClock = **TRUE**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_RTC_CLOCK** test. The user must use the definition of **RTC_TESTS_VALID_FLAG**.

year

determines the year value of the clock. The value of this parameter should be between **0** and **9999**. If the user provides **RTC_NO_CHANGE**, the current year setting is used.

month

determines the month value of the clock. This parameter must be within the range of **1** to **12**. If the user provides **RTC_NO_CHANGE**, the current month setting is used.

date

determines the date value of the clock. This parameter must be in the range of **1** to **31**. If the user provides **RTC_NO_CHANGE**, the current date setting is used.

day

determines the day of the week value of the clock. This parameter must be in the range of **1** to **7**. If the user provides **RTC_NO_CHANGE**, the current day setting is used.

hour

determines the hour value of the clock. This parameter must be in the range of **0** to **23**. If the user provides **RTC_NO_CHANGE**, the current hour setting is used.

minute

determines the minute value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current minute setting is used.

second

determines the second value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current second setting is used.

restoreClock

determines if the clock is returned to its original time upon completion of the test. If the value is **FALSE** (**0**), the clock continues to maintain its new time setting. If the value is **TRUE** (non-zero), the clock is restored to its original time before the test.

Note The user should provide a number other than **RTC_NO_CHANGE** to at least one of the parameters to achieve a reasonable time in the future that the alarm expires. If this is not the case, the test may take an excessive amount of time to run. Also, if a calculated value causes a rollover of that parameter to the next higher time unit, that higher time unit is also updated to provide the intended amount of time to be applied to the alarm calculation.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_RTC_CLOCK_FAULT—RTC clock read fault

Real Time Clock Set Test

[**BIT_RTC_SET_CLOCK**]
tests/m48t37y_rtc/rtcTests.h

The **tests/m48t37y_rtc/rtcTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RTC_SET_CLOCK**] test.

This test verifies that the clock can be set.

This test has the following default test values:

Iteration:	1
Duration:	3000
Control:	HALT_ON_ERROR

Test Description

The clock set test ensures that the clock is running and then sets the time to values provided by the user. Once the clock is set, the test waits a predefined amount of time to ensure that the clock is running and that the time is accurate to the new time. If commanded by a user parameter, the clock is restored to its original value.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/rtcTests.h** (**RTC_TIME_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **RTC_TESTS_VALID_FLAG**
year = **2001**
month = **3**
date = **21**
day = **3**
hour = **4**
minute = **15**
second = **10**
restoreClock = **TRUE**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_RTC_SET_CLOCK** test. The user must use the definition of **RTC_TESTS_VALID_FLAG**.

year

determines the year value of the clock. The value of this parameter should be between **0** and **9999**. If the user provides **RTC_NO_CHANGE**, the current year setting is used.

month

determines the month value of the clock. This parameter must be within the range of **1** to **12**. If the user provides **RTC_NO_CHANGE**, the current month setting is used.

date

determines the date value of the clock. This parameter must be in the range of **1** to **31**. If the user provides **RTC_NO_CHANGE**, the current date setting is used.

day

determines the day of the week value of the clock. This parameter must be in the range of **1** to **7**. If the user provides **RTC_NO_CHANGE**, the current day setting is used.

hour

determines the hour value of the clock. This parameter must be in the range of **0** to **23**. If the user provides **RTC_NO_CHANGE**, the current hour setting is used.

minute

determines the minute value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current minute setting is used.

second

determines the second value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current second setting is used.

restoreClock

determines if the clock is returned to its original time upon completion of the test. If the value is **FALSE** (**0**), the clock continues to maintain its new time setting. If the value is **TRUE** (non-zero), the clock is restored to its original time before the test.

Note The user should provide a number other than **RTC_NO_CHANGE** to at least one of the parameters to achieve a reasonable time in the future that the alarm expires. If this is not the case, the test may take an excessive amount of time to run. Also, if a calculated value causes a rollover of that parameter to the next higher time unit, that higher time unit is also updated to provide the intended amount of time to be applied to the alarm calculation.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_RTC_CLOCK_SET_FAULT—RTC clock set fault

Real Time Clock Accuracy Test

[**BIT_RTC_CLOCK_ACCURACY**]
tests/m48t37y_rtc/rtcTests.h

The **tests/m48t37y_rtc/rtcTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RTC_CLOCK_ACCURACY**] test.

This test verifies that the clock increments with a predetermined accuracy.

This test has the following default test values:

Iteration:	1
Duration:	35000
Control:	HALT_ON_ERROR

Test Description

This test ensures that the clock is enabled and then waits a predefined amount of time to ensure that the clock has increments to the correct time.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/rtcTests.h** (**RTC_TIME_PARAMS**).

This test’s default parameters are as follows:
validParamsFlag = **RTC_TESTS_VALID_FLAG**
year = **RTC_NO_CHANGE**
month = **RTC_NO_CHANGE**
date = **RTC_NO_CHANGE**

day = **RTC_NO_CHANGE**
hour = **RTC_NO_CHANGE**
minute = **RTC_NO_CHANGE**
second = **RTC_NO_CHANGE**
restoreClock = **TRUE**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_RTC_CLOCK_ACCURACY** test. The user must use the definition of **RTC_TESTS_VALID_FLAG**.

year

determines the year value of the clock. The value of this parameter should be between **0** and **9999**. If the user provides **RTC_NO_CHANGE**, the current year setting is used.

month

determines the month value of the clock. This parameter must be within the range of **1** to **12**. If the user provides **RTC_NO_CHANGE**, the current month setting is used.

date

determines the date value of the clock. This parameter must be in the range of **1** to **31**. If the user provides **RTC_NO_CHANGE**, the current date setting is used.

day

determines the day of the week value of the clock. This parameter must be in the range of **1** to **7**. If the user provides **RTC_NO_CHANGE**, the current day setting is used.

hour

determines the hour value of the clock. This parameter must be in the range of **0** to **23**. If the user provides **RTC_NO_CHANGE**, the current hour setting is used.

minute

determines the minute value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current minute setting is used.

second

determines the second value of the clock. This parameter must be in the range of **0** to **59**. If the user provides **RTC_NO_CHANGE**, the current second setting is used.

restoreClock

determines if the clock is returned to its original time upon completion of the test. If the value is **FALSE** (**0**), the clock continues to maintain its new time setting. If the value is **TRUE** (non-zero), the clock is restored to its original time before the test.

Note The user should provide a number other than **RTC_NO_CHANGE** to at least one of the parameters to achieve a reasonable time in the future that the alarm expires. If this is not the case, the test may take an excessive amount of time to run. Also, if a calculated value causes a rollover of that parameter to the next higher time unit, that higher time unit is also updated to provide the intended amount of time to be applied to the alarm calculation.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed

BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_RTC_CLOCK_ACCURACY_FAULT—RTC clock accuracy fault

Watchdog Timer Test

[**BIT_RTC_WATCHDOG**]
tests/m48t37y_rtc/rtcTests.h

The **tests/m48t37y_rtc/rtcTests.h** file defines the parameter structures and fields mentioned in the [**BIT_RTC_WATCHDOG**] test.

This test verifies that the watchdog timer functions properly.

This test has the following default test values:

Iteration:	1
Duration:	3000
Control:	HALT_ON_ERROR

Test Description

This test verifies that the timer can be set and restarted before an interrupt occurs. It also verifies that an interrupt is generated at the appropriate time, waking the waiting test.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/m48t37y_rtc/rtcTests.h** (**RTC_WATCHDOG_PARAMS**).

This test’s default parameters are as follows:
validParamsFlag = **RTC_TESTS_VALID_FLAG**
resolution = **0**
multiplier = **2**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_RTC_WATCHDOG** test. The user must use the definition of **RTC_TESTS_VALID_FLAG**.

resolution

determines the watchdog timer resolution. The user must provide one of the parameters defined in the **RTC_RESOLUTION** enumeration. The available choices are **ONE_SIXTEENTH**, **ONE_QUARTER**, **ONE_SECOND**, and **FOUR_SECONDS**.

multiplier

determines the amount of time the watchdog is active by multiplying this value with the *resolution*. The range of this parameter is between **1** and **31**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_RTC_WATCHDOG_FAULT—RTC watchdog timer fault

BIT_RTC_WATCHDOG_EARLY_FAULT—RTC watchdog timer early fault

UART Tests 11

This chapter provides descriptions and requirements for the following UART tests:

UART Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_SERIAL_REGISTER]	tests/serial/serialTests.h	SERIAL_TEST_PARAMS
[BIT_SERIAL_BAUD_RATE]	tests/serial/serialTests.h	SERIAL_TEST_PARAMS
[BIT_SERIAL_INTERNAL_ LOOPBACK_POLL]	tests/serial/serialTests.h	SERIAL_TEST_PARAMS
[BIT_SERIAL_INTERNAL_ LOOPBACK_INT]	tests/serial/serialTests.h	SERIAL_TEST_PARAMS
[BIT_SERIAL_INTERNAL_ LOOPBACK_DMA]	tests/serial/serialTests.h	SERIAL_TEST_PARAMS
[BIT_SERIAL_EXTERNAL_ LOOPBACK]	tests/serial/serialTests.h	SERIAL_TEST_PARAMS

All of the UART tests have the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

UART Register Test

[**BIT_SERIAL_REGISTER**]
tests/serial/serialTests.h

The **tests/serial/serialTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SERIAL_REGISTER**] test.

This test verifies that the chip's registers can be written and read.

Test Description

This test sets the UART registers to predefined values and reads the register back to verify that the correct bits are set or cleared. All registers with reserved bits of either 0 or 1 are tested to ensure that they are in the proper state. All the remaining bits in each register are tested to ensure that they can be set to all ones, all zeroes, and an alternating bit pattern to ensure that there are no stuck or shorted bits adjacent to each other. In some cases, there are register bits that cannot be changed because they may cause the system to go into an indeterminate state. These bits are not altered to protect the system's integrity.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/serial/serialTests.h** (**SERIAL_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SERIAL_TESTS_VALID_FLAG**
baudRate = **SERIAL_IOCTL_P_BAUD_9600**
charLength = **SERIAL_IOCTL_P_CHAR_8**
stopBits = **SERIAL_IOCTL_P_STOP_1**
parity = **SERIAL_IOCTL_P_NO_PARITY**
mode = **SERIAL_IOCTL_P_POLLED**
loopBack = **SERIAL_IOCTL_P_LOOP_ON**
modem = **SERIAL_IOCTL_P_MODEM_OFF**
duration = **0**
readTimeout = **0**
writeTimeout = **0**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SERIAL_REGISTER** test. The user must use the definition of **SERIAL_TESTS_VALID_FLAG**.

baudRate

determines the rate at which the UART transmits the serial data. The value in this parameter should match the receive rate of the device that is connected to the port under test. The user should use one of the baud rate definitions listed below. These values range from **300** to **115200** baud.

SERIAL_IOCTL_P_BAUD_115200

SERIAL_IOCTL_P_BAUD_56000

SERIAL_IOCTL_P_BAUD_38400

SERIAL_IOCTL_P_BAUD_19200

SERIAL_IOCTL_P_BAUD_9600

SERIAL_IOCTL_P_BAUD_7200

SERIAL_IOCTL_P_BAUD_4800

SERIAL_IOCTL_P_BAUD_3600

SERIAL_IOCTL_P_BAUD_2400

SERIAL_IOCTL_P_BAUD_2000

SERIAL_IOCTL_P_BAUD_1800

SERIAL_IOCTL_P_BAUD_1200

SERIAL_IOCTL_P_BAUD_600

SERIAL_IOCTL_P_BAUD_300

The most commonly used parameter is **SERIAL_IOCTL_P_BAUD_9600**.

charLength

determines the number of bits to be contained in each byte of the serial data stream. Like the *baudRate* selection, this parameter should be set to match the serial device that is connected. The user should use one of the following definitions for this parameter:

SERIAL_IOCTL_P_CHAR_5

SERIAL_IOCTL_P_CHAR_6

SERIAL_IOCTL_P_CHAR_7

SERIAL_IOCTL_P_CHAR_8

The more commonly used of these parameters is **SERIAL_IOCTL_P_CHAR_8**.

stopBits

determines the number of stop bits to be used for each character transmission. The user should provide one of the following definitions in this parameter:

SERIAL_IOCTL_P_STOP_1

SERIAL_IOCTL_P_MULTI_STOP

The most commonly used of these definitions is **SERIAL_IOCTL_P_STOP_1**.

parity

determines how the output character's parity is set. This parameter should be set to match the external device's parity configuration. The user should provide one of the following definitions:

SERIAL_IOCTL_P_NO_PARITY

SERIAL_IOCTL_P_EVEN_PARITY

SERIAL_IOCTL_P_ODD_PARITY

The most commonly used of these definitions is **SERIAL_IOCTL_P_NO_PARITY**.

mode

allows the user to determine if the test runs using polling or interrupts. This test should have this parameter set to **SERIAL_IOCTL_P_POLLED**.

loopBack

determines if the test runs using the internal loopback capability of the UART. This test should have this parameter set to **SERIAL_IOCTL_P_LOOP_ON**.

modem

determines if the hardware handshake lines are used during the test. This test should have this parameter set to **SERIAL_IOCTL_P_MODEM_OFF**.

duration

is set to **0** by default and is ignored otherwise.

readTimeout

is used to modify the amount of time the test waits for an input to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

writeTimeout

is used to modify the amount of time the test waits for an output to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SERIAL_REGISTER_FAULT—serial register test fault

UART Baud Rate Test

[**BIT_SERIAL_BAUD_RATE**]
tests/serial/serialTests.h

The **tests/serial/serialTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SERIAL_BAUD_RATE**] test.

This test verifies that the UART can properly transmit a data stream.

Test Description

This test configures the UART to transmit data as determined by the input test parameters. Upon configuration, the test sends a character data stream, indicating the selected baud rate. The data is sent for the duration of the test as determined by the input test parameter.

Affected Peripheral Devices

This test sends a character data stream out the serial port connector. If there is an external device connected to this port, it should be prepared to receive a data stream of characters or be disconnected from the system before this test is run.

Required Test Equipment

Depending on the serial channel and the equipment under test, the user may have to connect a serial terminal device or other suitable serial input device. This device must be capable of detecting the serial data and displaying the data in a format that the user can easily interpret to determine if the serial port is functioning properly.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/serial/serialTests.h** (**SERIAL_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SERIAL_TESTS_VALID_FLAG**
baudRate = **SERIAL_IOCTL_P_BAUD_9600**
charLength = **SERIAL_IOCTL_P_CHAR_8**
stopBits = **SERIAL_IOCTL_P_STOP_1**
parity = **SERIAL_IOCTL_P_NO_PARITY**
mode = **SERIAL_IOCTL_P_POLLED**
loopBack = **SERIAL_IOCTL_P_LOOP_OFF**
modem = **SERIAL_IOCTL_P_MODEM_OFF**
duration = **sysClkRateGet ( ) / 2**
readTimeout = **0**
writeTimeout = **0**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SERIAL_BAUD_RATE** test. The user must use the definition of **SERIAL_TESTS_VALID_FLAG**.

baudRate

determines the rate at which the UART transmits the serial data. The value in this parameter should match the receive rate of the device that is connected to the port under test. The user should use one of the baud rate definitions listed below. These values range from **300** to **115200** baud.

SERIAL_IOCTL_P_BAUD_115200

SERIAL_IOCTL_P_BAUD_56000

SERIAL_IOCTL_P_BAUD_38400

SERIAL_IOCTL_P_BAUD_19200

SERIAL_IOCTL_P_BAUD_9600

SERIAL_IOCTL_P_BAUD_7200

SERIAL_IOCTL_P_BAUD_4800

SERIAL_IOCTL_P_BAUD_3600

SERIAL_IOCTL_P_BAUD_2400

SERIAL_IOCTL_P_BAUD_2000

SERIAL_IOCTL_P_BAUD_1800

SERIAL_IOCTL_P_BAUD_1200

SERIAL_IOCTL_P_BAUD_600

SERIAL_IOCTL_P_BAUD_300

The most commonly used parameter is **SERIAL_IOCTL_P_BAUD_9600**.

charLength

determines the number of bits to be contained in each byte of the serial data stream. Like the *baudRate* selection, this parameter should be set to match the serial device that is connected. The user should use one of the following definitions for this parameter:

SERIAL_IOCTL_P_CHAR_5

SERIAL_IOCTL_P_CHAR_6

SERIAL_IOCTL_P_CHAR_7

SERIAL_IOCTL_P_CHAR_8

The more commonly used of these parameters is **SERIAL_IOCTL_P_CHAR_8**.

stopBits

determines the number of stop bits to be used for each character transmission. The user should provide one of the following definitions in this parameter:

SERIAL_IOCTL_P_STOP_1

SERIAL_IOCTL_P_MULTI_STOP

The most commonly used of these definitions is **SERIAL_IOCTL_P_STOP_1**.

parity

determines how the output character's parity is set. This parameter should be set to match the external device's parity configuration. The user should provide one of the following definitions:

SERIAL_IOCTL_P_NO_PARITY

SERIAL_IOCTL_P_EVEN_PARITY

SERIAL_IOCTL_P_ODD_PARITY

The most commonly used of these definitions is **SERIAL_IOCTL_P_NO_PARITY**.

mode

allows the user to determine if the test runs using polling or interrupts. This test should have this parameter set to **SERIAL_IOCTL_P_POLLED**.

loopBack

determines if the test runs using the internal loopback capability of the UART. This test should have this parameter set to **SERIAL_IOCTL_P_LOOP_OFF**.

modem

determines if the hardware handshake lines are used during the test. This test should have this parameter set to **SERIAL_IOCTL_P_MODEM_OFF**.

duration

determines how long the test continues sending the serial data stream. This value is represented in clock ticks. This test should have the parameter set to **sysClkRateGet ( ) / 2**.

readTimeout

is used to modify the amount of time the test waits for an input to complete. If the user does not desire to change the time-out value, a **0** should be provided to use the default. The default value is **1 second**.

writeTimeout

is used to modify the amount of time the test waits for an output to complete. If the user does not desire to change the time-out value, a **0** should be provided to use the default. The default value is **1 second**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SERIAL_RECV_FAULT—serial receiver fault (parity, frame, overrun)

UART Internal Loopback Polled Mode Test

[**BIT_SERIAL_INTERNAL_LOOPBACK_POLL**]
tests/serial/serialTests.h

The **tests/serial/serialTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SERIAL_INTERNAL_LOOPBACK_POLL**] test.

This test verifies that the UART can properly transmit and receive a data stream in polled mode.

Test Description

This test configures the UART to transmit and receive data at a baud rate that is determined by the input test parameters. Upon configuration, the test sends a character data stream that is presented at the receiver. The data is transmitted and received until a complete text string is transferred. Upon completion, the received stream is compared with what was transmitted to ensure data veracity.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/serial/serialTests.h** (**SERIAL_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SERIAL_TESTS_VALID_FLAG**
baudRate = **SERIAL_IOCTL_P_BAUD_9600**
charLength = **SERIAL_IOCTL_P_CHAR_8**
stopBits = **SERIAL_IOCTL_P_STOP_1**
parity = **SERIAL_IOCTL_P_NO_PARITY**
mode = **SERIAL_IOCTL_P_POLLED**
loopBack = **SERIAL_IOCTL_P_LOOP_ON**
modem = **SERIAL_IOCTL_P_MODEM_OFF**
duration = **0**
readTimeout = **0**
writeTimeout = **0**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SERIAL_INTERNAL_LOOPBACK_POLL** test. The user must use the definition of **SERIAL_TESTS_VALID_FLAG**.

baudRate

determines the rate at which the UART transmits the serial data. The value in this parameter should match the receive rate of the device that is connected to the port under test. The user should use one of the baud rate definitions listed below. These values range from **300** to **115200** baud.

SERIAL_IOCTL_P_BAUD_115200

SERIAL_IOCTL_P_BAUD_56000

SERIAL_IOCTL_P_BAUD_38400

SERIAL_IOCTL_P_BAUD_19200

SERIAL_IOCTL_P_BAUD_9600

SERIAL_IOCTL_P_BAUD_7200

SERIAL_IOCTL_P_BAUD_4800

SERIAL_IOCTL_P_BAUD_3600

SERIAL_IOCTL_P_BAUD_2400

SERIAL_IOCTL_P_BAUD_2000

SERIAL_IOCTL_P_BAUD_1800

SERIAL_IOCTL_P_BAUD_1200

SERIAL_IOCTL_P_BAUD_600

SERIAL_IOCTL_P_BAUD_300

The most commonly used parameter is **SERIAL_IOCTL_P_BAUD_9600**.

charLength

determines the number of bits to be contained in each byte of the serial data stream. Like the *baudRate* selection, this parameter should be set to match the serial device that is connected. The user should use one of the following definitions for this parameter:

SERIAL_IOCTL_P_CHAR_5

SERIAL_IOCTL_P_CHAR_6

SERIAL_IOCTL_P_CHAR_7

SERIAL_IOCTL_P_CHAR_8

The more commonly used of these parameters is **SERIAL_IOCTL_P_CHAR_8**.

stopBits

determines the number of stop bits to be used for each character transmission. The user should provide one of the following definitions in this parameter:

SERIAL_IOCTL_P_STOP_1

SERIAL_IOCTL_P_MULTI_STOP

The most commonly used of these definitions is **SERIAL_IOCTL_P_STOP_1**.

parity

determines how the output character's parity is set. This parameter should be set to match the external device's parity configuration. The user should provide one of the following definitions:

SERIAL_IOCTL_P_NO_PARITY

SERIAL_IOCTL_P_EVEN_PARITY

SERIAL_IOCTL_P_ODD_PARITY

The most commonly used of these definitions is **SERIAL_IOCTL_P_NO_PARITY**.

mode

allows the user to determine if the test runs using polling or interrupts. This test should have this parameter set to **SERIAL_IOCTL_P_POLLED**.

loopBack

determines if the test runs using the internal loopback capability of the UART. This test should have this parameter set to **SERIAL_IOCTL_P_LOOP_ON**.

modem

determines if the hardware handshake lines are used during the test. This test should have this parameter set to **SERIAL_IOCTL_P_MODEM_OFF**.

duration

is set to **0** by default and is ignored otherwise.

readTimeout

can be used to modify the amount of time that the test waits for an input to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

writeTimeout

is used to modify the amount of time the test waits for an output to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)

BIT_DATA_MISCOMPARE—data miscompare on write and read sequence
BIT_SERIAL_RECV_FAULT—serial receiver fault (parity, frame, overrun)
BIT_TEST_TIMED_OUT—the test timed out prior to completion

UART Internal Loopback Interrupt Mode Test

[**BIT_SERIAL_INTERNAL_LOOPBACK_INT**]
tests/serial/serialTests.h

The **tests/serial/serialTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SERIAL_INTERNAL_LOOPBACK_INT**] test.

This test verifies that the UART can properly transmit and receive a data stream in interrupt mode.

Test Description

This test configures the UART to transmit and receive data at a baud rate that is determined by the input test parameters. Upon configuration, the test sends a character data stream that is presented at the receiver. The data is transmitted and received until a complete text string is transferred. Upon completion, the received stream is compared with what was transmitted to ensure data veracity.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/serial/serialTests.h** (**SERIAL_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SERIAL_TESTS_VALID_FLAG**
baudRate = **SERIAL_IOCTL_P_BAUD_9600**
charLength = **SERIAL_IOCTL_P_CHAR_8**
stopBits = **SERIAL_IOCTL_P_STOP_1**
parity = **SERIAL_IOCTL_P_NO_PARITY**
mode = **SERIAL_IOCTL_P_INTRPT**
loopBack = **SERIAL_IOCTL_P_LOOP_ON**
modem = **SERIAL_IOCTL_P_MODEM_OFF**
duration = **0**
readTimeout = **0**
writeTimeout = **0**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SERIAL_INTERNAL_LOOPBACK_INT** test. The user must use the definition of **SERIAL_TESTS_VALID_FLAG**.

baudRate

determines the rate at which the UART transmits the serial data. The value in this parameter should match the receive rate of the device that is connected to the port under test. The user should use one of the baud rate definitions listed below. These values range from **300** to **115200** baud.

SERIAL_IOCTL_P_BAUD_115200

SERIAL_IOCTL_P_BAUD_56000

SERIAL_IOCTL_P_BAUD_38400

SERIAL_IOCTL_P_BAUD_19200

SERIAL_IOCTL_P_BAUD_9600

SERIAL_IOCTL_P_BAUD_7200

SERIAL_IOCTL_P_BAUD_4800

SERIAL_IOCTL_P_BAUD_3600

SERIAL_IOCTL_P_BAUD_2400

SERIAL_IOCTL_P_BAUD_2000

SERIAL_IOCTL_P_BAUD_1800

SERIAL_IOCTL_P_BAUD_1200

SERIAL_IOCTL_P_BAUD_600

SERIAL_IOCTL_P_BAUD_300

The most commonly used parameter is **SERIAL_IOCTL_P_BAUD_9600**.

charLength

determines the number of bits to be contained in each byte of the serial data stream. This parameter should be set to match the serial device that is connected. The user should use one of the following definitions for this parameter:

SERIAL_IOCTL_P_CHAR_5

SERIAL__IOCTL_PCHAR_6

SERIAL__IOCTL_PCHAR_7

SERIAL__IOCTL_PCHAR_8

The more commonly used of these parameters is **SERIAL_IOCTL_P_CHAR_8**.

stopBits

determines the number of stop bits to be used for each character transmission. The user should provide one of the following definitions in this parameter:

SERIAL_IOCTL_P_STOP_1

SERIAL_IOCTL_P_MULTI_STOP

The most commonly used of these definitions is **SERIAL_IOCTL_P_STOP_1**.

parity

determines how the output character's parity is set. This parameter should be set to match the external device's parity configuration. The user should provide one of the following definitions:

SERIAL_IOCTL_P_NO_PARITY

SERIAL_IOCTL_P_EVEN_PARITY

SERIAL_IOCTL_P_ODD_PARITY

The most commonly used of these definitions is **SERIAL_IOCTL_P_NO_PARITY**.

mode

allows the user to determine if the test runs using polling or interrupts. This test should have this parameter set to **SERIAL_IOCTL_P_INTRPT**.

loopBack

determines if the test runs using the internal loopback capability of the UART. This test should have this parameter set to **SERIAL_IOCTL_P_LOOP_ON**.

modem

determines if the hardware handshake lines are used during the test. This test should have this parameter set to **SERIAL_IOCTL_P_MODEM_OFF**.

duration

is set to **0** by default and is ignored otherwise.

readTimeout

is used to modify the amount of time the test waits for an input to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

writeTimeout

is used to modify the amount of time the test waits for an output to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence
BIT_SERIAL_RECV_FAULT—serial receiver fault (parity, frame, overrun)
BIT_TEST_TIMED_OUT—the test timed out prior to completion

UART Internal Loopback DMA Mode Test

[**BIT_SERIAL_INTERNAL_LOOPBACK_DMA**]
tests/serial/serialTests.h

The **tests/serial/serialTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SERIAL_INTERNAL_LOOPBACK_DMA**] test.

This test verifies that the UART can properly transmit and receive a data stream using interrupts.

Test Description

This test configures the UART to transmit and receive data at a baud rate that is determined by the input test parameters. Upon configuration, the test sends a character data stream that is presented at the receiver. A data stream is transmitted and then the data is read from a received data queue. Upon completion, the received stream is compared with what was transmitted to ensure data veracity.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/serial/serialTests.h** (**SERIAL_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SERIAL_TESTS_VALID_FLAG**
baudRate = **SERIAL_IOCTL_P_BAUD_9600**
charLength = **SERIAL_IOCTL_P_CHAR_8**
stopBits = **SERIAL_IOCTL_P_STOP_1**
parity = **SERIAL_IOCTL_P_NO_PARITY**
mode = **SERIAL_IOCTL_P_DMA**
loopBack = **SERIAL_IOCTL_P_LOOP_ON**
modem = **SERIAL_IOCTL_P_MODEM_OFF**
duration = **0**
readTimeout = **0**
writeTimeout = **0**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SERIAL_INTERNAL_LOOPBACK_DMA** test. The user must use the definition of **SERIAL_TESTS_VALID_FLAG**.

baudRate

determines the rate at which the UART transmits the serial data. The value in this parameter should match the receive rate of the device that is connected to the port under test. The user should use one of the baud rate definitions listed below. These values range from **300** to **115200** baud.

SERIAL_IOCTL_P_BAUD_115200

SERIAL_IOCTL_P_BAUD_56000

SERIAL_IOCTL_P_BAUD_38400

SERIAL_IOCTL_P_BAUD_19200

SERIAL_IOCTL_P_BAUD_9600

SERIAL_IOCTL_P_BAUD_7200

SERIAL_IOCTL_P_BAUD_4800

SERIAL_IOCTL_P_BAUD_3600

SERIAL_IOCTL_P_BAUD_2400

SERIAL_IOCTL_P_BAUD_2000

SERIAL_IOCTL_P_BAUD_1800

SERIAL_IOCTL_P_BAUD_1200

SERIAL_IOCTL_P_BAUD_600

SERIAL_IOCTL_P_BAUD_300

The most commonly used parameter is **SERIAL_IOCTL_P_BAUD_9600**.

charLength

determines the number of bits to be contained in each byte of the serial data stream. This parameter should be set to match the serial device that is connected. The user should use one of the following definitions for this parameter:

SERIAL_IOCTL_P_CHAR_5

SERIAL_IOCTL_P_CHAR_6

SERIAL_IOCTL_P_CHAR_7

SERIAL_IOCTL_P_CHAR_8

The more commonly used of these parameters is **SERIAL_IOCTL_P_CHAR_8**.

stopBits

determines the number of stop bits to be used for each character transmission. The user should provide one of the following definitions in this parameter:

SERIAL_IOCTL_P_STOP_1

SERIAL_IOCTL_P_MULTI_STOP

The most commonly used of these definitions is **SERIAL_IOCTL_P_STOP_1**.

parity

determines how the output character's parity is set. This parameter should be set to match the external device's parity configuration. The user should provide one of the following definitions:

SERIAL_IOCTL_P_NO_PARITY

SERIAL_IOCTL_P_EVEN_PARITY

SERIAL_IOCTL_P_ODD_PARITY

The most commonly used of these definitions is **SERIAL_IOCTL_P_NO_PARITY**.

mode

allows the user to determine if the test runs using polling or interrupts. This test should have this parameter set to **SERIAL_IOCTL_P_DMA**.

loopBack

determines if the test runs using the internal loopback capability of the UART. This test should have this parameter set to **SERIAL_IOCTL_P_LOOP_ON**.

modem

determines if the hardware handshake lines are used during the test. This test should have this parameter set to **SERIAL_IOCTL_P_MODEM_OFF**.

duration

is set to **0** by default and is ignored otherwise.

readTimeout

is used to modify the amount of time the test waits for an input to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

writeTimeout

is used to modify the amount of time the test waits for an output to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported

BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence
BIT_SERIAL_RECV_FAULT—serial receiver fault (parity, frame, overrun)
BIT_TEST_TIMED_OUT—the test timed out prior to completion

UART External Loopback with Modem Controls Test

[**BIT_SERIAL_EXTERNAL_LOOPBACK**]
tests/serial/serialTests.h

The **tests/serial/serialTests.h** file defines the parameter structures and fields mentioned in the [**BIT_SERIAL_EXTERNAL_LOOPBACK**] test.

This test verifies that the UART can properly transmit and receive a data stream using interrupts and hardware handshake control.

Test Description

This test configures the UART to transmit and receive data at a baud rate that is determined by the input test parameters. Upon configuration, the test sends a character data stream that is presented at the receiver. The data stream is transmitted and then the data is read from a received data queue. Upon completion, the received stream is compared with what was transmitted to ensure data veracity.

Affected Peripheral Devices

If an external device is connected to the serial port under test, it must be disconnected. An external loopback cable or adaptor must be attached to the port. As a result, this test cannot be run successfully if the test is controlled from a serial console.

Required Test Equipment

An external loopback cable or adaptor must be provided. This cable must physically connect both the transmit and receive signals together. Also, the Ready To Send (RTS) and Clear To Send (CTS) signals must be physically connected.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/serial/serialTests.h** (**SERIAL_TEST_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SERIAL_TESTS_VALID_FLAG**
baudRate = **SERIAL_IOCTL_P_BAUD_9600**
charLength = **SERIAL_IOCTL_P_CHAR_8**
stopBits = **SERIAL_IOCTL_P_STOP_1**
parity = **SERIAL_IOCTL_P_NO_PARITY**
mode = **SERIAL_IOCTL_P_INTRPT**
loopBack = **SERIAL_IOCTL_P_LOOP_OFF**
modem = **SERIAL_IOCTL_P_MODEM_ON**
duration = **0**
readTimeout = **0**
writeTimeout = **0**

The contents of the structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SERIAL_EXTERNAL_LOOPBACK** test. The user must use the definition of **SERIAL_TESTS_VALID_FLAG**.

baudRate

determines the rate at which the UART transmits the serial data. The value in this parameter should match the receive rate of the device that is connected to the port under test. The user should use one of the baud rate definitions listed below. These values range from **300** to **115200** baud.

SERIAL_IOCTL_P_BAUD_115200

SERIAL_IOCTL_P_BAUD_56000

SERIAL_IOCTL_P_BAUD_38400

SERIAL_IOCTL_P_BAUD_19200

SERIAL_IOCTL_P_BAUD_9600

SERIAL_IOCTL_P_BAUD_7200

SERIAL_IOCTL_P_BAUD_4800

SERIAL_IOCTL_P_BAUD_3600

SERIAL_IOCTL_P_BAUD_2400

SERIAL_IOCTL_P_BAUD_2000

SERIAL_IOCTL_P_BAUD_1800

SERIAL_IOCTL_P_BAUD_1200

SERIAL_IOCTL_P_BAUD_600

SERIAL_IOCTL_P_BAUD_300

The most commonly used parameter is **SERIAL_IOCTL_P_BAUD_9600**.

charLength

determines the number of bits to be contained in each byte of the serial data stream. This parameter should be set to match the serial device that is connected. The user should use one of the following definitions for this parameter:

SERIAL_IOCTL_P_CHAR_5

SERIAL_IOCTL_P_CHAR_6

SERIAL_IOCTL_P_CHAR_7

SERIAL_IOCTL_P_CHAR_8

The more commonly used of these parameters is **SERIAL_IOCTL_P_CHAR_8**.

stopBits

determines the number of stop bits to be used for each character transmission. The user should provide one of the following definitions in this parameter:

SERIAL_IOCTL_P_STOP_1

SERIAL_IOCTL_P_MULTI_STOP

The most commonly used of these definitions is **SERIAL_IOCTL_P_STOP_1**.

parity

determines how the output character's parity is set. This parameter should be set to match the external device's parity configuration. The user should provide one of the following definitions:

SERIAL_IOCTL_P_NO_PARITY

SERIAL_IOCTL_P_EVEN_PARITY

SERIAL_IOCTL_P_ODD_PARITY

The most commonly used of these definitions is **SERIAL_IOCTL_P_NO_PARITY**.

mode

allows the user to determine if the test runs using polling or interrupts. This test should have this parameter set to **SERIAL_IOCTL_P_INTRPT**.

loopBack

determines if the test runs using the internal loopback capability of the UART. This test should have this parameter set to **SERIAL_IOCTL_P_LOOP_OFF**.

modem

determines if the hardware handshake lines are used during the test. This test should have this parameter set to **SERIAL_IOCTL_P_MODEM_ON**.

duration

is set to **0** by default and is ignored otherwise.

readTimeout

is used to modify the amount of time the test waits for an input to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

writeTimeout

is used to modify the amount of time the test waits for an output to complete. If the user does not desire to change the time out value, a **0** should be provided to use the default. The default value is **1 second**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence

BIT_SERIAL_RECV_FAULT—serial receiver fault (parity, frame, overrun)
BIT_TEST_TIMED_OUT—the test timed out prior to completion

VME Bridge Tests 12

This chapter provides descriptions and requirements for the following VME bridge tests:

VME Bridge Tests

The table below highlights each tests’ string, header file, and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_VME_REGISTER]	tests/vme/vmeTests.h	VME_REGISTER_PARAMS
[BIT_VME_LOCMON]	tests/vme/vmeTests.h	VME_LOCMON_PARAMS
[BIT_VME_RW_TARGET]	tests/vme/vmeTests.h	VME_RW_TARGET_PARAMS
[BIT_VME_SHORT_IO]	tests/vme/vmeTests.h	VME_RW_TARGET_PARAMS
[BIT_VME_STANDARD_IO]	tests/vme/vmeTests.h	VME_RW_TARGET_PARAMS
[BIT_VME_EXTENDED_IO]	tests/vme/vmeTests.h	VME_RW_TARGET_PARAMS
[BIT_VME_CSR]	tests/vme/vmeTests.h	VME_RW_TARGET_PARAMS
[BIT_VME_DMA]	tests/vme/vmeTests.h	VME_RW_TARGET_PARAMS

During any test install, all VME bridge registers are saved. Then all functional (non-PCI configuration) registers are set to **0**, if possible. During the save/zero process, the MBIT probe routines that trap/report exceptions are used, and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** is configured to generate an exception on PCI bus errors (either master or target abort). Thus, all registers (and write-only registers in particular) are actually tested for accessibility during an install.

If the test is a target read/write test, and the caller has not supplied an I/O buffer, input and output buffers are allocated large enough to accommodate the desired transfer.

If DMA is active, test installation fails with **BIT_INSTALL_DEV_FAILED**, under the assumption that another task is actively using the bridge. There is currently no way to prevent concurrent access to the VME bridge control registers.

Test installation disables all target (PCI memory space) and slave (VME) access to the VME bridge, rendering it invisible to the processor and other VME masters (except for PCI I/O access to the registers). It also disables all PCI and VME interrupt inputs/outputs (handlers/generators), and clears any pending PCI or VME interrupts. Thus, if the bridge is configured as the VME SYSCON, it does not respond to VME bus interrupts during the test. It does, however, continue to act as the VME bus arbiter, so other VME masters/slaves can communicate with each other.

The test installation also installs a test-specific interrupt handler for the LM0-3 interrupts and DMA interrupts. The installed handler posts a semaphore if/when any of these interrupts occur. If any of the semaphores cannot be created, the test returns **BIT_INIT_ALLOCATION_ERROR**.

For tests involving programmed I/O (PIO) to/from the VME bus, (LOCMON and Target PIO), the test installation always creates its own mapping from a local (CPU) address to a PCI address, through the VME bridge to VME (even if the range that includes the target address was previously mapped by the VME bridge). The install assumes that at least a 128KB window in PCI memory space has been allocated by the board support package (BSP), mapped through the memory management unit (MMU) to a local address, and mapped from that local address by the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** to a PCI address. The static

BSP definitions for **BIT_VME_XXX_MSTR_LOCAL** and **BIT_VME_XXX_MSTR_SIZE** are used internally to either select a local CPU address for mapping a target, or to validate a user-supplied local address. The corresponding **BIT_VME_XXX_MSTR_BUS** values in PCI space are used to program the VME bridge translation registers. These mappings are only enabled for the duration of an I/O subtest execution (disabled between iterations).

BIT_VME_XXX_MSTR_LOCAL:	**BIT_VME_A16_MSTR_LOCAL** **BIT_VME_A24_MSTR_LOCAL** **BIT_VME_A32_MSTR_LOCAL** **BIT_VME_LM_MSTR_LOCAL** **BIT_VME_CRCSR_MSTR_LOCAL**
BIT_VME_XXX_MSTR_SIZE:	**BIT_VME_A16_MSTR_SIZE** **BIT_VME_A24_MSTR_SIZE** **BIT_VME_A32_MSTR_SIZE** **BIT_VME_LM_MSTR_SIZE** **BIT_VME_CRCSR_MSTR_SIZE**
BIT_VME_XXX_MSTR_BUS:	**BIT_VME_A16_MSTR_BUS** **BIT_VME_A24_MSTR_BUS** **BIT_VME_A32_MSTR_BUS** **BIT_VME_LM_MSTR_BUS** **BIT_VME_CRCSR_MSTR_BUS**

If the BSP or user applications make dynamic modifications to the default CPU to PCI mappings described by the above parameters, or the kernel on which the MBIT test suite is loaded used different static values than the BSP against which MBIT was built, the VME I/O subtests may fail and/or overwrite system/application memory.

Also note that during VME subtest installation/execution, attempts by other tasks to access the **BIT_PCI_TO_VME_BRIDGE** and/or the VME bus cannot be prevented. During VME subtest execution, the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** remains configured to generate exceptions upon PCI or VME bus error while PCI/VME address ranges have been disabled or remapped elsewhere. If another task is accessing VME resources while a VME subtest is in progress, that task either receives an exception (likely fatal), reads erroneous data, or possibly performs I/O to an unintended VME target. There is also a slight possibility that the VME subtest itself receives an exception. The VME subtest could report an erroneous test failure due to an I/O or data mis-compare error.

If a user pattern list is provided, test installation performs a probe read of the pattern list beginning/end. If a user buffer is provided, test install performs a write or read probe of the start/end of the buffer to ensure it is accessible to the calling task.

For tests involving writing/reading default bit patterns, the following 32-bit patterns are used: 0xAAAAAAAA, 0x55555555, 0xFFFF0000, 0x0000FFFF, 0xFF00FF00, 0x00FF00FF, 0xF0F0F0F0, 0x33333333, 0xCCCCCCCC, 0x7F7F7F7F, 0xFFFFFFFF and 0x00000000.

For target I/O tests, an output buffer is filled with bytes, half words or long words containing the low 1, 2 or 4-bytes of each pattern. The output/input buffers are accessed 1, 2 or 4-bytes at a time, and written/read to/from the target 1, 2, 4 or 8-bytes at a time (8-bytes for DMA only; though the DMA test attempts to pack 64-bit bursts to/from VME, the PCI data interface to host memory is typically only 32-bits wide).

For DMA target tests, the DMA engine is configured to interrupt on completion, successful or otherwise. Also for target I/O tests, if the number of bytes to be written or read would generate more than two I/O accesses at the specified width, a write and/or read probe is done to the start and end addresses of the VME range prior to beginning the actual buffer transfer.

Upon test uninstall, register contents are restored, interrupt handlers removed, and **BIT_LOCAL_BUS_TO_PCI_BRIDGE** configuration restored. If any of these actions cannot be performed, the test may return **BIT_SYS_RESTORE_FAILED**, **BIT_BUS_ERROR**, or any other **BIT_FAULT** value returned by a lower level routine.

VME Bridge Register Read/Write Test

[**BIT_VME_REGISTER**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_REGISTER**] test.

This test verifies that the chip's registers can be written and read. It also tries to verify that all read/write bits can be toggled both 0 to 1 and 1 to 0, and that there is no interaction between bits with respect to this ability.

Even though basic register I/O was verified during test install, all test I/O to the registers is also done with probe routines and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** configured to trap PCI bus errors and report failure. This is done in the event that a particular bit pattern triggers such a response on a read-only register whose contents are not verified.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

This test sets the VME bridge functional read/write registers (all except the PCI configuration registers) to predefined values (using an array of 32-bit patterns) and reads the register back to verify that the correct bits are set or cleared. In read/write registers with reserved bits of either 0 or 1, such bits are ignored (always written with 0, and contents never verified). Likewise, bits where a read or write clears a 1 are not verified for write/read (they are written with 0). All the remaining bits in each read/write register are treated like memory, and are tested to ensure that they can be set to all ones, all zeroes, and alternating bit patterns to ensure that there are no stuck or shorted bits (whether adjacent or not).

For read-only registers where all bits are reserved or have a known, fixed value, the test verifies that each has the correct value. If any bits are not reserved or fixed, the test merely verifies that the register can be read without fault. Write-only registers are not re-tested at present (only exercised during test install).

In some cases, there are register bits that cannot be changed because they may cause the system to go into an indeterminate state (or would, for example, reset the VME or PCI bus). These bits are written only with 0 to protect the system integrity, unless that would cause a side effect, in which case the register is ignored.

If an I/O error occurs during register test read or test write, the test returns **BIT_VME_REGISTER_IO_FAULT**.

If a data miscompare error occurs on a read/write register, or a read-only register with only reserved/fixed bits does not return the expected value, the test returns **BIT_VME_REGISTER_DATA_FAULT**.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices as described in *VME Bridge Tests* on page 12-1.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_REGISTER_PARAMS**).

This test's default parameter is as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**.

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_REGISTER** test. The user must use the definition of **VME_REGISTER_TESTS_VALID_FLAG**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_REGISTER_IO_FAULT—VME bridge register write/read access fault
BIT_VME_REGISTER_DATA_FAULT—VME bridge register write/read data miscompare
BIT_VME_REGISTER_VALUE_FAULT—VME bridge register read value fault

VME Bridge Location Monitor Test

[**BIT_VME_LOCMON**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_LOCMON**] test.

This test verifies proper operation of the location monitor (LM) functions of the VME bridge. If the LM functionality of the bridge has been configured/enabled by the underlying operating system, the test can operate using the current configuration. The LM is a broadcast interrupt facility for the VMEbus used for synchronization/notification (such as mailbox notification) between VME masters and slaves.

The **BIT_LOCAL_BUS_TO_PCI_BRIDGE** is left configured (as by the test install) to generate an exception on PCI master or target abort.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test takes all parameters from the caller, the LM number to exercise (0-3), and optionally a VME and locally mapped (CPU) base address at which to access the LM. Here, base implies to the base to which the LM number (0-3) is applied to generate an access; the address need not be 4KB or 64KB aligned. Suitable 4KB windows are mapped that incorporate the base address. If the LM slave interface (that is, the VME address/space at which the LM would respond) was not configured and enabled prior to the test, the user must supply a valid VME base address to access the LM. Otherwise, the *Addr* parameters may be given special values that cause the test to use the existing LM slave configuration.

The test enables the four LM interrupts to PCI (LM interrupts onto VME are left disabled). The actual LM local (CPU) address is calculated from the LM number (for the Universe II, it is placed in bits 4:3 of the address).

The actual test consists of a read access to the local LM address, using a MBIT probe routine configured to return an explicit error if an exception occurs. If an exception occurs, the test returns **BIT_VME_LOCMON_IO_FAULT**. If the access succeeds, but the correct interrupt does not follow within 500 ms, the test returns **BIT_VME_LOCMON_INT_FAULT**.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices as described in *VME Bridge Tests* on page 12-1. If a VME address is supplied that corresponds to another VME bus host, that host may also be affected.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_LOCMON_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **0**
vmeAddr = **BIT_VME_ASSIGN_ADDR**
locAddr = **BIT_VME_ASSIGN_ADDR**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_LOCMON** test. The user must use the definition of **VME_LOCMON_TESTS_VALID_FLAG**.

vmeSpace

is ignored if *vmeAddr* is **BIT_VME_ASSIGN_ADDR**. Otherwise, a VME address space is valid for the LM (Short, Standard or Extended I/O; CR/CSR is not valid for LM). *vmeSpace* and *vmeAddr* are used to map the VME address/space at which the LM should respond.

vmeAddr

is either a VME address (valid for *vme_Space*) at which the LM should be read (modified by the *vmeLocMon* parameter) or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test attempts to map the LM to whatever address/space it was at prior to the start of the test, and uses the lowest address of that 4KB range (modified by the *vmeLocMon* parameter) as the read address. If the LM slave interface was not previously enabled (even if it was mapped), specifying **BIT_VME_ASSIGN_ADDR** results in an error return of **BIT_VME_LOCMON_IO_FAULT** (since there is no safe default). If *locAddr* is given, *vmeAddr* must be given; it cannot be assigned by the test in this case. If an address is given, it must be module 32 (32 byte aligned) to allow for ORing in the *vmeLocMon* value to bits 4:3.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space, or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested. If a specific *locAddr* is given, a specific *vmeAddr* must be given as well. If both *locAddr* and *vmeAddr* are supplied, the test requires that they be byte-aligned in the first 64KB.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable

BIT_VME_LOCMON_FAULT—VME bridge location monitor (mailbox) interrupt fault
BIT_VME_LOCMON_IO_FAULT—VME bridge location monitor (mailbox) access fault

VME General-Purpose Target I/O Test

[**BIT_VME_RW_TARGET**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_RW_TARGET**] test.

This test allows flexible, parameterized exercising of the VME bridge target interface.

This test has the following default test values:

Iteration:	1
Duration:	6000
Control:	HALT_ON_ERROR

Test Description

In this context, target implies the mapping and translation of CPU to PCI programmed I/O (PIO) cycles (where the CPU is the PCI master and the VME bridge is the PCI target) into VME bus cycles where the VME bridge is the master, and the slave interface of some other host is the VME target. PIO is done using the MBIT probe routines to trap I/O errors. This subtest can also exercise the DMA capabilities of the VME bridge, where the bridge is both PCI and VME master, acting independently of the CPU.

The next five tests in this section are essentially subsets of the **BIT_VME_RW_TARGET** test (all use the same test parameter structure). Each test exercises a specific VME address space or DMA and defaults some of the parameters, trading flexibility for simplicity, and being able to refer to a set of tests for a particular address space by a unique name/enumeration.

If DMA is not requested (neither *writeWidth* nor *readWidth* is **BIT_VME_RDWR_DMA**), then *vmeSpace*, *vmeAddr* and max (*readBytes*, *writeBytes*) are used to map the VME address range to a local address range (*locAddr*) where PIO can be performed to access the target. The actual address range mapped is 64KB aligned above and below the target range.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices as described in *VME Bridge Tests* on page 12-1. The target VME host is affected.

Required Test Equipment

There must be at most one VME host in the system, besides the host under test, that is capable of responding at the user-supplied VME address space/range. There must be a functional VME arbiter in the system.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_RW_TARGET_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **BIT_VME_SPACE_EXTIO**
vmeAddr = **(UINT8 *)BIT_PLAYPEN_A32_SLV_BUS**
locAddr = **(UINT8 *)((BIT_VME_A32_MSTR_LOCAL**
+ BIT_VME_A32_MSTR_SIZE)
– (BIT_PLAYPEN_SLV_SIZE))
bufferPtr = **NULL**
tgtValid = **TRUE**
readBytes = **0x1FF00**
readWidth = **BIT_VME_RDWR_8**
writeBytes = **0x20000**
writeWidth = **BIT_VME_RDWR_32**
patternPtr = **NULL**
numPatterns = **0**

rwMask = **0xF1E3C78F**
vmeMaxDw = **0**
vmeAm = **0**
vmeImage = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_RW_TARGET** test. The user must use the definition of **VME_TARGET_TESTS_VALID_FLAG**.

vmeSpace

is a VME address space valid for the proposed target host.

vmeAddr

is a VME address (valid for *vmeSpace*) at which the target responds. This is mapped to *locAddr* (unless both *readWidth* and *writeWidth* are **BIT_VME_RDWR_DMA**). *vmeAddr* + (max (*readBytes*, *writeBytes*) – 1) must be valid for *vmeSpace*, and may not wrap back to **0**.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space, or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested.

bufferPtr

is a local memory address accessible to the caller to be used as the source or sink for data to/from the target host. If **NULL**, the test allocates separate internal buffers for input/output. If *bufferPtr* is given, the test does not write canned patterns, and does not verify that the read data matches what was written. *bufferPtr* + (max (*readBytes*, *writeBytes*) – 1) must be within the address space accessible to the caller. Note that *rwMask* is not applied to data written from a user-supplied buffer. It is assumed that the data in the buffer is ready for delivery to the target. Only one of *bufferPtr* or *patternPtr* may be provided; the other must be **NULL**.

tgtValid

defines the target address range. If **TRUE**, the target address range is expected to respond. An error is returned by the test if a target I/O error occurs (PCI or VME). If **FALSE**, the target address range is not expected to respond. An error is returned by the test if a target I/O error does not occur.

writeWidth

represents the number of bytes to write at one time to the target using programmed I/O (PIO). If **BIT_VME_RDWR_DMA** (8 bytes), then DMA is used to start a write transfer of *writeBytes* bytes using 64-bit VME block transfers if possible. If patterns are being written, *writeWidth* (but never more than 4) bytes worth of the pattern are ANDed with *rwMask* and written to an internal write buffer. The order in which multiple (2 or 4) bytes from the pattern are written to the buffer is architecture-dependent. The canned patterns provided are constructed such that whatever byte order is used, all bits in the target location (subject to *rwMask*) are toggled.

readWidth

represents the number of bytes to read at one time from the target using programmed I/O (PIO). If **BIT_VME_RDWR_DMA** (8 bytes), then DMA is used to start a read transfer of *readBytes* bytes using 64-bit VME block transfers if possible. *writeWidth* and *readWidth* need not be the same; DMA writes could be combined with 1-byte PIO reads.

writeBytes

represents the number of bytes to write to the target (max 128KB). If **0**, a read-only test is done. The bytes written may be patterns (internal or external) or data from *bufferPtr*. If patterns are being written and read, a verify is done on the read data to ensure it matches what was written; *writeBytes* must be >= *readBytes* so that the verify phase can be safely performed.

readBytes

represents the number of bytes to read back (or simply read if *writeBytes* is **0**) from the target (max 128K). If **0**, a write-only test is performed. If patterns are being used, *readBytes* must be <= *writeBytes* if *writeBytes* is > **0**. If patterns are being written, *readBytes* worth of pattern data is verified against what was written.

patternPtr

is a pointer to an array of 32-bit values to be used as bit patterns to write/read to/from the target. If **NULL**, canned patterns, as described in *VME Bridge Tests* on page 12-1, are used. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

numPatterns

represents the number of patterns in *patternPtr*. If **0**, *patternPtr* must be **NULL**.

rwMask

is a bit mask to be applied to (that is, ANDed with) each 32-bit pattern before it is copied to the internal write buffer. Only the low *writeWidth* bits of the mask are actually used. The name *rwMask* is really a misnomer; the mask is only applied to write data. Read data from the target is copied directly to the internal or user-supplied buffer. Likewise, *rwMask* is not applied to data written from a user-supplied buffer.

vmeMaxDw

must be set to **0**.

vmeAM

must be set to **0**.

vmeImage

must be set to **0**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_RW_TARGET_IO_FAULT—VME write/read target access fault
BIT_VME_RW_TARGET_DATA_FAULT—VME write/read target data miscompare
BIT_VME_RW_TARGET_INT_FAULT—VME write/read target interrupt fault
BIT_VME_NO_TARGET_RW_IO_FAULT—VME target write/read to non-existent address succeeded

VME Short Target I/O Test

[**BIT_VME_SHORT_IO**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_SHORT_IO**] test.

This test exercises the target interface of the VME bridge with translation of PCI addresses to the VME Short I/O (A16) address space.

This test has the following default test values:

Iteration:	1
Duration:	6000
Control:	HALT_ON_ERROR

Test Description

This test differs from the others in this series, in that it is possible to specify (in *readWidth* or *writeWidth*) the number of bytes to write/read, but only one write or read (of the specified width) is performed.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices, as described in *VME Bridge Tests* on page 12-1. The target VME host is affected.

Required Test Equipment

There must be at most one VME host in the system, besides the host under test, that is capable of responding at the user-supplied Short I/O (A16) VME address space/range. There must be a functional VME arbiter in the system.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_RW_TARGET_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **BIT_VME_SPACE_SHORTIO**
vmeAddr = **(UINT8 *)BIT_PLAYPEN_A16_SLV_BUS**
locAddr = **(UINT8 *)BIT_VME_A16_MSTR_LOCAL**
bufferPtr = **NULL**
tgtValid = **TRUE**
readBytes = **2**
readWidth = **BIT_VME_RDWR_8**
writeBytes = **3**
writeWidth = **BIT_VME_RDWR_8**
patternPtr = **NULL**
numPatterns = **0**
rwMask = **0xE7**
vmeMaxDw = **0**

vmeAm = **0**
vmeImage = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_RW_TARGET** test. The user must use the definition of **VME_TARGET_TESTS_VALID_FLAG**.

vmeSpace

is a VME address space valid for the proposed target host.

vmeAddr

is a 16-bit VME address (0x0-0xFFFF) at which the target responds. It is mapped to *locAddr*. *vmeAddr* + (max (*readBytes*, *writeBytes*) – 1) must be valid for *vmeSpace* and may not wrap back to **0**.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested.

bufferPtr

is a local memory address accessible to the caller to be used as the source or sink for data to/from the target host. If **NULL**, the test allocates separate internal buffers for input/output. If *bufferPtr* is given, the test does not write canned patterns and does not verify that the read data matches what was written. *bufferPtr* + (max (*readBytes*, *writeBytes*) – 1) must be within the address space accessible to the caller. Note that *rwMask* is not applied to data written from a user-supplied buffer. It is assumed that the data in the buffer is ready for delivery to the target. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

tgtValid

defines the target address range. If **TRUE**, the target address range is expected to respond. An error is returned by the test if a target I/O error occurs (PCI or VME). If **FALSE**, the target address range is not expected to respond. An error is returned by the test if a target I/O error does not occur.

writeWidth

represents the number of bytes (1, 2 or 4) to write at one time to the target using programmed I/O (PIO). If patterns are being written, *writeWidth* bytes worth of the pattern are ANDed with *rwMask* and written to an internal write buffer. The order in which multiple (2 or 4) bytes from the pattern are written to the buffer is architecture-dependent. The canned patterns provided are constructed such that whatever byte order is used, all bits in the target location (subject to *rwMask*) are toggled.

readWidth

represents the number of bytes (1, 2 or 4) to read at one time from the target using programmed I/O (PIO). *writeWidth* and *readWidth* need not be the same; 4-byte PIO writes could be combined with 1-byte PIO reads.

writeBytes

represents the number of bytes to write to the target (max 128KB). If **0**, a read-only test is done. The bytes written may be patterns (internal or external) or data from *bufferPtr*. If patterns are being written and read, a verify is done on the read data to ensure it matches what was written; *writeBytes* must be >= *readBytes* so that the verify phase can be safely performed.

readBytes

represents the number of bytes to read back (or simply read if *writeBytes* is **0**) from the target (max 128K). If **0**, a write-only test is performed. If patterns are being used, *readBytes* must be <= *writeBytes* if *writeBytes* is > **0**. If patterns are being written, *readBytes* worth of pattern data is verified against what was written.

patternPtr

is a pointer to an array of 32-bit values to be used as bit patterns to write/read to/from the target. If **NULL**, canned patterns, as described in *VME Bridge Tests* on page 12-1, are used. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

numPatterns

represents the number of patterns in *patternPtr*. If **0**, *patternPtr* must be **NULL**.

rwMask

is a bit mask to be applied to (that is, ANDed with) each 32-bit pattern before it is copied to the internal write buffer. Only the low *writeWidth* bits of the mask are actually used. The name *rwMask* is really a misnomer; the mask is only applied to write data. Read data from the target is copied directly to the internal or user-supplied buffer. Likewise, *rwMask* is not applied to data written from a user-supplied buffer.

vmeMaxDw

must be set to **0**.

vmeAM

must be set to **0**.

vmeImage

must be set to **0**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied

BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_RW_TARGET_IO_FAULT—VME write/read target access fault
BIT_VME_RW_TARGET_DATA_FAULT—VME write/read target data miscompare
BIT_VME_RW_TARGET_INT_FAULT—VME write/read target interrupt fault
BIT_VME_NO_TARGET_RW_IO_FAULT—VME target write/read to non-existent address succeeded

VME Standard Target I/O Test

[**BIT_VME_STANDARD_IO**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_STANDARD_IO**] test.

This test exercises the target interface of the VME bridge with translation of PCI addresses to the VME Standard I/O (A24) address space.

This test has the following default test values:

Iteration:	1
Duration:	6000
Control:	HALT_ON_ERROR

Test Description

writeWidth and *readWidth* are ignored and default to **4** (32-bit), thus *writeBytes* and *readBytes* (if non-zero) must be multiples of **4**.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices, as described in *VME Bridge Tests* on page 12-1. The target VME host is affected.

Required Test Equipment

There must be at most one VME host in the system, besides the host under test, that is capable of responding at the user-supplied Standard I/O (A24) VME address space/range. There must be a functional VME arbiter in the system.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_RW_TARGET_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **BIT_VME_SPACE_STDIO**
vmeAddr = **(UINT8 *)BIT_PLAYPEN_A24_SLV_BUS + 0x10000**
locAddr = **(UINT8 *)BIT_VME_A24_MSTR_LOCAL + 0x10000**
bufferPtr = **NULL**
tgtValid = **TRUE**
readBytes = **0x8**
readWidth = **BIT_VME_RDWR_16**
writeBytes = **0xc**
writeWidth = **BIT_VME_RDWR_16**
patternPtr = **NULL**
numPatterns = **0**
rwMask = **0x7EF8**
vmeMaxDw = **0**
vmeAm = **0**
vmeImage = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_RW_TARGET** test. The user must use the definition of **VME_TARGET_TESTS_VALID_FLAG**.

vmeSpace

is a VME address space valid for the proposed target host.

vmeAddr

is a 24-bit VME address (0x0–0xFFFFFF) at which the target responds. It is mapped to *locAddr*. *vmeAddr* + (max (*readBytes*, *writeBytes*) – 1) must be valid for *vmeSpace* and may not wrap back to **0**.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested.

bufferPtr

is a local memory address accessible to the caller to be used as the source or sink for data to/from the target host. If **NULL**, the test allocates separate internal buffers for input/output. If *bufferPtr* is given, the test does not write canned patterns and does not verify that the read data matches what was written. *bufferPtr* + (max (*readBytes*, *writeBytes*) – 1) must be within the address space accessible to the caller. Note that *rwMask* is not applied to data written from a user-supplied buffer. It is assumed that the data in the buffer is ready for delivery to the target. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

tgtValid

defines the target address range. If **TRUE**, the target address range is expected to respond. An error is returned by the test if a target I/O error occurs (PCI or VME). If **FALSE**, the target address range is not expected to respond. An error is returned by the test if a target I/O error does not occur.

writeBytes

represents the number of bytes to write to the target (max 128KB). If **0**, a read-only test is done. The bytes written may be patterns (internal or external) or data from *bufferPtr*. If patterns are being written and read, a verify is done on the read data to ensure it matches what was written; *writeBytes* must be >= *readBytes* so that the verify phase can be safely performed.

readBytes

represents the number of bytes to read back (or simply read if *writeBytes* is **0**) from the target (max 128KB). If patterns are being used, *readBytes* must be <= *writeBytes* if *writeBytes* is > **0**. If patterns are being written, *readBytes* worth of pattern data is verified against what was written.

writeWidth

represents the number of bytes (1, 2 or 4) to write at one time to the target using programmed I/O (PIO). If patterns are being written, *writeWidth* bytes worth of the pattern are ANDed with *rwMask* and written to an internal write buffer. The order in which multiple (2 or 4) bytes from the pattern are written to the buffer is architecture-dependent. The canned patterns provided are constructed such that whatever byte order is used, all bits in the target location (subject to *rwMask*) are toggled.

readWidth

represents the number of bytes (1, 2 or 4) to read at one time from the target using programmed I/O (PIO). *writeWidth* and *readWidth* need not be the same; 4-byte PIO writes could be combined with 1-byte PIO reads.

patternPtr

is a pointer to an array of 32-bit values to be used as bit patterns to write/read to/from the target. If **NULL**, canned patterns, as described in *VME Bridge Tests* on page 12-1, are used. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

numPatterns

represents the number of patterns in *patternPtr*. If **0**, *patternPtr* must be **NULL**.

rwMask

is a bit mask to be applied to (that is, ANDed with) each 32-bit pattern before it is copied to the internal write buffer. Only the low *writeWidth* bits of the mask are actually used. The name *rwMask* is really a misnomer; the mask is only applied to write data. Read data from the target is copied directly to the internal or user-supplied buffer. Likewise, *rwMask* is not applied to data written from a user-supplied buffer.

vmeMaxDw

must be set to **0**.

vmeAM

must be set to **0**.

vmeImage

must be set to **0**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied

BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_RW_TARGET_IO_FAULT—VME write/read target access fault
BIT_VME_RW_TARGET_DATA_FAULT—VME write/read target data miscompare
BIT_VME_RW_TARGET_INT_FAULT—VME write/read target interrupt fault
BIT_VME_NO_TARGET_RW_IO_FAULT—VME target write/read to non-existent address succeeded

VME Extended Target I/O Test

[**BIT_VME_EXTENDED_IO**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_EXTENDED_IO**] test.

This test exercises the target interface of the VME bridge with translation of PCI addresses to the VME Extended I/O (A32) address space.

This test has the following default test values:

Iteration:	1
Duration:	6000
Control:	HALT_ON_ERROR

Test Description

writeWidth and *readWidth* are ignored and default to **4** (32-bit), thus *writeBytes* and *readBytes* (if non-zero) must be multiples of **4**.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices, as described in *VME Bridge Tests* on page 12-1. The target VME host is affected.

Required Test Equipment

There must be at most one VME host in the system, besides the host under test, that is capable of responding at the user-supplied Extended I/O (A32) VME address space/range. There must be a functional VME arbiter in the system.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_RW_TARGET_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **BIT_VME_SPACE_EXTIO**
vmeAddr = **(UINT8 *)(BIT_PLAYPEN_A32_SLV_BUS + 0x30000)**
locAddr = **(UINT8 *)((BIT_VME_A32_MSTR_LOCAL**
+ BIT_VME_A32_MSTR_SIZE)
– (BIT_PLAYPEN_SLV_SIZE)
+ 0x30000)
bufferPtr = **NULL**
tgtValid = **TRUE**
readBytes = **0x20000**
readWidth = **BIT_VME_RDWR_32**
writeBytes = **0x20000**
writeWidth = **BIT_VME_RDWR_32**
patternPtr = **NULL**
numPatterns = **0**
rwMask = **0xFFFFFFFF**
vmeMaxDw = **0**
vmeAm = **0**
vmeImage = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_RW_TARGET** test. The user must use the definition of **VME_TARGET_TESTS_VALID_FLAG**.

vmeSpace

is a VME address space valid for the proposed target host.

vmeAddr

is a 32-bit VME address at which the target responds. It is mapped to *locAddr*. *vmeAddr* + (max (*readBytes*, *writeBytes*) – 1) must be valid for *vmeSpace* and may not wrap back to **0**.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested.

bufferPtr

is a local memory address accessible to the caller to be used as the source or sink for data to/from the target host. If **NULL**, the test allocates separate internal buffers for input/output. If *bufferPtr* is given, the test does not write canned patterns and does not verify that the read data matches what was written. *bufferPtr* + (max (*readBytes*, *writeBytes*) – 1) must be within the address space accessible to the caller. Note that *rwMask* is not applied to data written from a user-supplied buffer. It is assumed that the data in the buffer is ready for delivery to the target. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

tgtValid

defines the target address range. If **TRUE**, the target address range is expected to respond. An error is returned by the test if a target I/O error occurs (PCI or VME). If **FALSE**, the target address range is not expected to respond. An error is returned by the test if a target I/O error does not occur.

writeBytes

represents the number of bytes to write to the target (max 128KB). If **0**, a read-only test is done. The bytes written may be patterns (internal or external) or data from *bufferPtr*. If patterns are being written and read, a verify is done on the read data to ensure it matches what was written; *writeBytes* must be >= *readBytes* so that the verify phase can be safely performed.

readBytes

represents the number of bytes to read back (or simply read if *writeBytes* = **0**) from the target (max 128KB). If patterns are being used, *readBytes* must be <= *writeBytes* if *writeBytes* is > **0**. If patterns are being written, *readBytes* worth of pattern data is verified against what was written.

writeWidth

represents the number of bytes (1, 2 or 4) to write at one time to the target using programmed I/O (PIO). If patterns are being written, *writeWidth* bytes worth of the pattern are ANDed with *rwMask* and written to an internal write buffer. The order in which multiple (2 or 4) bytes from the pattern are written to the buffer is architecture-dependent. The canned patterns provided are constructed such that whatever byte order is used, all bits in the target location (subject to *rwMask*) are toggled.

readWidth

represents the number of bytes (1, 2 or 4) to read at one time from the target using programmed I/O (PIO). *writeWidth* and *readWidth* need not be the same; 4-byte PIO writes could be combined with 1-byte PIO reads.

patternPtr

is a pointer to an array of 32-bit values to be used as bit patterns to write/read to/from the target. If **NULL**, canned patterns, as described in *VME Bridge Tests* on page 12-1, are used. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

numPatterns

represents the number of patterns in *patternPtr*. If **0**, *patternPtr* must be **NULL**.

rwMask

is a bit mask to be applied to (that is, ANDed with) each 32-bit pattern before it is copied to the internal write buffer. Only the low *writeWidth* bits of the mask are actually used. The name *rwMask* is really a misnomer; the mask is only applied to write data. Read data from the target is copied directly to the internal or user-supplied buffer. Likewise, *rwMask* is not applied to data written from a user-supplied buffer.

vmeMaxDw

must be set to **0**.

vmeAM

must be set to **0**.

vmeImage

must be set to **0**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied

BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_RW_TARGET_IO_FAULT—VME write/read target access fault
BIT_VME_RW_TARGET_DATA_FAULT—VME write/read target data miscompare
BIT_VME_RW_TARGET_INT_FAULT—VME write/read target interrupt fault
BIT_VME_NO_TARGET_RW_IO_FAULT—VME target write/read to non-existent address succeeded

VME CR/CSR Visibility Test

[**BIT_VME_CSR**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_CSR**] test.

This test exercises the target interface of the VME bridge with translation of PCI addresses to the VME CR/CSR (control register/control status register) address space.

This test has the following default test values:

Iteration:	1
Duration:	6000
Control:	HALT_ON_ERROR

Test Description

The test performs a read-only access of *readBytes* bytes, in 4-byte (32-bit) accesses. *writeWidth* and *readWidth* are ignored. *readBytes* must be a multiple of **4**.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices, as described in *VME Bridge Tests* on page 12-1. The target VME host is affected.

Required Test Equipment

There must be at most one VME host in the system, besides the host under test, that is capable of responding at the user-supplied CR/CSR VME address space/range. There must be a functional VME arbiter in the system.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_RW_TARGET_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **BIT_VME_SPACE_CRCSR**
vmeAddr = **(UINT8 *)(BIT_PLAYPEN_CRCSR_SLV_BUS + 0x7F000 + UNIVERSE_VSI7_BS)**
locAddr = **BIT_VME_ASSIGN_ADDR**
bufferPtr = **NULL**
tgtValid = **TRUE**
readBytes = **4**
readWidth = **BIT_VME_RDWR_32**
writeBytes = **4**
writeWidth = **BIT_VME_RDWR_32**
patternPtr = **NULL**
numPatterns = **0**
rwMask = **0xFFFF0000**
vmeMaxDw = **0**
vmeAm = **0**
vmeImage = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_RW_TARGET** test. The user must use the definition of **VME_TARGET_TESTS_VALID_FLAG**.

vmeSpace

is a VME address space valid for the proposed target host.

vmeAddr

is a VME address (valid for **BIT_VME_CRCSR [CR/CSR]**) at which the target responds. It is mapped to *locAddr*. *vmeAddr* + (*readBytes* – 1) must be valid for *vmeSpace* and may not wrap back to 0.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested.

bufferPtr

is a local memory address accessible to the caller to be used as the source or sink for data to/from the target host. If **NULL**, the test allocates separate internal buffers for input/output. If *bufferPtr* is given, the test does not write canned patterns and does not verify that the read data matches what was written. *bufferPtr* + (max (*readBytes*, *writeBytes*) – 1) must be within the address space accessible to the caller. Note that *rwMask* is not applied to data written from a user-supplied buffer. It is assumed that the data in the buffer is ready for delivery to the target. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

tgtValid

defines the target address range. If **TRUE**, the target address range is expected to respond. An error is returned by the test if a target I/O error occurs (PCI or VME). If **FALSE**, the target address range is not expected to respond. An error is returned by the test if a target I/O error does not occur.

writeBytes

represents the number of bytes to write to the target (max 128KB). If **0**, a read-only test is done. The bytes written may be patterns (internal or external) or data from *bufferPtr*. If patterns are being written and read, a verify is done on the read data to ensure it matches what was written; *writeBytes* must be >= *readBytes* so that the verify phase can be safely performed.

readBytes

represents the number of bytes to read back (or simply read if *writeBytes* = **0**) from the target (max 128KB). If patterns are being used, *readBytes* must be <= *writeBytes* if *writeBytes* is > **0**. If patterns are being written, *readBytes* worth of pattern data is verified against what was written.

writeWidth

represents the number of bytes (1, 2 or 4) to write at one time to the target using programmed I/O (PIO). If patterns are being written, *writeWidth* bytes worth of the pattern are ANDed with *rwMask* and written to an internal write buffer. The order in which multiple (2 or 4) bytes from the pattern are written to the buffer is architecture-dependent. The canned patterns provided are constructed such that whatever byte order is used, all bits in the target location (subject to *rwMask*) are toggled.

readWidth

represents the number of bytes (1, 2 or 4) to read at one time from the target using programmed I/O (PIO). *writeWidth* and *readWidth* need not be the same; 4-byte PIO writes could be combined with 1-byte PIO reads.

patternPtr

is a pointer to an array of 32-bit values to be used as bit patterns to write/read to/from the target. If **NULL**, canned patterns, as described in *VME Bridge Tests* on page 12-1, are used. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

numPatterns

represents the number of patterns in *patternPtr*. If **0**, *patternPtr* must be **NULL**.

rwMask

is a bit mask to be applied to (that is, ANDed with) each 32-bit pattern before it is copied to the internal write buffer. Only the low *writeWidth* bits of the mask are actually used. The name *rwMask* is really a misnomer; the mask is only applied to write data. Read data from the target is copied directly to the internal or user-supplied buffer. Likewise, *rwMask* is not applied to data written from a user-supplied buffer.

vmeMaxDw

must be set to **0**.

vmeAM

must be set to **0**.

vmeImage

must be set to **0**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied

BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_RW_TARGET_IO_FAULT—VME write/read target access fault
BIT_VME_RW_TARGET_DATA_FAULT—VME write/read target data miscompare
BIT_VME_RW_TARGET_INT_FAULT—VME write/read target interrupt fault
BIT_VME_NO_TARGET_RW_IO_FAULT—VME target write/read to non-existent address succeeded

VME DMA (Extended I/O) Target Test

[**BIT_VME_DMA**]
tests/vme/vmeTests.h

The **tests/vme/vmeTests.h** file defines the parameter structures and fields mentioned in the [**BIT_VME_DMA**] test.

This test exercises the DMA interface of the VME bridge with mastering of PCI to VME transfers in the VME Extended I/O (A32) address range.

This test has the following default test values:

Iteration:	1
Duration:	6000
Control:	HALT_ON_ERROR

Test Description

The DMA engine is configured to use 64-bit block transfers on VME and 64-bit transfers on PCI, if possible, and to interrupt upon transfer completion or failure. A single DMA direct-mode command is given to the engine for each direction, requesting *writeBytes* or *readBytes* to be transferred. If the interrupt does not occur within 500 ms, or if error status is returned in the DMA status registers, an error is returned by the test.

writeWidth and *readWidth* are ignored and default to 8-bytes (64-bit), thus *writeBytes* and *readBytes* (if non-zero) must be multiples of 8. Also, if *bufferPtr* is provided, it must be aligned with *vmeAddr* in the first 8-bytes.

Affected Peripheral Devices

This test affects the **BIT_PCI_TO_VME_BRIDGE** and the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** devices, as described in *VME Bridge Tests* on page 12-1. The target VME host is affected.

Required Test Equipment

There must be at most one VME host in the system, besides the host under test, that is capable of responding at the user-supplied Extended I/O (A32) VME address space/range. There must be a functional VME arbiter in the system.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/vme/vmeTests.h** (**VME_RW_TARGET_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **VME_REGISTER_TESTS_VALID_FLAG**
vmeSpace = **BIT_VME_SPACE_EXTIO**
vmeAddr = **(UINT8 *)(BIT_PLAYPEN_A32_SLV_BUS + 0x30000)**
locAddr = **BIT_VME_INVALID_ADDR**
bufferPtr = **NULL**
tgtValid = **TRUE**
readBytes = **0x20000**
readWidth = **BIT_VME_RDWR_DMA**
writeBytes = **0x20000**
writeWidth = **BIT_VME_RDWR_DMA**
patternPtr = **NULL**
numPatterns = **0**
rwMask = **0xFFFFFFFF**

vmeMaxDw = **0**
vmeAm = **0**
vmeImage = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided as a signature for the data structure passed to the **BIT_VME_RW_TARGET** test. The user must use the definition of **VME_TARGET_TESTS_VALID_FLAG**.

vmeSpace

is a VME address space valid for the proposed target host.

vmeAddr

is a 32-bit VME address at which the target responds. *vmeAddr* + (max (*readBytes*, *writeBytes*) – 1) must be valid for *vmeSpace* and may not wrap back to **0**. Although any 1-byte alignment of this address is legal, *vmeAddr* and *bufferPtr* (if provided) must have the same byte-alignment in the first 8-bytes.

locAddr

is either a local address that is currently mapped to the PCI memory space allocated by the BSP for translation to VME space or **BIT_VME_ASSIGN_ADDR**. If **BIT_VME_ASSIGN_ADDR**, the test picks an address from the range normally assigned by the BSP to map PCI to VME A32 (typically 0x10000000-0x18000000) and re-uses this for whatever *vmeSpace* is requested.

bufferPtr

is a local memory address accessible to the caller to be used as the source or sink for data to/from the target host. If **NULL**, the test allocates separate internal buffers for input/output. If *bufferPtr* is given, the test does not write canned patterns and does not verify that the read data matches what was written. *bufferPtr* + (max (*readBytes*, *writeBytes*) – 1) must be within the address space accessible to the caller. Note that *rwMask* is not applied to data written from a user-supplied buffer. It is assumed that the data in the buffer is ready for delivery to the target. This address must be aligned to *vmeAddr* in the first 8-bytes. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

tgtValid

defines the target address range. If **TRUE**, the target address range is expected to respond. An error is returned by the test if a target I/O error occurs (PCI or VME). If **FALSE**, the target address range is not expected to respond. An error is returned by the test if a target I/O error does not occur.

writeBytes

represents the number of bytes to write to the target (max 128KB). If **0**, a read-only test is done. The bytes written may be patterns (internal or external) or data from *bufferPtr*. If patterns are being written and read, a verify is done on the read data to ensure it matches what was written; *writeBytes* must be >= *readBytes* so that the verify phase can be safely performed.

readBytes

represents the number of bytes to read back (or simply read if *writeBytes* = **0**) from the target (max 128KB). If patterns are being used, *readBytes* must be <= *writeBytes* if *writeBytes* is > **0**. If patterns are being written, *readBytes* worth of pattern data is verified against what was written.

writeWidth

represents the number of bytes (1, 2 or 4) to write at one time to the target using programmed I/O (PIO). If patterns are being written, *writeWidth* bytes worth of the pattern are ANDed with *rwMask* and written to an internal write buffer. The order in which multiple (2 or 4) bytes from the pattern are written to the buffer is architecture-dependent. The canned patterns provided are constructed such that whatever byte order is used, all bits in the target location (subject to *rwMask*) are toggled.

readWidth

represents the number of bytes (1, 2 or 4) to read at one time from the target using programmed I/O (PIO). *writeWidth* and *readWidth* need not be the same; 4-byte PIO writes could be combined with 1-byte PIO reads.

patternPtr

is a pointer to an array of 32-bit values to be used as bit patterns to write/read to/from the target. If **NULL**, canned patterns, as described in *VME Bridge Tests* on page 12-1, are used. Only one of *patternPtr* or *bufferPtr* may be provided; the other must be **NULL**.

numPatterns

represents the number of patterns in *patternPtr*. If **0**, *patternPtr* must be **NULL**.

rwMask

is a bit mask to be applied to (that is, ANDed with) each 32-bit pattern before it is copied to the internal write buffer. Only the low *writeWidth* bits of the mask are actually used. The name *rwMask* is really a misnomer; the mask is only applied to write data. Read data from the target is copied directly to the internal or user-supplied buffer. Likewise, *rwMask* is not applied to data written from a user-supplied buffer.

vmeMaxDw

must be set to **0**.

vmeAM

must be set to **0**.

vmeImage

must be set to **0**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_DEVICE_NOT_PRESENT—device is not present
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_VME_RW_TARGET_IO_FAULT—VME write/read target access fault
BIT_VME_RW_TARGET_DATA_FAULT—VME write/read target data miscompare
BIT_VME_RW_TARGET_INT_FAULT—VME write/read target interrupt fault
BIT_VME_NO_TARGET_RW_IO_FAULT—VME target write/read to non-existent address succeeded

Common VME Bridge Test Return Values

BIT_NO_FAULT_DETECTED

The subtest succeeded with the parameter values provided.

BIT_DEVICE_NOT_SUPPORTED

The device parameter of *TEST_ENTRY* was not **PCI_TO_VME_BRIDGE**.

BIT_SUBTEST_NOT_SUPPORTED

The subtest parameter of *TEST_ENTRY* was < **VME_REGISTER** or > **VME_LOCMON**.

An internal test error occurred.

BIT_INVALID_TEST_PARAM

A **NULL** *testParamPtr* was passed in a *TEST_ENTRY*.

An internal test error occurred.

An error occurred when probing user-supplied local addresses for validity.

BIT_DEVICE_NOT_PRESENT

An MBIT handle could not be acquired for the **BIT_PCI_TO_VME_BRIDGE** device.

BIT_INSTALL_DEV_FAILED

An MBIT handle could not be acquired for the **BIT_LOCAL_BUS_TO_PCI_BRIDGE** device.

The VME DMA controller was found to be active upon VME subtest installation.

BIT_INIT_ALLOCATION_ERROR

A semaphore could not be created.

An input or output buffer for target I/O could not be allocated.

BIT_SYS_RESTORE_FAILED

Unable to restore register contents for **BIT_PCI_TO_VME_BRIDGE** or **BIT_LOCAL_BUS_TO_PCI_BRIDGE**.

BIT_BUS_ERROR

An unexpected non-target or non-probe-related I/O error occurred during test installation, execution or de-installation.

Ethernet Tests 13

This chapter provides descriptions and requirements for the following Ethernet tests:

Ethernet Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_ETHERNET_ROM_ACCESS]	tests/ethernet/ gd82559erTests.h	BIT_ETHERNET_ROM_ PARAMS
[BIT_ETHERNET_ROM_VERIFY]	tests/ethernet/ gd82559erTests.h	BIT_ETHERNET_ROM_ PARAMS
[BIT_ETHERNET_REGISTER_ ACCESS]	tests/ethernet/ gd82559erTests.h	BIT_ETHERNET_ REGISTER_PARAMS
[BIT_ETHERNET_REGISTER]	tests/ethernet/ gd82559erTests.h	BIT_ETHERNET_ REGISTER_PARAMS
[BIT_ETHERNET_INTERNAL_ LOOPBACK]	tests/ethernet/ gd82559erTests.h	BIT_ETHERNET_ LOOPBACK_PARAMS
[BIT_ETHERNET_EXTERNAL_ LOOPBACK]	tests/ethernet/ gd82559erTests.h	BIT_ETHERNET_ LOOPBACK_PARAMS

Serial EEPROM Device Accessibility Test

[**BIT_ETHERNET_ROM_ACCESS**]
tests/ethernet/gd82559erTests.h

The **tests/ethernet/gd82559erTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ETHERNET_ROM_ACCESS**] test.

This test reads from the serial EEPROM connected to the Ethernet controller to determine if the serial EEPROM is accessible. On the MVME51*xx*, the serial EEPROM size is 128 bytes.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test resets the Ethernet controller, configures it for use, reads the serial EEPROM, calculates a checksum on the portion read and resets the Ethernet controller to leave it in a quiescent state. The test accepts parameters for a buffer and portion of the serial EEPROM to read.

Affected Peripheral Devices

This test affects the Ethernet controller and the serial EEPROM connected to the Ethernet controller.

Caution

This test must not run after the VxWorks Ethernet driver for this device starts. This test destroys the state of the Ethernet controller device, but it does not affect the contents of the serial EEPROM.

Required Test Equipment

This test does not require test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/ethernet/gd82559erTests.h** (**BIT_ETHERNET_ROM_PARAMS**).

This test's default parameters are as follows:
signature = **BIT_ETHERNET_ROM_PARAMS_SIGNATURE**
buffer = **NULL**
length = **0**
offset = **0**

The contents of each structure and their effects on the test is discussed below:

signature

must contain **BIT_ETHERNET_ROM_PARAMS_SIGNATURE**.

buffer

is **NULL** or a pointer to a buffer at least *length* bytes long to receive the serial EEPROM contents.

length

is **0** or the number of bytes of serial EEPROM contents to place in a buffer. It must be a multiple of **2** and less than or equal to the serial EEPROM size in bytes.

offset

is the number of bytes from the start of the serial EEPROM to skip when reading. It must be a multiple of **2** and less than the serial EEPROM size in bytes.

checksum

is the checksum calculated on the portion of the serial EEPROM read here.

The sum of *length* and *offset* must be less than or equal to the serial EEPROM size in bytes. If *buffer* is **NULL**, *length* is **0** and *offset* is **0**, the test reads the entire contents of the serial EEPROM and returns no content data.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_ETHERNET_SROM_CHECKSUM_FAULT—the checksum of the Serial EEPROM (SROM) does not match the calculated value
BIT_ETHERNET_SROM_ACCESS_ERROR—an error occurred while accessing the Serial EEPROM (SROM) connected to the Ethernet device
BIT_ETHERNET_DRIVER_FAULT—a driver fault occurred
BIT_ETHERNET_TRANSMIT_ERROR—an error occurred during transmission
BIT_ETHERNET_TRANSMIT_BLOCK—the transmission would block

Serial EEPROM Device Verify Test

[**BIT_ETHERNET_ROM_VERIFY**]
tests/ethernet/gd82559erTests.h

The **tests/ethernet/gd82559erTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ETHERNET_ROM_VERIFY**] test.

This test reads the entire contents of the serial EEPROM connected to the Ethernet controller, calculates the checksum of the contents and compares this with the expected value to determine if the serial EEPROM operates properly and contains valid data.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test resets the Ethernet controller, configures it for use, reads the serial EEPROM, calculates a checksum on the portion read, verifies the checksum and resets the Ethernet controller to leave it in a quiescent state. The test accepts parameters for a buffer and a portion of the serial EEPROM to read. If only a portion of the serial EEPROM is specified, then the test most likely returns a **BIT_ETHERNET_SROM_CHECKSUM_FAULT**.

Affected Peripheral Devices

This test affects the Ethernet controller and the serial EEPROM connected to the Ethernet controller.

Caution

This test must not run after the VxWorks Ethernet driver for this device starts. This test destroys the state of the Ethernet controller device, but it does not affect the contents of the serial EEPROM.

Required Test Equipment

This test does not require test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/ethernet/gd82559erTests.h** (**BIT_ETHERNET_ROM_PARAMS**).

This test’s default parameters are as follows:
signature = **BIT_ETHERNET_ROM_PARAMS_SIGNATURE**
buffer = **NULL**
length = **0**
offset = **0**

The contents of each structure and their effects on the test is discussed below:

signature

must contain **BIT_ETHERNET_ROM_PARAMS_SIGNATURE**.

buffer

is **NULL** or a pointer to a buffer at least *length* bytes long to receive the serial EEPROM contents.

length

is **0** or the number of bytes of serial EEPROM contents to place in buffer. It must be a multiple of **2** and less than or equal to the serial EEPROM size in bytes.

offset

is the number of bytes from the start of the serial EEPROM to skip when reading. It must be a multiple of **2** and less than the serial EEPROM size in bytes.

checksum

is the checksum calculated on the portion of the serial EEPROM.

The sum of *length* and *offset* must be less than or equal to the serial EEPROM size in bytes. If *offset* is **NULL**, *length* is **0** and *offset* is **0**, the test reads the entire contents of the serial EEPROM and returns no content data.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_ETHERNET_SROM_CHECKSUM_FAULT—the checksum of the Serial EEPROM (SROM) does not match the calculated value
BIT_ETHERNET_SROM_VERIFY_FAULT—the contents of the Serial EEPROM (SROM) do not make sense or do not match the expected values
BIT_ETHERNET_DRIVER_FAULT—a driver fault occurred
BIT_ETHERNET_TRANSMIT_ERROR—an error occurred during transmission
BIT_ETHERNET_TRANSMIT_BLOCK—the transmission would block

Register Accessibility Test

[**BIT_ETHERNET_REGISTER_ACCESS**]
tests/ethernet/gd82559erTests.h

The **tests/ethernet/gd82559erTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ETHERNET_REGISTER_ACCESS**] test.

This test reads all of the Ethernet controller registers to determine if the Ethernet controller is accessible.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test resets the Ethernet controller, configures it for use, reads the Ethernet controller's registers and resets the Ethernet controller to leave it in a quiescent state.

Affected Peripheral Devices

This test affects the Ethernet controller and the serial EEPROM connected to the Ethernet controller.

Caution

This test must not run after the VxWorks Ethernet driver for this device starts. This test destroys the state of the Ethernet controller device, but it does not affect the contents of the serial EEPROM.

Required Test Equipment

This test does not require test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/ethernet/gd82559erTests.h** (**BIT_ETHERNET_REGISTER_PARAMS**).

This test's default parameters are as follows:
signature = **BIT_ETHERNET_REGISTER_PARAMS_SIGNATURE**
buffer = **NULL**
length = **0**

The contents of each structure and their effects on the test is discussed below:

signature

must contain **BIT_ETHERNET_REGISTER_PARAMS_SIGNATURE**.

buffer

is **NULL** or a pointer to a buffer at least *length* bytes long to receive the Ethernet controller register contents.

length

is **0** or the number of bytes of Ethernet controller register contents to place in *buffer*. If *length* is not **0**, it must be at least **64**.

If *buffer* is **NULL** and *length* is **0**, the test reads all of the Ethernet controller registers and returns no register data.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_ETHERNET_REGISTER_ACCESS_FAULT—an error occurred while accessing the Ethernet device registers
BIT_ETHERNET_DRIVER_FAULT—a driver fault occurred
BIT_ETHERNET_TRANSMIT_ERROR—an error occurred during transmission
BIT_ETHERNET_TRANSMIT_BLOCK—the transmission would block

Register Test

[**BIT_ETHERNET_REGISTER**]
tests/ethernet/gd82559erTests.h

The **tests/ethernet/gd82559erTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ETHERNET_REGISTER**] test.

This test reads and writes the Ethernet controller registers and tracks which bits change to determine if the registers contain stuck bits.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test resets the Ethernet controller device, initializes it for use, runs portions of the other tests and additional read and write tests to change modifiable bits, and resets the device and leaves it in a quiescent state. The parameters allow reading the register contents into a buffer.

Affected Peripheral Devices

This test affects the Ethernet controller and the serial EEPROM connected to the Ethernet controller.

Caution

This test must not run after the VxWorks Ethernet driver for this device starts. This test destroys the state of the Ethernet controller device, but it does not affect the contents of the serial EEPROM.

Required Test Equipment

This test does not require test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/ethernet/gd82559erTests.h** (**BIT_ETHERNET_REGISTER_PARAMS**).

This test's default parameters are as follows:
signature = **BIT_ETHERNET_REGISTER_PARAMS_SIGNATURE**
buffer = **NULL**
length = **0**

The contents of each structure and their effects on the test is discussed below:

signature

> must contain **BIT_ETHERNET_REGISTER_PARAMS_SIGNATURE**.

buffer

> is **NULL** or a pointer to a buffer at least *length* bytes long to receive the Ethernet controller register contents.

length

> is **0** or the number of bytes of Ethernet controller register contents to place in *buffer*. If *length* is not **0**, then it must be at least **64**.

If *buffer* is **NULL** and *length* is **0**, the test tests all the Ethernet controller registers, reads all of the Ethernet controller registers and returns no register data.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_ETHERNET_REGISTER_ACCESS_FAULT—an error occurred while accessing the Ethernet device registers
BIT_ETHERNET_DRIVER_FAULT—a driver fault occurred
BIT_ETHERNET_TRANSMIT_ERROR—an error occurred during transmission
BIT_ETHERNET_TRANSMIT_BLOCK—the transmission would block

Internal Loopback Test

[BIT_ETHERNET_INTERNAL_LOOPBACK]
tests/ethernet/gd82559erTests.h

The **tests/ethernet/gd82559erTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ETHERNET_INTERNAL_LOOPBACK**] test.

This test transmits an Ethernet frame containing a data payload in internal loopback mode and compares the sent and received data.

This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

Test Description

This test initializes the Ethernet controller and sets internal loopback mode, transmits an Ethernet frame, compares the sent and received frames, resets the device and leaves it in a quiescent state to verify the Ethernet controller operates properly in internal loopback mode.

Affected Peripheral Devices

This test affects the Ethernet controller and the serial EEPROM connected to the Ethernet controller.

Caution

This test must not run after the VxWorks Ethernet driver for this device starts. This test destroys the state of the Ethernet controller device, but it does not affect the contents of the serial EEPROM.

Required Test Equipment

This test does not require test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/ethernet/gd82559erTests.h** (**BIT_ETHERNET_LOOPBACK_PARAMS**).

This test's default parameters are as follows:
signature =**BIT_ETHERNET_LOOPBACK_PARAMS_SIGNATURE**
buffer = **NULL**
length = **0**

The contents of each structure and their effects on the test is discussed below:

signature

must contain **BIT_ETHERNET_LOOPBACK_PARAMS_SIGNATURE**.

buffer

is **NULL** or a pointer to a buffer at least *length* bytes long to transmit.

length

is **0** or the number of bytes to transmit from buffer. It must be at least **GD82559ER_MINIMUM_ETHERNET_PACKET_SIZE** and no more than **GD82559ER_MAXIMUM_PACKET_SIZE**.

If *buffer* is **NULL** and *length* is not **0** and valid, the test transmits *length* bytes with a repeating pattern of: 0x00000000, 0xFFFFFFFF, 0xFFFF0000, 0x0000FFFF, 0xFF00FF00, 0x00FF00FF, 0xF0F0F0F0, 0x0F0F0F0F, 0xCCCCCCCC, 0x33333333, 0xAAAAAAAA, 0x55555555, 0x00000000.

If *buffer* is **NULL** and *length* is **0**, the test transmits 1500 bytes with a repeating pattern of: 0x00000000, 0xFFFFFFFF, 0xFFFF0000, 0x0000FFFF, 0xFF00FF00, 0x00FF00FF, 0xF0F0F0F0, 0x0F0F0F0F, 0xCCCCCCCC, 0x33333333, 0xAAAAAAAA, 0x55555555, 0x00000000.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_ETHERNET_LOOPBACK_DATA_FAULT—the transmitted and received data sent in loopback do not match
BIT_ETHERNET_DRIVER_FAULT—a driver fault occurred
BIT_ETHERNET_TRANSMIT_ERROR—an error occurred during transmission
BIT_ETHERNET_TRANSMIT_BLOCK—the transmission would block

External Loopback Test

[**BIT_ETHERNET_EXTERNAL_LOOPBACK**]
tests/ethernet/gd82559erTests.h

The **tests/ethernet/gd82559erTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ETHERNET_EXTERNAL_LOOPBACK**] test.

This test transmits an Ethernet frame containing a data payload in external loopback mode and compares the sent and received data.

This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

Test Description

This test initializes the Ethernet controller and sets external loopback mode, transmits an Ethernet frame, compares the sent and received frames, resets the device and leaves it in a quiescent state to verify the Ethernet controller operates properly in external loopback mode.

Affected Peripheral Devices

This test affects the Ethernet controller and the serial EEPROM connected to the Ethernet controller.

Caution

This test must not run after the VxWorks Ethernet driver for this device starts. This test destroys the state of the Ethernet controller device, but it does not affect the contents of the serial EEPROM.

Required Test Equipment

This test requires an external loopback cable inserted in the Ethernet controller's connector.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/ethernet/gd82559erTests.h** (**BIT_ETHERNET_LOOPBACK_PARAMS**).

This test's default parameters are as follows:
signature =**BIT_ETHERNET_LOOPBACK_PARAMS_SIGNATURE**
buffer = **NULL**
length = **0**

The contents of each structure and their effects on the test is discussed below:

signature

must contain **BIT_ETHERNET_LOOPBACK_PARAMS_SIGNATURE**.

buffer

is **NULL** or a pointer to a buffer at least *length* bytes long to transmit.

length

is **0** or the number of bytes to transmit from *buffer*. It must be at least **GD82559ER_MINIMUM_ETHERNET_PACKET_SIZE** and no more than **GD82559ER_MAXIMUM_PACKET_SIZE**.

If *buffer* is **NULL** and *length* is not **0** and valid, the test transmits *length* bytes with a repeating pattern of: 0x00000000, 0xFFFFFFFF, 0xFFFF0000, 0x0000FFFF, 0xFF00FF00, 0x00FF00FF, 0xF0F0F0F0, 0x0F0F0F0F, 0xCCCCCCCC, 0x33333333, 0xAAAAAAAA, 0x55555555, 0x00000000.

If *buffer* is **NULL** and *length* is **0**, the test transmits 1500 bytes with a repeating pattern of: 0x00000000, 0xFFFFFFFF, 0xFFFF0000, 0x0000FFFF, 0xFF00FF00, 0x00FF00FF, 0xF0F0F0F0, 0x0F0F0F0F, 0xCCCCCCCC, 0x33333333, 0xAAAAAAAA, 0x55555555, 0x00000000.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_ETHERNET_LOOPBACK_DATA_FAULT—the transmitted and received data sent in loopback do not match
BIT_ETHERNET_DRIVER_FAULT—a driver fault occurred
BIT_ETHERNET_TRANSMIT_ERROR—an error occurred during transmission
BIT_ETHERNET_TRANSMIT_BLOCK—the transmission would block

System I/O Controller Test 14

This chapter provides a description and requirements for the system I/O controller test.

System I/O Controller Test

The table below highlights the test's string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_ISA_DEVICE_VISIBILITY]	tests/visibilityTests.h	VISIBILITY_PARAMS

ISA Host Bridge Device Visibility Test

[**BIT_ISA_DEVICE_VISIBILITY**]
tests/visibilityTests.h

The **tests/visibilityTests.h** file defines the parameter structures and fields mentioned in the [**BIT_ISA_DEVICE_VISIBILITY**] test.

This test probes devices in the list to determine if the device is visible. The ISA host bridge, PCI host bridge (PHB) and system memory controller (SMC) visibility tests use a common test routine and test mechanism.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

This test will use its default parameters by passing a **NULL** test parameter structure to the test routine. This test will ensure the visibility of devices on both sides of the **BIT_PCI_TO_ISA_BRIDGE**. A list of devices that will be tested are as follows:

BIT_PCI_TO_ISA_BRIDGE

BIT_ASYNC_SERIAL_DEVICE3

BIT_ASYNC_SERIAL_DEVICE4

BIT_PARALLEL_DEVICE1

A location is designated for visibility testing on each device. For locations designated as read-only or write-only, the routine does not verify that the read or write succeeded. The only situations that cause a read-only or write-only visibility test to fail is when an exception occurs due to the read or write. For locations designated as read and write, the routine first saves the contents of the location, then writes a test pattern that differs from the location's original contents, reads the test pattern back, compares the patterns, and finally restores the location's original contents. If the pattern comparison fails, a failure status is returned.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/visibilityTests.h** (**VISIBILITY_PARAMS**).

This test's default parameters are as follows:
deviceList = **NULL**
numDevices = **0**

The contents of the structure and their effects on the test are discussed below:

deviceList

is set to **NULL** by default and is ignored otherwise.

numDevices

is set to **0** by default and is ignored otherwise.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_INSTALL_DEV_FAILED—device installation failed
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_VISIBILITY_FAULT—device failed visibility test
BIT_NO_VISIBILITY_LOCATION—no location on device designated for visibility testing

Parallel Device Tests 15

This chapter provides descriptions and requirements for the following parallel device tests:

Parallel Device Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_PARALLEL_ REGISTER]	tests/parallel/parallelTests.h	PARALLEL_TEST_PARAMS
[BIT_PARALLEL_FIFO]	tests/parallel/parallelTests.h	PARALLEL_TEST_PARAMS

Both parallel device tests have the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Parallel Port Register Test

[**BIT_PARALLEL_REGISTER**]
tests/parallel/parallelTests.h

The **tests/parallel/parallelTests.h** file defines the parameter structures and fields mentioned in the [**BIT_PARALLEL_REGISTER**] test.

This test verifies that the chip's registers can be written and read.

Test Description

This test sets the parallel port's registers to predefined values and reads the register back to verify that the correct bits are set or cleared. All registers with reserved bits of either 0 or 1 are tested to ensure that they are in the proper state. All the remaining bits in each register are tested to ensure that they can be set to all ones, all zeroes, and an alternating bit pattern to ensure that there are no stuck or shorted bits adjacent to each other.

In some cases, there are register bits that cannot be changed because they may cause the system to go into an indeterminate state. These bits are not altered to protect the system integrity.

Affected Peripheral Devices

Any peripherals connected to the parallel port should be disconnected before running this test.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/parallel/parallelTests.h** (**PARALLEL_TEST_PARAMS**).

This test's default parameter is as follows:
validParamsFlag = **PARALLEL_TESTS_VALID_FLAG**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_PARALLEL_REGISTER** test. The user must use the definition of **PARALLEL_TESTS_VALID_FLAG**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_PARALLEL_REGISTER_FAULT—parallel register test fault

Parallel Port FIFO Test

[**BIT_PARALLEL_FIFO**]
tests/parallel/parallelTests.h

The **tests/parallel/parallelTests.h** file defines the parameter structures and fields mentioned in the [**BIT_PARALLEL_FIFO**] test.

This test verifies that the chip's FIFO can be written to and the written data can be read.

Test Description

This test configures the parallel port to enable the internal FIFO. Once the FIFO is enabled, the test writes a predetermined number of bytes and then reads the same number back into memory. The acquired data is compared with the data that was written. If an item or items do not match, an error is reported.

Affected Peripheral Devices

Any peripherals connected to the parallel port should be disconnected before running this test.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/parallel/parallelTests.h** (**PARALLEL_TEST_PARAMS**).

This test's default parameter is as follows:
validParamsFlag = **PARALLEL_TESTS_VALID_FLAG**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_PARALLEL_FIFO** test. The user must use the definition of **PARALLEL_TESTS_VALID_FLAG**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INIT_NOT_PERFORMED—MBIT initialization was not performed
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_PARALLEL_FIFO_FAULT—parallel FIFO test fault

SCSI Device Tests 16

This chapter provides descriptions and requirements for the following SCSI device tests:

SCSI Device Tests

The table below highlights each tests' string, header file, and parameter structure for quick access.

Test String	Header File	Parameter Structure
[BIT_SCSI_RAM]	tests/scsi/sym895ATests.h	SYM895A_LOCAL_DATA_ PARAMS
BIT_[SCSI_LOCAL_DATA]	tests/scsi/sym895ATests.h	SYM895A_LOCAL_DATA_ PARAMS
[BIT_SCSI_ID]	tests/scsi/sym895ATests.h	SYM895A_LOOPBACK_ PARAMS
[BIT_SCSI_INTERNAL_ LOOPBACK]	tests/scsi/sym895ATests.h	SYM895A_LOOPBACK_ PARAMS
[BIT_SCSI_PARITY]	tests/scsi/sym895ATests.h	SYM895A_LOOPBACK_ PARAMS
[BIT_SCSI_ INSTRUCTIONS]	tests/scsi/sym895ATests.h	SYM895A_LOOPBACK_ PARAMS

SCSI SCRIPTS RAM Test

[**BIT_SCSI_RAM**]
tests/scsi/sym895ATests.h

The **tests/scsi/sym895ATests.h** file defines the parameter structures and fields mentioned in the [**BIT_SCSI_RAM**] test.

This test verifies the functional condition of the PCI bus interface and SCSI SCRIPTS RAM on the SYM53C895A SCSI controller.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

Caution

This test is destructive to the state of the SYM53C895A SCSI controller and must be run before the driver is started.

This test sends various data patterns across the PCI bus from main memory to the SYM53C895A SCRIPTS memory. The test provides two methods of testing the PCI bus interface. The user may provide a test data buffer for the test or otherwise the test uses a predefined set of patterns: 0xFFFFFFFF, 0xFFFF0000, 0xFF00FF00, 0xF0F0F0F0, 0xCCCCCCCC, 0xAAAAAAAA, 0x0000FFFF, 0x00FF00FF, 0x0F0F0F0F, 0x33333333, 0x55555555, and 0x00000000. When the predefined patterns are used, the test creates a buffer of a single pattern to fill the SCRIPTS RAM. The buffer is then transferred to SCRIPTS RAM and verified. The test repeats this process until all patterns have been transferred and verified. The predefined test patterns detect stuck-at and bridging faults on the PCI bus interface or in the SCSI SCRIPTS RAM.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/scsi/sym895ATests.h** (**SYM895A_LOCAL_DATA_PARAMS**).

This test's default parameters are as follows:
validParamsFlag =
SYM895A_LOCAL_DATA_TESTS_VALID_FLAG
bufferOffset = **0**
dataBufferPtr = **NULL**
numBytes = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SCSI_RAM** test. The user must use the definition of **SYM895A_LOCAL_DATA_TESTS_VALID_FLAG**.

bufferOffset

is provided to allow the test to operate on a subset of the SCRIPTS RAM. The starting address is the offset in bytes from the base of SCRIPTS RAM. If the value of this parameter is **0**, the test begins at the base address of SCRIPTS RAM. If the value is not **0**, the memory from *bufferOffset* to *numBytes* is tested. If a *dataBufferPtr* is not provided (**NULL**), the offset should be set to **0**. The sum of *bufferOffset* and *numBytes* should not exceed the size of SCRIPTS RAM (8KB, 2048 x 32 bits).

dataBufferPtr

allows the user to provide data patterns by setting this parameter to the address of the test data buffer. To indicate that a *dataBufferPtr* has not been provided, set this parameter to **NULL**. If **NULL**, the predefined test patterns are used by this test.

numBytes

specifies the number of bytes contained in the test data buffer pointed to by *dataBufferPtr*. The parameter indicates the number of bytes in SCRIPTS RAM that are tested beginning at *bufferOffset*. The sum of *bufferOffset* and *numBytes* should not exceed the size of the SCRIPTS RAM. If a *dataBufferPtr* is not provided (**NULL**), this parameter should be set to **0**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SCSI_INIT_ERROR—SCSI bus not free for initialization
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SCSI_FIFO_CLEAR_FAULT—SCSI and/or DMA FIFOs failed to clear
BIT_SYS_RESTORE_FAILED—unable to restore pre-test system state
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence

SCSI Bridging Fault Test

[BIT_SCSI_LOCAL_DATA]
tests/scsi/sym895ATests.h

The **tests/scsi/sym895ATests.h** file defines the parameter structures and fields mentioned in the [**BIT_SCSI_LOCAL_DATA**] test.

This test verifies the functional condition of the PCI bus interface using the Load and Store SCSI scripts instructions. These instructions are specific to the SYM53C895A SCSI controller.

This test has the following default test values:

Iteration:	1
Duration:	2000
Control:	HALT_ON_ERROR

Test Description

Caution

This test is destructive to the state of the SYM53C895A SCSI controller and must be run before the driver is started.

This test uses the Load and Store SCSI scripts instructions to read, then write a series of test patterns from and to main memory. The test begins by passing the address that contains the Load instruction script, causing the Scratch A register to be loaded. The Store instruction script is then executed, causing the contents of the Scratch A register to be stored in a known main memory location. The test then reads the stored test patterns from the destination memory locations and verifies the contents.

The test provides two methods for testing the Load and Store instruction scripts. If a data buffer is not provided, the test uses a predefined set of patterns: 0xFFFFFFFF, 0xFFFF0000, 0xFF00FF00, 0xF0F0F0F0, 0xCCCCCCCC, 0xAAAAAAAA, 0x0000FFFF, 0x00FF00FF, 0x0F0F0F0F, 0x33333333, 0x55555555, and 0x00000000. Alternatively, if a data buffer is provided, the test uses the data buffer instead of the predefined set of patterns. The size of the user-defined data buffer must be a multiple of four since the 32-bit Scratch A register is filled for each load and store performed on the data. The predefined test patterns detect stuck-at and bridging faults on the PCI bus interface or in the Scratch A register.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/scsi/sym895ATests.h** (**SYM895A_LOCAL_DATA_PARAMS**).

This test's default parameters are as follows:
validParamsFlag =
SYM895A_LOCAL_DATA_TESTS_VALID_FLAG
dataBufferPtr = **NULL**
numBytes = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SCSI_LOCAL_DATA** test. The user must use the definition of **SYM895A_LOCAL_DATA_TESTS_VALID_FLAG**.

dataBufferPtr

allows the user to provide an array of 32-bit data patterns by setting the parameter to the address of the test data buffer. To indicate that a data buffer has not been provided, set this parameter to **NULL**. If **NULL**, the predefined test patterns are used by this test.

numBytes

specifies the number of bytes contained in the test data buffer, pointed to by *dataBufferPtr*. The *length* must be a valid multiple of four since the instruction script loads and stores four bytes at a time, allowing each load to fill the Scratch A register. If a *dataBufferPtr* is not provided (**NULL**), this parameter should be set to **0**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SCSI_INIT_ERROR—SCSI bus not free for initialization
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SCSI_FIFO_CLEAR_FAULT—SCSI and/or DMA FIFOs failed to clear
BIT_SCSI_INTERRUPT_FAULT—SCSI interrupt failed to assert
BIT_SYS_RESTORE_FAILED—unable to restore pre-test system state
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence

SCSI Target Arbitration Test

[**BIT_SCSI_ID**]
tests/scsi/sym895ATests.h

The **tests/scsi/sym895ATests.h** file defines the parameter structures and fields mentioned in the [**BIT_SCSI_ID**] test.

This test verifies the correct operation of the timers on the SYM53C895A SCSI controller.

This test has the following default test values:

Iteration:	1
Duration:	3500
Control:	HALT_ON_ERROR

Test Description

Caution

This test is destructive to the state of the SYM53C895A SCSI controller and must be run before the driver is started.

This test ensures that the handshake-to-handshake, general-purpose, and selection timers operate correctly. The test is configured to cause the target selection to fail due to the target ID being set to a nonexistent or unavailable target. This causes each of the timers to expire, resulting with a time-out interrupt. The user may adjust the time-out period.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/scsi/sym895ATests.h** (**SYM895A_LOOPBACK_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SYM895A_LOOPBACK_TESTS_VALID_FLAG**
timerPeriod = **0**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SCSI_ID** test. The user must use the definition of **SYM895A_LOOPBACK_TESTS_VALID_FLAG**.

timerPeriod

allows the user to adjust the timer period. The parameter must be in the range of **0** to **15**. If set to **0**, the default value of **2**, indicating 200 ms, is used. Shown below is a list of the minimum time-outs associated with the parameter value (refer to the *LSI53C895A PCI to Ultra2 SCSI Controller Technical Manual*, listed in Appendix A, *Related Documentation*).

HTH [3:0] SEL [3:0] GEN [3:0]	**Minimum Time-out (80 MHz Clock) With Scale Factor Bit Cleared**
1	100 µs
2	200 µs
3	400 µs
4	800 µs
5	1.6 ms
6	3.2 ms
7	6.4 ms
8	12.8 ms
9	25.6 ms
10	51.2 ms
11	102.4 ms

HTH [3:0] SEL [3:0] GEN [3:0]	Minimum Time-out (80 MHz Clock) With Scale Factor Bit Cleared
12	204.8 ms
13	409.6 ms
14	819.2 ms
15	1.6 - s

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SCSI_INIT_ERROR—SCSI bus not free for initialization
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SYS_RESTORE_FAILED—unable to restore pre-test system state
BIT_SCSI_DATA_LINE_FAULT—SCSI bus data lines contain bad data
BIT_SCSI_INTERRUPT_FAULT—SCSI interrupt failed to assert
BIT_SCSI_INTERRUPT_TIMEOUT—SCSI timeout interrupt received
BIT_SCSI_SELECTION_FAULT—SCSI target selection failed
BIT_SCSI_FIFO_CLEAR_FAULT—SCSI and/or DMA FIFOs failed to clear
BIT_SCSI_CONTROL_LINE_FAULT—SCSI bus control lines contain bad data
BIT_SCSI_ARBITRATE_FAULT—SCSI target arbitration fault
BIT_SCSI_TIMER_FAULT—SCSI bus timer(s) indication of timeout failed

SCSI Internal Loopback Test

[BIT_SCSI_INTERNAL LOOPBACK]
tests/scsi/sym895ATests.h

The **tests/scsi/sym895ATests.h** file defines the parameter structures and fields mentioned in the [**BIT_SCSI_INTERNAL LOOPBACK**] test.

This test verifies the transmission and reception of data on the SCSI bus while in loopback mode. SCSI data lines are tested for stuck-at faults using the SCSI SCRIPT processor.

This test has the following default test values:

Iteration:	1
Duration:	4000
Control:	HALT_ON_ERROR

Test Description

Caution

This test is destructive to the state of the SYM53C895A SCSI controller and must be run before the driver is started.

This test first verifies that the initiator can select the target in loopback mode. The SCSI SCRIPTS processor then executes a block move of data from memory to fill the DMA FIFO. This data transfers to the SCSI bus. The data transferred across the SCSI bus is read from the SCSI input data latch (SIDL) register and saved to a known memory location. The data is then read from this location and if the read data fails to verify or a time-out occurs, an error is reported. The test provides two methods of testing the SCSI bus. A data buffer may be provided for the test or otherwise a predefined set of patterns is used: 0xFFFFFFFF, 0xF00FF00F, 0xCC33CC33, 0xAA55AA55, 0x55AA55AA, and 0x00000000.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/scsi/sym895ATests.h** (**SYM895A_LOOPBACK_PARAMS**).

This test's default parameters are as follows:
validParamsFlag = **SYM895A_LOOPBACK_TESTS_VALID_FLAG**
dataBufferPtr = **NULL**
numBytes = **0**
enableWideScsi = **FALSE**

The contents of each structure and their effects on the test are discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SCSI_INTERNAL_LOOPBACK** test. The user must use the definition of **SYM895A_LOOPBACK_TESTS_VALID_FLAG**.

dataBufferPtr

allows the user to provide data patterns by setting the parameter to the address of the test data buffer. To indicate that a data buffer has not been provided, set this parameter to **NULL**. If **NULL**, the predefined test patterns are used by this test.

numBytes

specifies the number of bytes contained in the test data buffer, pointed to by *dataBufferPtr*. The buffer length must not exceed the size of the DMA FIFO (944 bytes). If a *dataBufferPtr* is not provided (**NULL**), this parameter should be set to **0**.

enableWideScsi

allows the user to configure the SYM53C895A SCSI controller to transfer data in 16-bit wide SCSI mode. If the value of this parameter is **TRUE**, the data transfer is 16 bits. If the value is **FALSE**, the data transfer is 8 bits.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SCSI_INIT_ERROR—SCSI bus not free for initialization
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SYS_RESTORE_FAILED—unable to restore pre-test system state
BIT_SCSI_DATA_LINE_FAULT—SCSI bus data lines contain bad data
BIT_SCSI_INTERRUPT_FAULT—SCSI interrupt failed to assert
BIT_SCSI_INTERRUPT_TIMEOUT—SCSI timeout interrupt received
BIT_SCSI_SELECTION_FAULT—SCSI target selection failed
BIT_SCSI_FIFO_CLEAR_FAULT—SCSI and/or DMA FIFOs failed to clear
BIT_SCSI_CONTROL_LINE_FAULT—SCSI bus control lines contain bad data
BIT_SCSI_BMOVE_TIMEOUT—SCSI block move timed out
BIT_DATA_MISCOMPARE—data miscompare on write and read sequence

SCSI Parity Detection Test

[**BIT_SCSI_PARITY**]
tests/scsi/sym895ATests.h

The **tests/scsi/sym895ATests.h** file defines the parameter structures and fields mentioned in the [**BIT_SCSI_PARITY**] test.

This test verifies that a SCSI parity interrupt is correctly indicated.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

Caution

This test is destructive to the state of the SYM53C895A SCSI controller and must be run before the driver is started.

This test asserts even SCSI parity. This forces a SCSI parity error on each byte sent to the SCSI bus from the processor while in loopback mode. The test performs a block move SCSI SCRIPT instruction to transmit the data across the SCSI bus. Since the test has asserted even parity, the data transferred causes a SCSI parity interrupt.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/scsi/sym895ATests.h** (**SYM895A_LOOPBACK_PARAMS**).

This test's default parameter is as follows:
validParamsFlag = **SYM895A_LOOPBACK_TESTS_VALID_FLAG**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SCSI_PARITY** test. The user must use the definition of **SYM895A_LOOPBACK_TESTS_VALID_FLAG**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SCSI_INIT_ERROR—SCSI bus not free for initialization
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SYS_RESTORE_FAILED—unable to restore pre-test system state
BIT_SCSI_DATA_LINE_FAULT—SCSI bus data lines contain bad data
BIT_SCSI_INTERRUPT_FAULT—SCSI interrupt failed to assert
BIT_SCSI_INTERRUPT_TIMEOUT—SCSI timeout interrupt received
BIT_SCSI_SELECTION_FAULT—SCSI target selection failed
BIT_SCSI_FIFO_CLEAR_FAULT—SCSI and/or DMA FIFOs failed to clear
BIT_SCSI_CONTROL_LINE_FAULT—SCSI bus control lines contain bad data

BIT_SCSI_NO_PARITY_FAULT—SCSI parity error interrupt failed to assert

SCSI Illegal Instruction Detection Test

[**BIT_SCSI_INSTRUCTIONS**]
tests/scsi/sym895ATests.h

The **tests/scsi/sym895ATests.h** file defines the parameter structures and fields mentioned in the [**BIT_SCSI_INSTRUCTIONS**] test.

This test verifies that the illegal instruction interrupt is properly asserted and de-asserted.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

Caution

This test is destructive to the state of the SYM53C895A SCSI controller and must be run before the driver is started.

This test points the SCSI scripts processor to an illegal instruction and verifies that the illegal instruction interrupt asserts. The test then clears the interrupt indicator and points the SCSI scripts processor to a block move SCRIPT instruction. The test verifies that the illegal instruction interrupt de-asserts and the DMA transfer successfully completes.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

The MBIT API passes test-specific parameters by reference. The test parameter structure is defined in **tests/scsi/sym895ATests.h** (**SYM895A_LOOPBACK_PARAMS**).

This test's default parameter is as follows:
validParamsFlag = **SYM895A_LOOPBACK_TESTS_VALID_FLAG**

The contents of the structure and its effects on the test is discussed below:

validParamsFlag

is provided in an attempt to ensure that the user is providing the right data structure to the **BIT_SCSI_INSTRUCTION** test. The user must use the definition of **SYM895A_LOOPBACK_TESTS_VALID_FLAG**.

This test will ignore any other fields of the parameter structure.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_SCSI_INIT_ERROR—SCSI bus not free for initialization
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_SYS_RESTORE_FAILED—unable to restore pre-test system state
BIT_SCSI_DATA_LINE_FAULT—SCSI bus data lines contain bad data
BIT_SCSI_INTERRUPT_FAULT—SCSI interrupt failed to assert
BIT_SCSI_INTERRUPT_TIMEOUT—SCSI timeout interrupt received

BIT_SCSI_SELECTION_FAULT—SCSI target selection failed
BIT_SCSI_FIFO_CLEAR_FAULT—SCSI and/or DMA FIFOs failed to clear
BIT_SCSI_NO_ILLEGAL_FAULT—SCSI illegal instruction interrupt failed to assert
BIT_SCSI_CONTROL_LINE_FAULT—SCSI bus control lines contain bad data

Flash Memory Tests 17

This chapter provides descriptions and requirements for the following Flash memory tests:

Flash Memory Tests

These tests detect any stuck-at or floating conditions on the data lines connected to the Flash memory modules.

The table below highlights each tests' string and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_FLASH_STUCK]	None	None
[BIT_FLASH_FLOAT]	None	None

Flash Stuck Bit Test

[BIT_FLASH_STUCK]

This test reads the Flash memory data until it detects that all bits have toggled at least once and return success.

This test has the following default test values:

Iteration:	1
Duration:	5000
Control:	HALT_ON_ERROR

Test Description

This test begins reading the Flash memory data beginning at the base address defined in the BSP for the Flash module. It then continues with 32-bit reads of data, tracking the bits that toggle. When the test detects that all bits have toggled at least once, it then returns success. If the test reaches the end of the Flash memory and all of the data lines have not toggled, the test then returns an error code indicating that either there was a stuck-at condition on one of the data lines or there was insufficient variability in the Flash data to determine the integrity of the data lines. Because the test only reads from the Flash memory, the data is unchanged and the state of the Flash memory is preserved.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_DEVICE_ENABLE_FAULT—failed to enable a disabled device
BIT_BUS_ERROR—device did not respond to transfer
BIT_DEVICE_DISABLE_FAULT—failed to disable a enabled device
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable

BIT_FLASH_STUCK_ERROR—possible stuck bit was detected, or insufficient variability in the Flash data

Flash Float Bit Test

[**BIT_FLASH_FLOAT**]

This test detects a floating condition on the data lines connected to the Flash memory modules.

This test has the following default test values:

Iteration:	1
Duration:	10000
Control:	HALT_ON_ERROR

Test Description

This test reads the Flash memory data into a buffer. The data is read, beginning at the base address defined in the BSP, and continues for the memory size also defined in the BSP. The Flash memory data is then reread and compared against the original value stored in the buffer. Any data mis-compare causes an error to indicate a possible floating data line. Because the test only reads from the Flash memory, the data is unchanged and the state of the Flash memory is preserved.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_DEVICE_ENABLE_FAULT—failed to enable a disabled device
BIT_BUS_ERROR—device did not respond to transfer
BIT_DEVICE_DISABLE_FAULT—failed to disable a enabled device
BIT_INIT_ALLOCATION_ERROR—required resources for initialization are unavailable
BIT_FLASH_FLOAT_ERROR—possible floating bit was detected

DS1621 Thermometer Tests 18

This chapter provides descriptions and requirements for the following DS1621 thermometer tests:

DS1621 Thermometer Tests

The tests for the DS1621 test a number of different areas. These tests detect stuck bits in the registers, read and write time-outs that could occur, and also test that the thermostat is behaving correctly when the temperature breaks either of the thresholds.

The table below highlights each tests' string and parameter structure for quick access.

Test String	**Header File**	**Parameter Structure**
[BIT_THERMOMETER_READ_TEMP]	None	None
[BIT_THERMOMETER_ACCESS_TH]	None	None
[BIT_THERMOMETER_ACCESS_TL]	None	None
[BIT_THERMOMETER_ACCESS_CONFIG]	None	None
[BIT_THERMOMETER_READ_COUNTER_ SLOPE]	None	None
[BIT_THERMOMETER_ALARM_TEST]	None	None

Read Temperature Test

[**BIT_THERMOMETER_READ_TEMP**]

This test ensures that the temperature read after the start convert T is within the allowed range.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

This test ensures that the temperature read is between –55° and +125° Celsius, which is specified by the *DS1621 Thermometer Data Sheet*, listed in Appendix A, *Related Documentation*. If the temperature is outside of the range, an error is reported.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied

BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_THERMOMETER_I2C_RW_FAULT—Thermometer I^{2}C read/write fault
BIT_THERMOMETER_RANGE_FAULT—Thermometer thermal range fault
BIT_THERMOMETER_REGISTER_FAULT—Thermometer register fault

Access TH Command Test

[**BIT_THERMOMETER_ACCESS_TH**]

This test checks for stuck bits and read/write errors upon use of the access TH command, excluding the read-only bits.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The TH register only uses the nine most significant bits of the two bytes, therefore the remaining bits are not tested. The test writes various data patterns on the I^{2}C data bus to the TH register. The patterns are as follows: 0xFFFF, 0xFF00, 0xF0F0, 0xCCCC, 0xAAAA, 0x3333, 0x5555, 0x00FF, 0x0F0F, 0x0000. The patterns have the seven least significant bits masked. For each pattern, the test reads back from the register and verifies the contents are equivalent to the value written to it. If the test fails to verify on any of these patterns, an error is reported.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_THERMOMETER_I2C_RW_FAULT—Thermometer I^2C read/write fault
BIT_THERMOMETER_REGISTER_FAULT—Thermometer register fault

Access TL Command Test

[**BIT_THERMOMETER_ACCESS_TL**]

This test checks for stuck bits and read/write errors upon use of the access TL command, excluding the read-only bits.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The TL register only uses the nine most significant bits of the two bytes, therefore the remaining bits are not tested. The test writes various data patterns on the I^2C data bus to the TL register. The patterns are as follows: 0xFFFF, 0xFF00, 0xF0F0, 0xCCCC, 0xAAAA, 0x3333, 0x5555, 0x00FF, 0x0F0F, 0x0000. The patterns have the seven least significant bits masked. For each pattern, the test reads back from the register and verifies the contents are equivalent to the value written to it. If the test fails to verify on any of these patterns, an error is reported.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_THERMOMETER_I2C_RW_FAULT—Thermometer I^2C read/write fault
BIT_THERMOMETER_REGISTER_FAULT—Thermometer register fault

Access Config Command Test

[**BIT_THERMOMETER_ACCESS_CONFIG**]

This test checks for stuck bits and read/write errors upon use of the access config command, excluding the read-only bits.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

The test writes various data patterns on the I^{2}C data bus to the configuration register. For each pattern, the test reads back from the register and verifies the contents are equivalent to the value written to it. If the test fails to verify on any of these patterns, an error is reported. The patterns are as follows: 0xFF 0xF0, 0xCC, 0xAA, 0x55, 0x33, 0x0F, 0x00. The patterns have been chosen in a way so that all the bits get toggled. The patterns are masked in the code so that the read-only bits in the register are not toggled for testing.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_THERMOMETER_I2C_RW_FAULT—Thermometer I^{2}C read/write fault
BIT_THERMOMETER_REGISTER_FAULT—Thermometer register fault

Read Counter Slope Test

[**BIT_THERMOMETER_READ_COUNTER_SLOPE**]

This test verifies the value of read counter is less than the value of read slope when the start convert T command is issued.

This test has the following default test values:

Iteration:	1
Duration:	1000
Control:	HALT_ON_ERROR

Test Description

This test first issues the start convert T command and then reads the value from the read temperature register, then the read counter and finally the read slope register. If the value of the read counter register is less than the read slope, then **BIT_NO_FAULT_DETECTED** is returned. Otherwise an error is returned.

Note The reason for reading the temperature, the read counter and then the read slope is described in the *DS1621 Thermometer Data Sheet*, listed in Appendix A, *Related Documentation*.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field (configuration error)
BIT_THERMOMETER_I2C_RW_FAULT—Thermometer I^2C read/write fault
BIT_THERMOMETER_REGISTER_FAULT—Thermometer register fault

Tout Test

[**BIT_THERMOMETER_ALARM_TEST**]

This test checks the functionality of the thermal output signal.

This test has the following default test values:

Iteration:	1
Duration:	2000
Control:	HALT_ON_ERROR

Test Description

The test begins by issuing the start convert T command. Next, the test reads the current temperature using the read temperature command. After reading the temperature, then the value of the TH register is purposely set to be lower than the current temperature read. The test then verifies that an interrupt occurred by doing a **semTake** and checking the THF bit in the configuration register when returning from the ISR. If the THF bit is not set or control was never transferred to the ISR, then an error is returned. Inside the ISR the interrupt is disabled. After control returns to the main line code the interrupt must be re-enabled. Next, the test verifies that the tout deactivates when the temperature drops below the value in the TL register. To accomplish this task, the test sets the TH register significantly greater than the current temperature and sets the value in the TL register higher than the current temperature, but less than the value in the TH register. At this point, the TLF bit in the configuration register should be set high and the tout should deactivate. If the tout does not deactivate or the TLF bit does not get set, then the test returns an error.

Affected Peripheral Devices

This test does not affect any peripheral devices.

Required Test Equipment

This test does not require any test equipment.

Test Specific Parameters

This test does not require, nor does it use, any test parameters. Therefore, the test parameter pointer must be **NULL**.

Return Values

BIT_SUBTEST_NOT_SUPPORTED—selected subtest is not supported on this device
BIT_DEVICE_NOT_SUPPORTED—device is not supported
BIT_NO_FAULT_DETECTED—no fault detected, successful
BIT_INVALID_TEST_PARAM—invalid test parameter was supplied
BIT_INVALID_DEVICE_DESC—device descriptor has invalid field

(configuration error)
BIT_THERMOMETER_I2C_RW_FAULT—Thermometer I^{2}C read/write fault
BIT_THERMOMETER_REGISTER_FAULT—Thermometer register fault
BIT_THERMOMETER_INACTIVE_FAULT—Thermometer inactive alarm fault
BIT_THERMOMETER_RANGE_FAULT—Thermometer thermal range fault
BIT_THERMOMETER_ALARM_FAULT—Thermometer thermal alarm fault

Related Documentation A

Motorola Computer Group Documents

The Motorola publications listed below are referenced in this manual. You can obtain paper or electronic copies of Motorola Computer Group publications by:

❏ Contacting your local Motorola sales office

❏ Visiting Motorola Computer Group's World Wide Web literature site, http://www.motorola.com/computer/literature

Table A-1. Motorola Computer Group Documents

Document Title	**Motorola Publication Number**
Motorola Built-In Test (MBIT) Diagnostic Software User's Manual	MBITA/UM
MVME5100 Single Board Computer Installation and Use	V5100A/IH
MVME5100 Single Board Computer Programmer's Reference Guide	V5100A/PG
IPMC712/761 Module Installation and Use	VIPMCA/IH
MVME712M Transition Module Installation and Use	MVE712MA/IH
MVME761 Transition Module Installation and Use	VME761A/IH

To obtain the most up-to-date product information in PDF or HTML format, visit http://www.motorola.com/computer/literature.

Manufacturers' Documents

For additional information, refer to the following table for manufacturers' data sheets or user's manuals. As an additional help, a source for the listed document is provided. Please note that, while these sources have been verified, the information is subject to change without notice.

Table A-2. Manufacturers' Documents

Document Title and Source	**Publication Number**
PowerPCTM Microprocessor Family: The Programming Environment for 32-Bit Microprocessors Literature Distribution Center for Motorola Telephone: 1-800- 441-2447 FAX: (602) 994-6430 or (303) 675-2150 WebSite: http://e-www.motorola.com/webapp/DesignCenter/ E-mail: ldcformotorola@hibbertco.com OR IBM Microelectronics Programming Environment Manual Web Site: http://www.chips.ibm.com/techlib/products/powerpc/manuals	MPCFPE/AD G522-0290-01
LSI53C895A PCI to Ultra2 SCSI Controller Technical Manual LSI Logic Corporation http://www.lsilogic.com/techlib/techdocs/storage_stand_prod/PCISCSICont/Chips/895a.pdf	v2.1
DS1621 Thermometer Data Sheet Dallas Semiconductor http://www.dalsemi.com	DS1621

URLs

The following URLs (uniform resource locators) may provide helpful sources of additional information about this product, related services, and development tools. Please note that, while these URLs have been verified, they are subject to change without notice.

- ❏ Motorola Computer Group, http://www.motorola.com/computer
- ❏ Motorola Computer Group OEM Services, http://www.motorola.com/computer/support
- ❏ Wind River Systems, Inc., http://www.windriver.com

Index